UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

"UM MECANISMO PARA TROCA DE PROTOCOLOS DE SINCRONIZAÇÃO DE SIMULAÇÃO DISTRIBUÍDA EM TEMPO DE EXECUÇÃO"

JOÃO CARLOS DE MORAES MORSELLI JUNIOR

Tese apresentada ao Instituto de Física de São Carlos, Universidade de São Paulo, para obtenção do título de Doutor em Ciências "Física Aplicada – Opção Física Computacional"

Orientadora:
Profa. Dra. Regina Helena Carlucci Santana

USP/IFSC/SBI
8-2-001376

Comissão Julgadora:
Profa. Dra. Regina Helena Carlucci Santana (ICMC-USP)
Prof. Dr. Carlos Antonio Ruggiero (IFSC-USP)
Prof. Dr. Luis Carlos Trevelin (UFSCar)
Profa. Dra. Patrícia Magna (COCRP)
Prof. Dr. Nandamudi Lankalapalli Vijaykumar (INPE)

São Carlos – São Paulo
2000

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

Morselli Jr, João Carlos de Moraes

"Um mecanismo para troca de protocolos de sincronização de simulação distribuída em tempo de execução"/ João Carlos de Moraes Morselli Junior – São Carlos, 2000

Tese (Doutorado) – Instituto de Física de São Carlos da Universidade de São Paulo, 2000 – páginas: 227
Área: Física Aplicada-Opção Física Computacional
Profa. Dra. Regina Helena Carlucci Santana

1. Protocolos de sincronização, 2. Simulação distribuída, 3. Tempo de execução

I. Título

UNIVERSIDADE DE SÃO PAULO
Instituto de Física de São Carlos

Caixa Postal 369 - CEP 13560-970 - São Carlos - SP
Fone: (0xx16) 273-9777 - Fax: (0xx16) 273-9777
E-mail: wladerez@if.sc.usp.br

MEMBROS DA COMISSÃO JULGADORA DA TESE DE DOUTORADO DE JOÃO CARLOS DE MORAES MORSELLI JÚNIOR APRESENTADA AO INSTITUTO DE FÍSICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM 25 DE SETEMBRO DE 2000.

COMISSÃO JULGADORA:

Profa. Dra. Regina Helena Carlucci Santana/ICMC-SCE-USP

Prof. Dr. Carlos Antonio Ruggiero/IFSC-FFI-USP

Prof. Dr. Luis Carlos Trevelin/UFSCar

Profa. Dra. Patrícia Magna/COCRP

Prof. Dr. Nandamudi Lankalapalli Vijaykumar/INPE

AGRADECIMENTOS

A Deus, por tudo que tenho realizado na minha vida.

À Profa. Dra. **Regina Helena Carlucci Santana**, minha amiga e orientadora, que com extrema competência sempre me mostrou pacientemente, desde o mestrado , o "caminho das pedras". Muito obrigado.

A minha família, cujo amor e compreensão são suficientes para a realização de "vários doutorados". Obrigado.

A minha família de Vera Cruz, que com incrível carinho sempre me acolheu nos bons e maus momentos. Obrigado.

Ao amigo e Prof. Dr. **Marcos José Santana**, pela amizade e por ter sido a primeira pessoa na USP a acreditar na minha capacidade. Muito Obrigado.

Ao amigo e Prof. Dr. **Odelar Leite Linhares**, um exemplo de profissional e pessoa humana. Muito Obrigado.

Aos Professores da USP com os quais eu aprendi muito durante o período em que convivi. Muito Obrigado.

Às amigas Roberta e Renata, pelas preciosas colaborações e pela gratificante amizade. Obrigado.

À Patrícia e ao Maurício, respectivamente diretora e coordenador da Faculdade de Ciências Exatas e Tecnológicas da Unimar e aos colegas professores, pela compreensão e pela amizade.

Aos meus amigos Marcelo e Renata, pelo carinho das nossas amizades e pela força. Que venham as viagens...

À Prof.ª Mariê pelo valioso auxílio na realização dos Testes de Hipóteses.

Aos amigos do Grupo de Sistemas Distribuídos e Programação Concorrente, pela amizade e disposição em ajudar, com grande vontade, em qualquer tipo de situação.

À todas as pessoas que direta e indiretamente contribuíram para a realização deste trabalho.

À Capes e Universidade de Marília, pelo auxílio financeiro.

Índice

CAPÍTULO 1: INTRODUÇÃO

1.1 Introdução 1

CAPÍTULO 2: COMPUTAÇÃO PARALELA e PLATAFORMAS NÃO CONVENCIONAIS

2.1 Introdução 10

2.2 Conceitos Básicos Sobre Computação Paralela 13

2.3 Arquiteturas Paralelas 20

2.3.1 Classificação de Flynn 21

2.3.2 Classificação de Duncan 25

2.3.2.1 Arquiteturas Síncronas 25

2.3.2.2 Arquiteturas Assíncronas 28

2.4 Ferramentas para Construção de Algoritmos Paralelos 31

2.5 Programação Paralela Utilizando Ambientes de Passagem de Mensagens 36

2.5.1 Computação Paralela em Sistemas Distribuídos 39

2.6 Conclusões 41

CAPÍTULO 3: SIMULAÇÃO DISTRIBUÍDA

3.1 Introdução 43

3.2 Limitações da Simulação Seqüencial 45

3.3 Esquema Básico de Sincronização para Simulação Distribuída Via Troca de Mensagens 47

3.3.1 Lookahead 48

3.3.2 Lookahead Tempos Característicos da Passagem de um Evento pelo Processos Lógicos 51

3.4 Protocolos de Sincronização para Simulação Distribuída 54

3.4.1 Protocolos de Sincronização Conservativos 56
3.4.1.1 Sincronização Lock-Step Time-Driven 56
3.4.1.2 Sincronização Fixed Time-Bucket 57
3.4.1.3 Sincronização Chandy - Misra – Bryant (CMB) 58
3.4.1.3.1 Resolução de *Deadlock* Usando Mensagens Nulas 60
3.4.2 Protocolos de Sincronização Otimistas 62
3.4.2.1 Sincronização Time Warp 63
3.4.2.2 Sincronização Breathing Time Buckets 67
3.4.2.3 Sincronização Breathing Time Warp 70
3.5 Conclusões 71

CAPÍTULO 4: MECANISMO PARA TROCA DINÂMICA DE PROTOCOLOS DE SINCRONIZAÇÃO

4.1 Introdução 73
4.2 Esquema Básico para Simulação Distribuída Via Troca de Mensagens 77
4.2.1 Protocolo CMB 80
4.2.2 Protocolo Time Warp 83
4.2.3 Protocolos Conservativos Versus Otimistas e Protocolos Híbridos 85
4.3 Mecanismo de Troca de Protocolos de Sincronização para Simulação Distribuída 88
4.3.1 Garantia de Consistência 92
4.3.2 Mecanismo de Conversão 95
4.3.2.1 Estruturas de Dados 96
4.3.2.2 Canais de Comunicação 105
4.3.2.3 Variáveis de Tempo 108
4.4 Algoritmo dos Processos do Mecanismo de Troca de Protocolos 113
4.4.1 Algoritmo do Processo Observador 113
4.4.2 Algoritmo do Processo Gerenciador 114
4.4.3 Algoritmo do Processo Conversor 115
4.5 Conclusões 117

CAPÍTULO 5: MODELO E IMPLEMENTAÇÃO DE UM SISTEMA DE SIMULAÇÃO PARA O ESTUDO DO COMPORTAMENTO DO PROTOCOLO CMB

5.1 Introdução 123

5.2 Simulation Program Language (SMPL) 124

5.2.1 Estruturas do SMPL 125

5.3 Modelo da Simulação Utilizada para Análise do Comportamento do Protocolo CMB 129

5.3.1 Estrutura Global do Modelo de Simulação 130

5.3.2 Modelagem e Simulação das Características do Protocolo CMB 133

5.3.3 Modelo da Plataforma 140

5.3.3.1 Plataformas Representativas 142

5.3.4 Modelo da Aplicação 150

5.3.4.1 Modelos de Aplicação Utilizados 151

5.3.4.2 Características de Implementação do Modelo da Aplicação 155

5.4 Métodos Utilizados para a Validação 158

5.4.1 Validação do Modelo para o Protocolo CMB 158

5.4.2 Validação do Modelo da Plataforma e da Aplicação 159

5.5 Estratégia de Execução 160

5.6 Conclusões 162

CAPÍTULO 6: COMPORTAMENTO DO PROTOCOLO CMB:ANÁLISE DOS RESULTADOS

6.1 Introdução 166

6.2 Variáveis de Desempenho e Medidas Estatísticas Utilizadas para Análise dos Resultados 167

6.2.1 Parâmetros de Entrada 168

6.2.2 Parâmetros de Saída 170

6.3 Análise Individual 172

6.3.1 Relação entre a Quantidade de Mensagens Nulas e a Probabilidade de Bloqueio 172

6.3.2 Influência da Probabilidade de Bloqueio do Modelo e das Mensagens Nulas em Relação ao Tempo Simulado 176

6.3.3 Influência das Mensagens Nulas no Desempenho dos Recursos da Simulação 181

6.4 Análise Comparativa 185

6.4.1 Influência do Desempenho dos Processadores e do Meio Físico em Relação ao Tempo de Simulação 186

6.4.2 Probabilidade de Bloqueio e Característica da Plataforma 190

6.4.3 Relação entre a Prob. de Bloqueio do Modelo da Aplicação e a Plataforma na Quantidade de Transmissões de Mensagens Nulas 194

6.4.4 Análises Complementares sobre as Características do Comportamento do Protocolo CMB 201

6.5 Conclusões 202

CAPÍTULO 7: CONCLUSÕES FINAIS

7.1 Introdução 207

7.2 Conclusões Finais 211

7.2.1 Mecanismo de Troca de Protocolos 211

7.2.2 Estudo do Comportamento do Protocolo CMB para Realização da Troca de Protocolos 212

7.3 Contribuições 222

7.4 Propostas para Trabalhos Futuros 225

APÊNDICE A: ANÁLISE ESTATÍSTICAS 228

APÊNDICE B: CÓDIGO DAS SIMULAÇÕES 235

APÊNDICE C: ESTUDO DO COMPORTAMENTO DO PROTOCOLO CMB, RESULTADOS ADICIONAIS 259

REFERÊNCIAS BIBLIOGRÁFICAS 268

Lista de Figuras

Figura 2.1 Tipos de Paralelismo 15
Figura 2.2 Arquitetura SISD 22
Figura 2.3 Arquitetura SIMD 23
Figura 2.4 Arquitetura MISD 24
Figura 2.5 Arquitetura MIMD 26
Figura 2.6 Classificação de Duncan para Arquiteturas Paralelas. 27
Figura 2.7 Paradigma de Passagem de Mensagem 36

Figura 3.1 Sistema de Fila FIFO 49
Figura 3.2 Processo Lógico do Exemplo do Sistema de Fila Simples 50
Figura 3.3 Tempos Característicos para as Tarefas 52
Figura 3.4 *Lookahead* para o Exemplo do Servidor de Fila Única 53
Figura 3.5 Processamento Otimista de Eventos 64
Figura 3.6 *RollBack* no Time Warp 66
Figura 3.7 Utilização de antimensagens no Time Warp 67
Figura 3.8 Horizonte de Eventos 69

Figura 4.1 Valor do *clock* de um processo 79
Figura 4.2 Processo Lógico CMB 82
Figura 4.3 Processo Lógico Time Warp 87
Figura 4.4 Estrutura dos processos envolvidos no sistema de simulação utilizando a troca dinâmica de protocolos 90
Figura 4.5. Mecanismo de ativação dos processos envolvidos no Mecanismo de Troca 91
Figura 4.6 Mensagem Utilizada pelo Mecanismo de Troca de Protocolos 99

Figura 4.7 Fluxo de Mensagens durante a Troca CMB para Time Warp 102
Figura 4.8 Troca de Protocolos Considerando os Canais de Comunicação 107
Figura 4.9 Variáveis de Tempo Utilizadas nos Protocolos CMB e Time Warp 108
Figura 4.10 Mecanismo de Troca: Variáveis de Tempo Utilizadas no Modo CMB 110
Figura 4.11 Mecanismo de Troca: Variáveis de Tempo Utilizadas no Modo Time Warp 111
Figura 4.12 Processo Lógico Utilizado pelo Mecanismo de Troca 119

Figura 5.1 Elemento básico de dados 126
Figura 5.2 Composição do Modelo Geral para Análise do CMB 132
Figura 5.3 Rotina *atualizar_gvt()* 138
Figura 5.4 Código que garante o princípio conservativo 139
Figura 5.5 Construção do Modelo da Plataforma 142
Figura 5.6 Relação entre o desempenho das Plataformas Utilizadas 145
Figura 5.7 Modelo de Redes de Filas: Servidor Central 153
Figura 5.8 Modelo de Redes de Filas: Sistema Computacional Simplificado 156
Figura 5.9 Validação do Modelo do Protocolo CMB 159
Figura 5.10 Estratégia de Execução das Simulações 162

Figura 6.1 Parâmetros de Entrada e Saída 168
Figura 6.2 Tabela de Configuração 173
Figura 6.3 Probabilidade de Bloqueio x % Nulas: Arquitetura AA 174
Figura 6.4 Probabilidade de Bloqueio x % Nulas: Modelo SC 175
Figura 6.5 Probabilidade de Bloqueio x Tempo Simulado: Arquitetura AA 177
Figura 6.6 Probabilidade de Bloqueio x Tempo Simulado: Arquitetura CRAY 180
Figura 6.7 Tempo Simulado x % Nulas: Arquitetura CRAY Modelo SC 181
Figura 6.8 Tempo Simulado x % Nulas: Arquitetura CRAY Modelo SCS 182
Figura 6.9 Utilização da CPU x % Nulas: Plataforma AA 184
Figura 6.10 Tempo Simulado x % Nulas: Plataforma AA 184
Figura 6.11 Utilização da CPU x % Bloqueio: Plataforma AA 185
Figura 6.12 Gráfico Comparativo: Tempo Simulado x Ciclos 187

Figura 6.13 Arquiteturas com o mesmo processador e com meios de comunicação distintos 188

Figura 6.14 Gráfico Comparativo: Tempo Simulado x Ciclos 189

Figura 6.15 Tempo Simulado x % Bloqueio: 10000 Ciclos 192

Figura 6.16 Tempo Simulado x % Bloqueio: 50000 Ciclos 193

Figura 6.17 Tempo Simulado x % Bloqueio: Plataformas AA e AB 193

Figura 6.18 Tempo Simulado x % Bloqueio: 1.0 Tempo entre Chegadas 194

Figura 6.19 Tempo Simulado x % Bloqueio: 50.0 Tempo entre Chegadas 195

Figura 6.20 Mensagens Nulas em Relação à Probabilidade de Bloqueio: 30000 Ciclos 196

Figura 6.21 Mensagens Nulas em Relação à Probabilidade de Bloqueio: 50000 Ciclos 197

Figura 6.22 Emissões de mensagens nulas e completas 198

Figura 6.23 Emissões de mensagens nulas 200

Lista de Tabelas

Tabela 2.1 Ambientes de Passagem de Mensagens. 38

Tabela 3.1 Tempos de Serviço. 50

Tabela 5.1 Arquiteturas IBM e CRAY: Latência e Bandwidth. 144

Tabela 5.2 Arquiteturas Representativas. 145

Tabela 5.3 Tamanho das informações de uma mensagem completa do protocolo CMB. 146

Tabela 5.4 Tempo de Transmissão de uma Mensagem CMB. 147

Tabela 5.5 Tempo de utilização dos processos *cpu* e *discos* em relação ao desempenho dos elementos de processamento. 150

Tabela 5.6 Validação do Modelo do Protocolo CMB. 160

Tabela 6.1 Variação dos tempos de simulação em relação à probabilidade de bloqueio: Plataforma AA. 178

Tabela 6.2 Variações dos tempos de simulação em relação à probabilidade de bloqueio considerando todas as plataformas. 191

Tabela 6.3 Situações Favoráveis à Troca de Protocolos. 203

Lista de Abreviaturas

		Página
CMB	*Chandy – Misra – Bryant*	4
SMPL	*Simulation Program Language*	8
SISD	*Single Instruction stream/Single Data stream*	21
SIMD	*Single Instruction stream/Multiple Data stream*	22
MISD	*Multiple Instruction stream/Single Data stream*	23
MIMD	*Multiple Instruction stream/Multiple Data stream*	24
PVM	*Parallel Virtual Machine*	33
MPI	*Message Passing Interface*	33
SPMD	*Single Program, Multiple Data*	35
MPMD	*Multiple Program, Multiple Data*	35
EP	*Elemento de Processamento*	36
MPP	*Massively Parallel Processor*	37
FDDI	*Fiber Distributed Data Interface*	41
ATM	*Asynchronous Transfer Mode*	41
RPC	*Remote Procedure Call*	47
FIFO	*First In First Out*	48
LVT	*Local Virtual Time*	52
TW	*Time Warp*	63
GVT	*Global Virtual Time*	65
IB	*Input Buffer*	81
OB	*Output Buffer*	81
CC	*Channel Clock*	81
EVL	*Event List*	81
LVTH	*Local Virtual Time Horizon*	82
S	*Status*	83
IQ	*Input Queue*	84
OQ	*Output Queue*	84
SCS	*Modelo Sistema de Computação Simplificado*	173
SC	*Modelo Servidor Central*	173

Resumo

Este trabalho propõe um novo mecanismo que possibilita a escolha entre os protocolos de sincronização para simulação distribuída, *CMB* (Chandy – Misra – Bryant) e *Time Warp*, durante a execução de uma simulação distribuída baseada em redes de filas. A utilização do mecanismo proposto é interessante porque a escolha do protocolo de sincronização mais apropriado é uma tarefa difícil. Desta forma, a troca entre os protocolos, automática e em tempo de execução, torna-se uma opção atrativa.

A troca de protocolos deve considerar duas questões principais: *como* realizar a troca, uma vez que cada protocolo apresenta suas próprias características, e *quando* a troca deve ser considerada.

Baseando-se nessas duas questões, o mecanismo para troca de protocolos proposto neste trabalho pode ser implementado através da execução dos processos lógicos da simulação concorrentemente com os processos: *observador* (responsável pela coleta de dados relativos ao desempenho da simulação), *gerenciador* (detecta a necessidade da troca de protocolos) e o *conversor* (responsável pela conversão entre os protocolos).

Este trabalho apresenta o detalhamento destes processos e um estudo sobre as características do protocolo CMB, que devem ser consideradas para avaliar seu comportamento e uma possível troca para o protocolo Time Warp.

Abstract

This paper addresses a novel mechanism to allow the choice between the CMB and the Time Warp (TW) protocols during the execution of queuing-based model distributed simulations. The mechanism proposed is very attractive because the choice of the most appropriate synchronisation protocol is a difficult task. Many times the use of both, conservative and optimistic protocols, is a good option.

The dynamic swapping of protocols involves two main questions: how to realise the swapping considering the distinct characteristics and how to define the best moment to perform the swapping. Considering these two questions, the mechanism for the protocol swapping, proposed in this work, can be implemented running the following three processes, concurrently with the logical processes: *observer* (responsible for collecting the performance metrics), *management* (responsible for the detection of the correct moment for the protocol swapping) and *converter* (responsible for the protocol swapping).

This work presents the description of these processes. A study about which features of the CMB protocol to avaluate its behaviour and a possible change between the CMB and TW protocol is also presented.

Capítulo 1

Introdução

Um assunto que vem preocupando os projetistas e usuários de sistemas computacionais é a análise de desempenho dos sistemas computacionais. A avaliação de desempenho é um processo essencial, uma vez que esta grandeza indica a qualidade do trabalho executada pelo sistema, sendo assim de vital importância na seleção de sistemas alternativos, no projeto de novos sistemas e na análise entre sistemas existentes.

Diferentes ferramentas podem ser utilizadas na avaliação de desempenho. A princípio, protótipos eram implementados, correspondendo às vezes, à construção de elementos de hardware complicados e caros, acoplados a um grande esforço de desenvolvimento de software. Os protótipos, além de apresentarem um alto custo e grande complexidade, oferecem uma flexibilidade muito limitada.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Uma alternativa consiste nas técnicas de modelagem, que são utilizadas para abstrair as características de um sistema em um modelo. A utilização de métodos analíticos na solução destes modelos pode oferecer uma solução simples em alguns casos, porém, para modelos mais robustos, sua utilização é limitada pela complexidade. Algumas vezes, hipóteses não muito realistas são feitas para que a solução analítica de um modelo se torne viável, comprometendo a credibilidade dos resultados. Entretanto, em situações onde a solução analítica é conhecida, sua utilização é recomendada por apresentar resultados exatos e sem a necessidade de muito esforço computacional.

Uma outra forma de solucionar os modelos e consequentemente avaliar o desempenho de sistemas computacionais mais complexos é a simulação, que constitui uma ferramenta poderosa utilizada em diversas áreas com finalidades diversificadas, devido principalmente à sua grande flexibilidade e ao baixo custo apresentado. Todavia uma simulação realista requer esforço computacional e desta forma o tempo de processamento necessário muitas vezes inviabiliza a obtenção de uma grande quantidade de resultados.

A utilização da simulação seqüencial, para modelos detalhados pode requerer horas ou mesmo dias de processamento computacional. Este problema é crítico quando estão sendo simulados sistemas computacionais.

Este cenário vem mudando com o uso de supercomputadores e de ambientes formados por processadores com grande capacidade de comunicação, onde o paralelismo pode ser explorado.

As técnicas de simulação tiveram também que evoluir neste sentido. A abordagem clássica para simulação seqüencial considera a manipulação de uma estrutura de dados chamada *lista de acontecimentos futuros*, de acordo com os valores de uma variável relógio que representa a passagem do tempo na simulação.

A natureza do mecanismo da lista de acontecimentos futuros imposta pela simulação seqüencial, onde a cada ciclo da simulação apenas um item é removido da lista de acontecimentos, inviabiliza a execução paralela do algoritmo em máquinas com mais de um elemento processador, uma vez que a lista de acontecimentos não pode ser efetivamente particionada. A existência de uma lista de acontecimentos posicionados cronologicamente impõe uma simulação seqüencial onde apenas um evento é simulado por vez .

Para que a abordagem tradicional e seqüencial, inerentemente lenta, desse lugar às técnicas de simulação distribuída que exploram convenientemente o paralelismo, foram necessárias alterações na forma de execução de uma simulação.

A idéia de simulação distribuída foi proposta por Chandy [Chandy et al. 1979] em uma série de conferências na Universidade de Waterloo e, independentemente, por R. E. Bryant [Misra 1986, Nicol; Fujimoto 1995]. A partir desta época foram propostos esquemas básicos onde mesmo não existindo uma estrutura seqüencial, como a lista de acontecimentos futuros, promovem o avanço coerente do tempo da simulação.

Tradicionalmente, uma simulação distribuída pode ser classificada como *Conservativa*, *Otimista* ou *Mista* [Spolon et al. 1999]. Esta classificação é baseada na maneira com que a sincronização dos processos lógicos da simulação é realizada através da execução dos eventos que compõem o modelo simulado.

Em um protocolo de sincronização *Conservativo* os tempos de ocorrência de cada um dos eventos é previamente verificado para que haja a certeza de que nenhum evento seja realizado fora da sua ordem cronológica. Desta forma garante-se que todos os eventos da simulação estão sendo executados conforme a ordem cronológica imposta no modelo real simulado. No protocolo CMB (Chandy – Misra – Bryant) [Chandy; Misra 1979], um típico mecanismo de sincronização conservativo, um processo lógico não pode executar um evento a menos que haja a certeza de que não chegará nenhuma mensagem com um tempo de ocorrência menor. Assim sendo, pode-se considerar que a principal característica dos protocolos conservativos é a ausência da possibilidade de um erro de causalidade. Um erro de causalidade ocorre quando é realizado o processamento de um evento fora da sua ordem correta de execução. O protocolo *CMB* é o principal representante desta categoria de protocolos.

Por outro lado, em um protocolo de sincronização *Otimista*, que tem como principal representante o protocolo *Time Warp* [Fujimoto 1990], cada evento é realizado incondicionalmente sem que haja uma verificação prévia quanto a sua ordem cronológica. Desta forma cada mensagem é recebida pelo processo lógico e depois executada sem que haja uma consulta ao

valor do tempo já simulado pelo processo lógico. Entretanto, quando uma mensagem fora da ordem cronológica é detectada o processo lógico é obrigado a restaurar um estado anterior (consistente) desfazendo-se assim as alterações provocadas pela mensagem prematura. Em seguida, os eventos são executados novamente na ordem correta de tempo de ocorrência.

Finalmente, denominam-se protocolos de sincronização *mistos* aqueles que procuram se beneficiar das vantagens oferecidas pelos dois mecanismos tradicionais (conservativos e otimistas) oferecidas em determinadas situações. Intuitivamente estes protocolos utilizam um "otimismo calculado" que permite explorar os benefícios dos protocolos otimistas sem suas inconveniências. Entretanto, é difícil para o modelador da simulação, que não está intimamente familiarizado com os protocolos de sincronização e com as arquiteturas paralelas, ajustar os parâmetros apropriados para obter um bom desempenho.

A questão relativa aos méritos dos protocolos de sincronização tem sido freqüentemente abordada. Existe um consenso geral de que a formulação de regras para determinar a superioridade de um protocolo em relação a outro é uma tarefa bastante difícil e que envolve uma série de circunstâncias relativas tanto ao modelo que está sendo simulado quanto a plataforma (hardware e software) utilizada [Ulson et al. 1999]. Além disto, a determinação do desempenho apresentado na utilização de um determinado protocolo de sincronização envolve a aplicação de conceitos de arquiteturas e programação paralela que, na maioria das vezes, não são do

conhecimento do usuário de simulação. A avaliação de desempenho de um sistema requer que o avaliador tenha bons conhecimentos tanto da técnica de avaliação que está sendo utilizada quanto do sistema a ser avaliado. Exigir que o avaliador conheça com profundidade os conceitos e características da computação paralela pode inviabilizar a utilização da simulação distribuída.

Diversos trabalhos vêm sendo desenvolvidos no grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP, com o intuito de facilitar a utilização das técnicas de simulação distribuída [Ulson 1999, Spolon et al. 1999]. No entanto, em todos esses trabalhos, o avaliador deve decidir que tipo de protocolo utilizar. Essa não é uma decisão trivial uma vez que uma série de conceitos devem ser considerados.

Desta forma, este trabalho propõe o desenvolvimento de um mecanismo que possibilite a troca entre os protocolos *CMB* e *Time Warp* automaticamente, durante a execução da simulação, denominado nesta tese de *mecanismo para troca dinâmica de protocolos*.

O mecanismo proposto permite que o avaliador de um sistema inicie uma simulação distribuída utilizando qualquer um dos protocolos disponíveis (no presente estudo foram considerados o CMB e o Time Warp). Após o início da simulação, seu desempenho será constantemente analisado e, quando necessário, a troca de protocolos será ativada. Com a utilização do mecanismo para troca de protocolos, a escolha de um protocolo não apropriado será menos crítica. O baixo desempenho da simulação será automaticamente detectado.

Desta forma, um primeiro passo para que exista a troca de protocolos é a detecção de uma situação de baixo desempenho ocasionado pelo protocolo correntemente utilizado para sincronizar a simulação. O processo para detecção da necessidade de troca de protocolos pode ser dividido em duas partes. A primeira, quando o protocolo Time Warp está sendo utilizado. Neste caso devem ser considerados o número de *rollbacks* executados e o número de antimensagens. Um outro trabalho de doutorado em desenvolvimento no grupo de Sistemas Distribuídos e Programação Concorrente vem analisando a influência destas e de outras características no desempenho de uma simulação distribuída utilizando protocolos otimistas [Spolon et al. 1999].

A segunda parte a ser considerada na detecção de baixo desempenho de uma simulação distribuída é quando o protocolo CMB está sendo utilizado. O estudo do comportamento do protocolo CMB é realizado neste trabalho sob diferentes contextos. Através deste estudo pode-se avaliar o impacto de determinadas características relativas ao contexto da simulação (plataforma, modelo de aplicação, características de bloqueio, etc.) e assim identificar situações nas quais este protocolo deve ser substituído pelo protocolo Time Warp, utilizando-se o mecanismo de troca dinâmica.

Para a avaliação do protocolo CMB foram considerados modelos e plataformas representativas de diferentes tipos de sistemas computacionais. A metodologia adotada é baseada na utilização de técnicas de modelagem, que pela sua flexibilidade viabilizaram o estudo do comportamento do protocolo CMB em plataformas não disponíveis. O modelo do protocolo

CMB foi simulado com o auxílio da extensão funcional para simulação *SMPL* escrita na linguagem de programação C [MacDougall 1987].

Para a realização das análises foram desenvolvidos modelos e plataformas representativas de diferentes tipos de sistemas computacionais, constituindo casos de testes que podem ser utilizados como referência para a avaliação de desempenho de diversos tipos de protocolos de sincronização em simulação distribuída.

Uma vez detectado o baixo desempenho do protocolo em utilização, um mecanismo de troca de protocolos deve ser ativado. Este mecanismo envolve a adequação das estruturas de dados, variáveis de contexto e de sincronização. Novamente, nesta etapa, a troca de protocolos deve ser dividida em duas partes: a troca de Time Warp para CMB e a troca de CMB para Time Warp.

Resumindo, o objetivo desta tese de doutorado é especificar um mecanismo que possibilite a troca de protocolos de sincronização durante a execução da simulação distribuída. A troca de protocolos envolve a detecção da necessidade de realização da troca e a adaptação das estruturas envolvidas em cada um dos protocolos considerados.

Os capítulos 2 e 3 desta tese apresentam uma revisão bibliográfica sobre os tópicos mais importantes dentro do contexto do trabalho desenvolvido. Uma vez que a realização de uma simulação distribuída envolve o conhecimento de arquiteturas e programação paralela, o capítulo 2 apresenta um panorama geral sobre os principais conceitos envolvidos

na construção de programas paralelos. O capítulo 3 faz uma revisão sobre as técnicas de simulação distribuída e seus principais protocolos de sincronização.

O capítulo 4 descreve o mecanismo de troca de protocolos considerando as questões relativas às estruturas de dados, variáveis de tempo e utilização dos meios de comunicação dos protocolos CMB e Time Warp. Além disso são discutidos os processos lógicos que auxiliam o mecanismo de troca de protocolos.

No capítulo 5 é apresentada a forma com que foi implementado o modelo de simulação utilizado para a análise do comportamento do protocolo CMB considerando as diferentes plataformas e modelos representativos utilizados.

O capítulo 6 apresenta uma análise dos resultados sobre o estudo do desempenho para a simulação distribuída considerando diferentes modelos de aplicação e as influências das características da plataforma no desempenho da simulação.

O capítulo 7 encerra o trabalho, apresentando as contribuições resultantes, as conclusões finais e os planos para continuidade do trabalho.

Capítulo 2

Computação Paralela e Plataformas Não Convencionais

Este capítulo apresenta os conceitos básicos envolvidos no processo de construção de sistemas paralelos. É feita uma descrição das arquiteturas definidas pelas classificações de Flynn e Duncan. As principais ferramentas utilizadas para a construção de algoritmos paralelos também são discutidas.

2.1 Introdução

O desenvolvimento tecnológico nas diversas áreas de conhecimento vem sendo apoiado por sistemas computacionais cada vez mais rápidos e

com maior capacidade. Por sua vez, grande parte dos avanços tecnológicos tem contribuído para a evolução dos sistemas computacionais, melhorando suas características, diminuindo seus preços, e tornando-os acessíveis a um número maior de pessoas e profissionais. Esse processo vem estimulando o progresso tecnológico de diversos países, na medida em que a maior parte da capacidade computacional tem sido usada na solução de problemas científicos, industriais e militares. Esses problemas abrangem: meteorologia, energia nuclear, aerodinâmica, prospeção de petróleo, eletrônica, pesquisa geral de física, astrofísica, simulação em geral, etc..

Para suportar essas aplicações, há uma crescente necessidade por maior velocidade para superar as restrições de tempo, devido à complexidade computacional dos algoritmos utilizados e/ou ao tamanho do conjunto de dados a serem processados. Nesse sentido, o aumento da velocidade tem ocorrido às custas de pesquisas e desenvolvimentos em áreas como microeletrônica, arquitetura, software e periféricos.

Apesar dos avanços específicos em cada área o paralelismo de maneira geral é que vem aumentando substancialmente a velocidade dos sistemas computacionais. Desta forma, enquanto tecnologias e arquiteturas distintas são usadas no hardware (microeletrônica, pipeline, etc..), técnicas de exploração de paralelismo são empregadas no software.

A computação paralela teve início na década de 70. O Illiac IV, um processador vetorial, tornou-se operacional em 1975; o primeiro Cray-1, um

processador vetorial com *pipeline*[1], surgiu em 1976; e os multiprocessadores com baixo custo não se encontraram disponíveis antes de 1984. O advento da integração em larga escala (VLSI – "Very Large-Scale Integration") anunciou uma nova era na computação, possibilitando não apenas o aparecimento do computador pessoal, mas também o desenvolvimento de dispositivos de computação consistindo em dezenas, centenas e até mesmo milhares de processadores, todos trabalhando em conjunto para realizar uma determinada computação [Quinn 1987].

Hwang [Hwang; Xu 1998] define processamento paralelo como a forma eficiente do processamento de informações, com ênfase na exploração de eventos concorrentes no processo computacional. O processamento paralelo implica na divisão de uma determinada aplicação em partes, de maneira que essas partes possam ser executadas simultaneamente, por vários elementos de processamento. Esses elementos de processamento devem cooperar entre si (comunicação e sincronismo), buscando eficiência, por meio da quebra do paradigma de execução seqüencial do fluxo de instruções, ditado pela filosofia de von Neumann. Vários fatores justificam a necessidade do processamento paralelo e a busca por maior desempenho é o principal entre esses fatores [Almasi; Gottlieb 1994, Quealy 1994]. Além desse fator, outros também motivaram o desenvolvimento da computação paralela [Almasi; Gottlieb 1994, Amorim et al. 1988]. Entre eles a configuração modular permite que

---

[1] Forma de execução de tarefas como uma "cascata", onde execuções diferentes ocorrem com sobreposição e defasagem no tempo.

vários elementos de processamento sejam agrupados de acordo com as características da aplicação e a expansão com a inclusão de novos elementos de processamento. Além disso, a possibilidade de desenvolvimento de sistemas tolerantes a falhas, por intermédio da redundância de *hardware* torna a computação paralela uma ferramenta bastante atrativa.

Os conceitos básicos sobre programação paralela e ambientes de passagem de mensagens são apresentados nas seções a seguir.

2.2 Conceitos Básicos Sobre Computação Paralela

Nesta seção são apresentados conceitos básicos tais como processos, tipos de paralelismo, granulosidade, *speedup*, *deadlock* e comunicação entre processos.

- **Processos Paralelos e Processos Concorrentes**

Um programa seqüencial é determinado pela execução seqüencial de uma lista de comandos: essa execução é chamada de *processo*. Um programa concorrente apresenta dois ou mais processos que são executados concorrentemente. Desta forma, pode-se dizer que existe execução concorrente de processos sempre que, em um determinado instante, dois ou mais processos tiverem sido iniciados mas ainda não

terminados, o que significa que estão em andamento. Os processos em execução podem apresentar paralelismo real ou pseudo-paralelismo [Quinn 1987]. A figura 2.1 ilustra esses tipos de execução e a possibilidade da existência de pseudo-paralelismo e paralelismo real em conjunto.

No pseudo-paralelismo existe a ocorrência intercalada de partes de processos concorrentes, de forma que em qualquer instante, apenas um processo estará em andamento no único recurso disponível. No paralelismo real existe a ocorrência sobreposta dos processos. Todos os processos paralelos estão em andamento simultaneamente, utilizando os múltiplos recursos existentes.

O termo paralelismo é normalmente utilizado para designar paralelismo real. Já o termo concorrência é empregado em casos nos quais se tem mais de um processo iniciado e não ainda terminado [Ulson 1999]. Isso não implica, necessariamente, na existência de mais de um elemento de processamento.

Desta forma, o paralelismo é um caso especial de concorrência, no qual existem processos sendo executados utilizando múltiplos elementos de processamento.

O paralelismo real pode ser classificado em: espacial, temporal (*pipeline*) ou combinado. O paralelismo espacial está ligado à sobreposição de execução de tarefas, podendo ser subdividido em síncrono ou assíncrono.

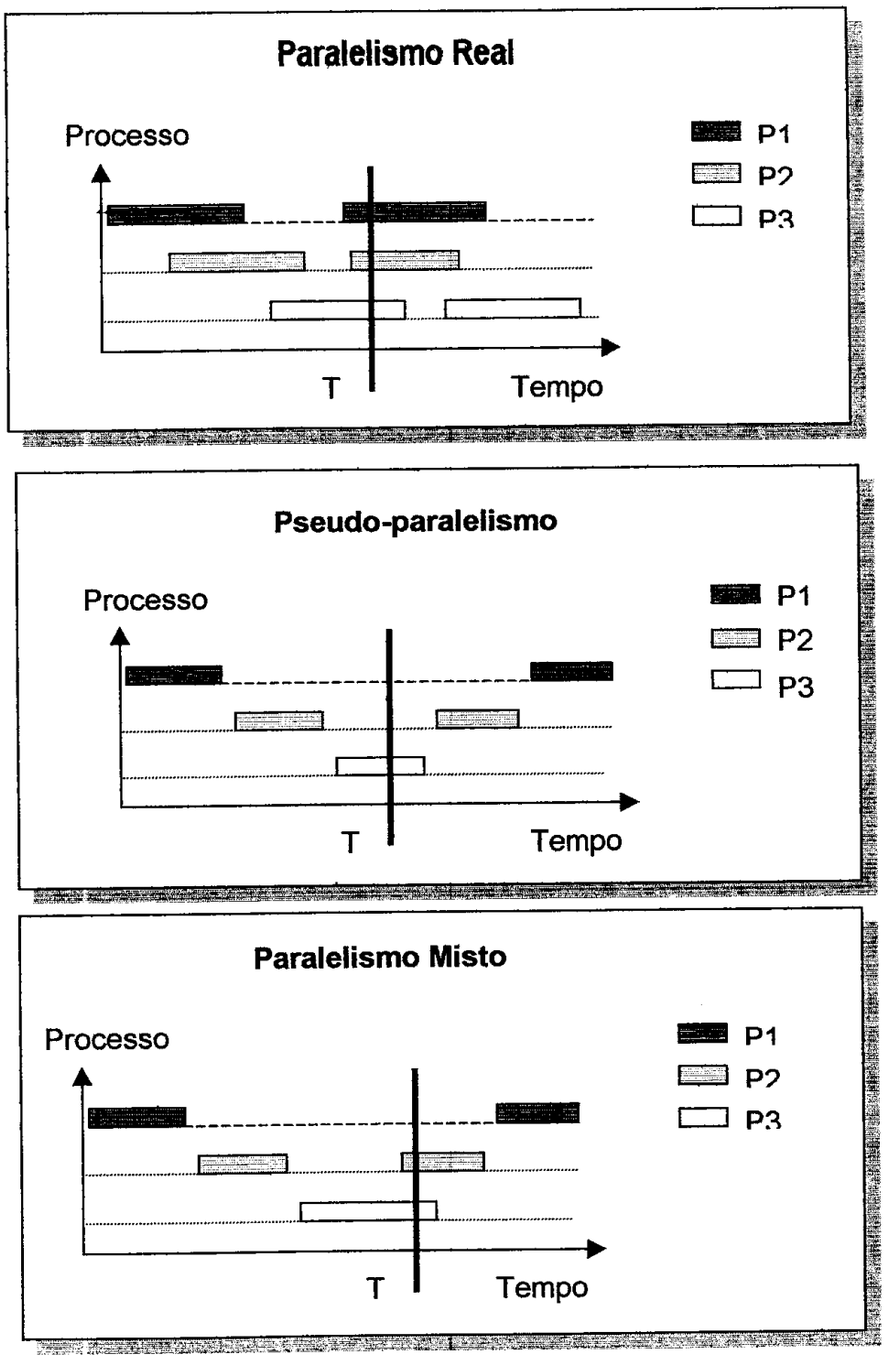


Figura 2.1 Tipos de Paralelismo. Identifica-se o paralelismo real quando existe uma sobreposição de execuções no mesmo instante de tempo.

No paralelismo síncrono vários elementos de processamento executam funções idênticas e são sincronizados por uma única unidade de controle. No paralelismo assíncrono os elementos de processamento executam funções diferentes e interagem entre si.

O paralelismo temporal está relacionado com a execução de tarefas como uma "cascata" ou estágios de sub-tarefas, de forma que execuções diferentes ocorram com sobreposição e defasagem no tempo. A combinação do paralelismo espacial com o paralelismo temporal (paralelismo combinado) permite a execução sobreposta de vários conjuntos de tarefas que, por sua vez, são executadas com defasagem no tempo.

- ***Deadlock***

Uma situação de *deadlock* pode ser formalmente definida, segundo Tanenbaum [Tanenbaum 1992], da seguinte maneira:

"Um conjunto de processos está em *deadlock* se cada processo, pertencente a um conjunto de processos, está esperando por um evento que nunca irá ocorrer pois apenas outro processo do conjunto, que também está esperando por um evento, pode causar".

Uma vez que todos os processos estão esperando pela ocorrência de um evento (ou pela disponibilidade de um determinado recurso) nenhum

deles irá realizar uma ação que faça com que um deles "acorde", e desta forma, todos permanecerão esperando infinitamente.

- **Granulosidade**

A granulosidade de um sistema paralelo corresponde ao tamanho das unidades de trabalho submetidas aos processadores. Isto acaba influenciando na determinação do porte e da quantidade de processadores, uma vez que existe uma relação entre esses dois fatores.

Em uma linguagem seqüencial, a unidade de paralelismo é todo o programa. Em uma linguagem paralela, entretanto, a unidade de paralelismo pode ser definida, em ordem decrescente de granulosidade, como um processo, um objeto, um comando, uma expressão ou uma cláusula [Bal et al. 1989, Hwang 1993].

Enquanto a granulosidade grossa implica em um pequeno número de processadores grandes e complexos, no outro extremo a granulosidade fina envolve um grande número de processadores pequenos e simples.

A distribuição destas unidades entre os processadores é chamada de *mapeamento*. Existem dois tipos mais utilizados de mapeamento: mapeamento estático (fixado em tempo de compilação) e mapeamento dinâmico (fixado em tempo de execução).

- **Speedup**

O principal objetivo na utilização da computação paralela é a diminuição do tempo de execução dos processos envolvidos. Diferentes métricas podem ser utilizadas para determinar se a utilização do processamento paralelo está sendo vantajosa e quantificar o desempenho alcançado. O *speedup* é uma das métricas mais utilizadas para atingir esse objetivo, sendo definido como o aumento de velocidade observado em um computador paralelo com *p* elementos de processamento, em relação a um computador seqüencial, como apresentado pela Eq. (2.1) [Hwang; Xu 1998].

$$Sp = Ts/Tp \qquad (2.1)$$

Onde:

Sp = *speedup* observado em um computador com *p* elementos de processamento;

Ts = tempo de execução do programa em um único elemento de processamento;

Tp = tempo de execução do mesmo programa em um computador paralelo, com *p* elementos de processamento.

Nos casos em que se atinge uma distribuição perfeita dos processos a serem executados em paralelo, obtém-se o valor máximo de *speedup*, ou seja, ***Sp = p*** (nesse caso, ***Tp = Ts/p***). Na prática, diversos fatores

influenciam no valor do *speedup* alcançado, principalmente no sentido de torná-lo menor. Alguns fatores que diminuem o *speedup* são: sobrecarga de comunicação e de sincronismo, balanceamento de carga e algoritmos ineficientes para explorar o paralelismo do problema a ser solucionado.

Em alguns casos o *speedup* obtido pode ser maior que o previsto, ou seja, $Sp > p$. Esse fenômeno, denominado *anomalia de speedup* ou *speedup super linear*, pode ser causado pelas características do algoritmo paralelo ou pelas características da plataforma de *hardware* (utilização da memória *cache* dos elementos de processamento) [Foster 1995, Quinn 1987].

- **Comunicação entre Processos**

Para que a cooperação entre os processos se torne possível, os processos que estão sendo executados concorrentemente devem se comunicar. A comunicação permite que a execução de um processo influencie na execução de outros processos. A comunicação entre processos é baseada no uso de *variáveis compartilhadas* (variáveis que podem ser referenciadas por mais de um processador) ou por *troca de mensagem* [Andrews; Schneider 1983].

A sincronização é freqüentemente necessária quando existe uma comunicação entre os processos. Os processos são executados com diferentes velocidades. Assim, para se comunicar, um processo deve realizar alguma ação para que o outro processo a detecte, algo como

atribuir um valor a uma variável ou enviar uma mensagem. A comunicação será possível apenas se esses eventos, "realizar uma ação" e "detectar uma ação", forem projetados para acontecerem nesta ordem.

- **Eficiência (*E*)**

A eficiência representa a porcentagem dos elementos de processamento que estão sendo efetivamente utilizados [Ayani; Berkman 1993, Hwang; Xu 1998] (Eq. (2.2)).

$$E = speedup/p * 100 \qquad (2.2)$$

Onde:

p = número de elementos de processamento.

2.3 Arquiteturas Paralelas

Tomando como base as características de uma arquitetura paralela é possível classificá-la, utilizando-se diferentes enfoques. A classificação de Flynn, proposta na década de 60, é a mais conhecida, embora apresente alguma dificuldade em abranger todas as arquiteturas atuais. A classificação de Duncan é mais abrangente, porém menos conhecida. Essas classificações são descritas a seguir.

2.3.1 Classificação de Flynn

A classificação baseada nos fluxos de instruções e de dados, proposta por Flynn [Duncan 1990, Hwang; Briggs 1984, Navaux 1989], é uma das mais intensamente utilizadas [Almasi; Gottlieb 1994, Quinn 1987]. Esta classificação considera o processo computacional como a execução de uma seqüência de instruções sobre um conjunto de dados. Aqui, o modelo de "von Neumann" , que corresponde a execução seqüencial de instruções, é visto como um fluxo único de instruções controlando um fluxo único de dados (*Single stream of Instructions/Single stream of Data* - SISD). A introdução de fluxos múltiplos de dados ou de fluxos múltiplos de instruções é que faz surgir o paralelismo. A combinação de fluxos de instrução únicos e múltiplos com fluxos de dados únicos e múltiplos produz as quatro categorias da classificação de Flynn [Duncan 1990, Hwang; Briggs 1984, Navaux 1989]:

- **SISD** - ***Single Instruction stream/Single Data stream***

  (Fluxo único de instruções, fluxo único de dados)

Esta organização representa a maioria dos computadores seqüenciais disponíveis atualmente (Princípio de Vonn Neumann). As instruções são executadas seqüencialmente mas pode existir uma defasagem de estados na execução (*pipeline*). Alguns sistemas monoprocessadores SISD são

estruturados em *pipeline*. A figura 2.2 apresenta um esquema deste tipo de arquitetura.

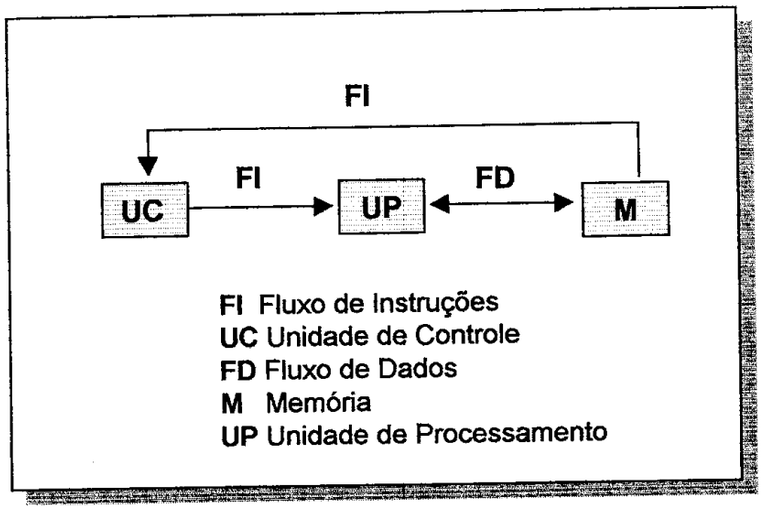


Figura 2.2 Arquitetura SISD. Único fluxo de instruções e único fluxo de dados.

- **SIMD** - ***Single Instruction stream/Multiple Data stream***

  (Fluxo único de instruções, fluxo múltiplo de dados)

Nesta classe existem vários elementos de processamento supervisionados por uma unidade de controle. Os processadores recebem as mesmas instruções de uma unidade de controle mas operam em diferentes conjuntos de dados. Os processadores vetoriais estão normalmente incluídos nesta classificação. A figura 2.3 apresenta um esquema deste tipo de arquitetura.

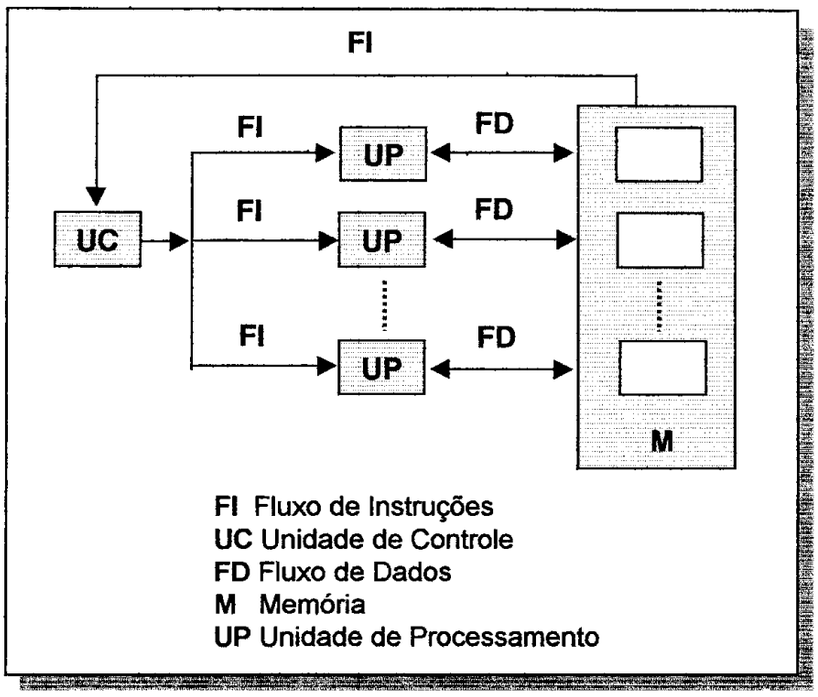


Figura 2.3 Arquitetura SIMD. Diagrama de uma arquitetura com único fluxo de instruções e múltiplos fluxos de dados.

- **MISD** - ***Multiple Instruction stream/Single Data stream***

  (Fluxo múltiplo de instruções, fluxo único de dados)

Existem *n* unidades de processamento, cada uma recebendo diferentes instruções sobre um mesmo conjunto de dados e seus derivados. Alguns autores consideram esta classificação pouco significativa, chegando mesmo a não considerá-la [Almasi; Gottlieb 1989, Quinn 1987]. Em alguns casos, os "macropipelines" são encaixados nesta categoria uma vez que as

saídas de uma unidade de processamento servem de entrada para outra unidade de processamento (figura 2.4).

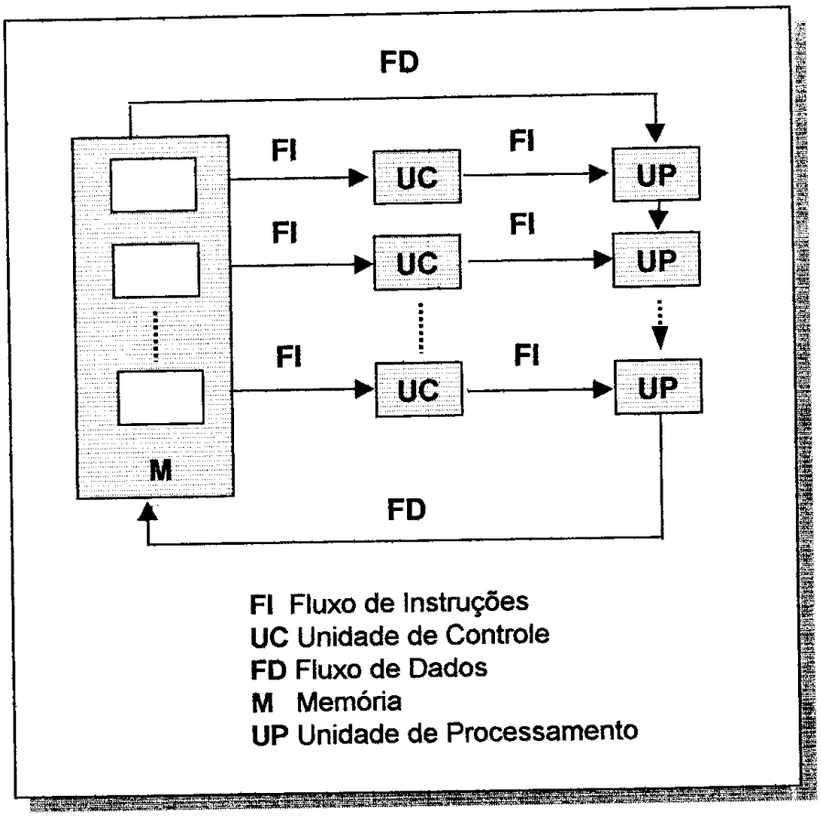


Figura 2.4 Arquitetura MISD. Diagrama de uma arquitetura com múltiplos fluxos de instruções e único fluxo de dados.

- **MIMD - *Multiple Instruction stream/Multiple Data stream***

  (Fluxo múltiplo de instruções, fluxo múltiplo de dados)

A maioria dos sistemas multiprocessadores estão incluídos nesta categoria. Nesta classe cada processador é controlado por uma unidade de

controle, executando instruções independentemente sobre diferentes fluxos de dados. A figura 2.5 mostra um esquema deste tipo de arquitetura. Esta arquitetura apresenta uma grande flexibilidade para desenvolvimento de algoritmos paralelos.

A classificação de Flynn foi considerada insuficiente por não abranger algumas arquiteturas modernas, como as máquinas de fluxo de dados e os processadores vetoriais, além de não apresentar uma hierarquia (a classificação *MIMD* engloba quase todas as arquiteturas paralelas sem apresentar subníveis) [Santana et al. 1997]. Duncan [Duncan 1990] propôs uma nova classificação, mais complexa e abrangente, descrita na seção 2.3.2.

2.3.2 Classificação de Duncan

A classificação de Duncan está dividida em dois grupos principais, arquiteturas síncronas e arquiteturas assíncronas [Duncan 1990]. Essa classificação é apresentada na figura 2.6 e detalhada nas seções 2.3.2.1 e 2.3.2.2.

2.3.2.1 Arquiteturas Síncronas

Denominam-se arquiteturas síncronas aquelas arquiteturas paralelas que coordenam as operações por meio de relógios globais, unidade de controle central ou controladores de unidades vetoriais.

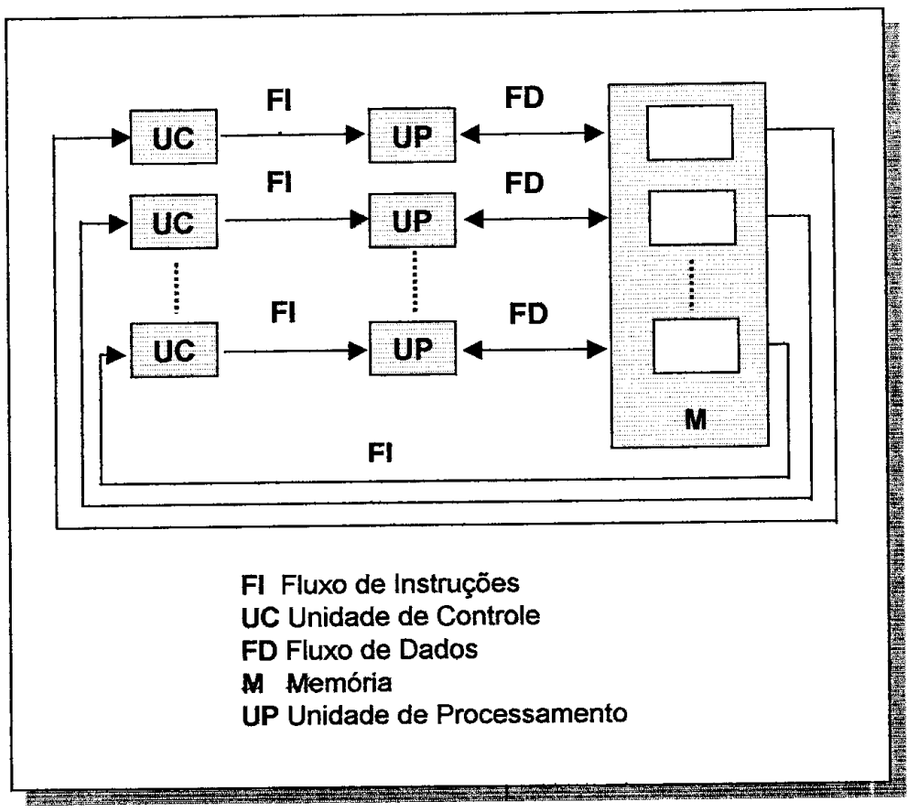


Figura 2.5 Arquitetura MIMD. Representação do fluxo de instruções e dados de uma máquina com arquitetura MIMD.

Essas arquiteturas são divididas em três classes:

- **Processadores Vetoriais**: são caracterizados por unidades funcionais que implementam operações sobre vetores;
- **Arquiteturas *SIMD***: apresentam uma única unidade de controle e múltiplos elementos de processamento. São divididas em duas categorias:

- **Processadores Matriciais**: desenvolvidos para aplicações específicas e possibilitam operações sobre matrizes de dados;

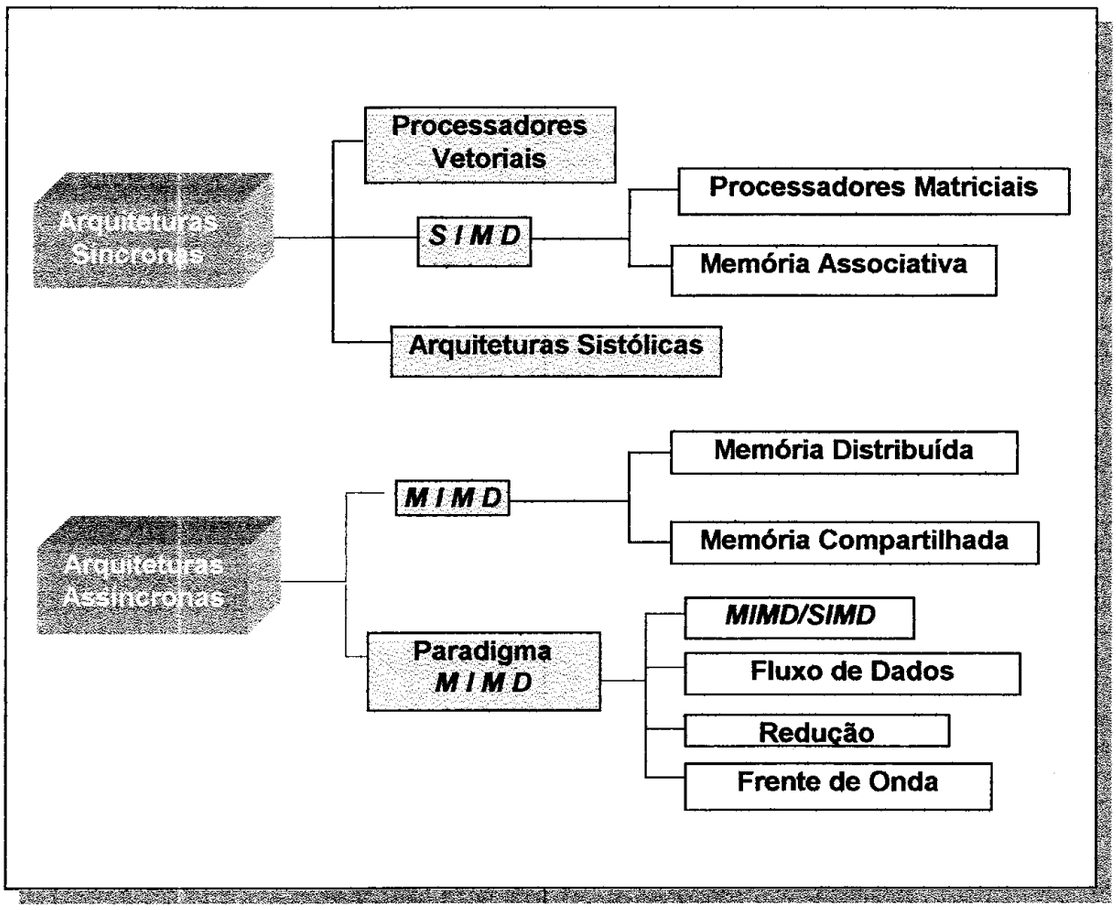


Figura 2.6 Classificação de Duncan para Arquiteturas Paralelas. A classificação das arquiteturas por Duncan considera duas classes principais, as arquiteturas Síncronas e as Assíncronas.

- **Memória Associativa**: arquiteturas que utilizam comparações lógicas para recuperar, em paralelo,

dados armazenados na memória, de acordo com seu conteúdo;

- **Arquiteturas Sistólicas**: foram propostas para resolver problemas que exigem computação intensa, com grande quantidade de operações de E/S (Entrada/Saída). São constituídas por vários elementos de processamento organizados de forma a constituir um *pipeline*, que formam uma cadeia, na qual apenas os elementos de processamento localizados nos limites dessa estrutura possuem comunicação com a memória [Santana et al. 1997].

2.3.2.2 Arquiteturas Assíncronas

As arquiteturas assíncronas são caracterizadas por um controle de *hardware* descentralizado, o que possibilita aos múltiplos elementos de processamento serem independentes. Esses elementos de processamento trabalham com conjunto de dados distintos. As arquiteturas assíncronas são classificadas em:

- **Arquiteturas *MIMD*:** são constituídas por múltiplos elementos de processamento independentes, que executam diferentes conjuntos de instruções, sobre conjuntos de dados distintos. Esse tipo de arquitetura oferece grande flexibilidade para a construção

de algoritmos paralelos [Blech 1994]. As arquiteturas *MIMD* podem ser divididas em duas categorias, dependendo do grau de interação entre os elementos de processamento:

- **Memória Compartilhada (multiprocessadores).** Caracterizam-se pela existência de uma memória global e única, a qual é utilizada por todos os elementos de processamento. A comunicação e a sincronização entre os diversos elementos é realizada por intermédio do compartilhamento de posições dessa memória [Santana et al. 1997, Tanenbaum 1995];

- **Memória Distribuída (multicomputadores).** Cada elemento de processamento possui sua própria memória local, e toda a comunicação e a sincronização são realizadas por meio da troca de mensagens [Tanenbaum 1995];

- **Paradigma *MIMD*:** engloba as arquiteturas assíncronas que, apesar de apresentarem as características de uma arquitetura *MIMD*, possuem também características próprias, que dificultam a sua classificação como puramente *MIMD*:

- ***MIMD/SIMD* (híbridas)**. São caracterizadas pelo fato de algumas partes da arquitetura *MIMD* estarem sob o controle *SIMD*;
- **Fluxo de Dados**. Nessas arquiteturas uma instrução é executada assim que os dados necessários (operandos) se tornam disponíveis. Desta forma, a seqüência de instruções é baseada apenas na dependência entre os dados;
- **Redução (dirigidas à demanda).** Nesse tipo de arquitetura uma instrução é habilitada para execução quando seu resultado é solicitado como operando de outra instrução, também habilitada para execução;
- **Frente de Onda.** Combinam uma estrutura sistólica de elementos de processamento com o paradigma de execução de fluxo de dados.

Apesar do desenvolvimento de máquinas complexas e com grande poder computacional, a utilização do processamento paralelo foi restrita durante muitos anos. Esse fato é justificado pelo lento desenvolvimento de ferramentas de *software*, dificultando e afastando o usuário comum desse tipo de processamento. O desenvolvimento de ferramentas como compiladores paralelizadores e bibliotecas possibilitou a mudança desse

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

cenário. A seção 2.4 apresenta os paradigmas para o desenvolvimento de algoritmos paralelos em arquiteturas MIMD.

2.4 Ferramentas para Construção de Algoritmos Paralelos

O modelo computacional MIMD tem obtido destaque no desenvolvimento de aplicações paralelas devido a sua flexibilidade, além da sua potencialidade na execução de programas paralelos de granulosidade média e grossa. Desta forma, optou-se neste trabalho pela restrição às arquiteturas MIMD.

A paralelização em modelos MIMD pode ser dividida em três categorias: *algoritmos pipeline, algoritmos particionados, e algoritmos relaxados* [Quinn 1987].

- **Algoritmos *Pipeline***

Um algoritmo *pipeline* é um conjunto ordenado de segmentos no qual a saída (*output*) de cada segmento serve como entrada (*input*) do seu sucessor. A entrada de dados do algoritmo serve como *input* do primeiro segmento; e o *output* do último segmento é a saída de dados do algoritmo. Todos os segmentos devem produzir resultados em uma mesma taxa, ou então o segmento mais lento se constituirá em um "gargalo". Alguns autores usam o termo "macropipeline" para se referir a este tipo de algoritmo. O

speedup resultante obtido é limitado ao mínimo *speedup* obtido pelos segmentos.

Um *algoritmo sistólico* é um tipo especial de algoritmo *pipeline.* Três características o distingue: o fluxo de dados é regular, o fluxo de dados não é unidirecional, e por último, as computações realizadas pelos segmentos são idênticas.

- **Algoritmos particionados**

Diferentemente do *pipeline*, no qual os processadores executam tarefas computacionais distintas, o particionamento é um compartilhamento da computação. Um determinado problema é dividido em subproblemas que são resolvidos por processadores individuais. As soluções dos subproblemas são combinadas para formar a solução do problema. Essa combinação das soluções implica na necessidade de sincronização dos processadores; por esta razão os algoritmos de particionamento também são chamados de *algoritmos sincronizados.*

- **Algoritmos relaxados**

Um algoritmo que trabalha sem sincronização dos processadores é denominado *relaxado.* Os processos podem trabalhar todos com a mesma meta (como nos algoritmos particionados) ou com cada um realizando uma tarefa específica (como em algoritmos *pipeline*), mas a principal característica é que nenhum processador precisa esperar por outro para ser

abastecido com dados. Alguns autores preferem chamar estes algoritmos de *algoritmos assíncronos.*

Os algoritmos para modelos MIMD se incluem nestes três tipos vistos anteriormente, na realidade essa distinção nem sempre se torna tão clara, e um algoritmo paralelo pode possuir características combinadas dos três tipos.

Independente de qual o tipo de algoritmo paralelo deseja-se construir, [Almasi; Gottlieb 1994] definem os seguintes tipos de ferramentas para o desenvolvimento de programas paralelos: compiladores paralelizadores, extensões paralelas para linguagens seriais (bibliotecas), compiladores especiais e linguagens concorrentes.

- **Compiladores paralelizadores**: geram automaticamente versões paralelas de programas não paralelos. Esses compiladores exigem o mínimo de trabalho do usuário, mas o desempenho obtido normalmente é modesto. Além disso, nem sempre esses compiladores estão disponíveis, principalmente para sistemas de memória distribuída. [Blech 1994, Hwang; Xu 1998]. Como exemplo, pode-se citar *Parafrase* [Almasi; Gottlieb 1994];

- **Extensões paralelas**: são bibliotecas cujo conjunto de instruções, aliado com as primitivas da linguagem hospedeira, permite o desenvolvimento de aplicações paralelas. Possuem como vantagem o fato de não exigirem do usuário o aprendizado de uma nova linguagem. Normalmente apresentam um desempenho melhor quando comparadas com os compiladores paralelizadores. Como exemplo tem-se *PVM* (*Parallel Virtual Machine*) e *MPI*

(*Message Passing Interface*, para arquiteturas de memória distribuída);

- **Compiladores especiais**: permitem a extensão de linguagens seqüenciais com ferramentas para o desenvolvimento de aplicações paralelas, como processos e monitores. Como exemplo pode-se citar Pascal Concorrente (para arquiteturas de memória compartilhada);

- **Linguagens concorrentes**: são linguagens criadas especificamente para esse fim. Apresentam a desvantagem de exigir do usuário o aprendizado de uma nova linguagem de programação. Por outro lado, podem oferecer melhor desempenho para as aplicações. Como exemplos, podem ser citadas as linguagens *Occam* e *Ada*.

A escolha de qual ferramenta utilizar deve ser feita considerando diferentes aspectos, como o tipo de aplicação, o perfil do usuário que utilizará a ferramenta e a arquitetura que executará os códigos gerados. Dois aspectos são fundamentais na escolha: o tempo de trabalho do usuário e o desempenho obtido [Almasi; Gottlieb 1994].

Definida a ferramenta a ser usada, deve-se determinar o paradigma a ser utilizado para o desenvolvimento da aplicação paralela, ou seja, especificar o modelo de programação que descreve o método por meio do qual os elementos de processamento são coordenados na execução de suas tarefas:

- **Troca de mensagens**: os diversos processos comunicam-se e sincronizam-se enviando e recebendo mensagens;
- **Memória compartilhada**: múltiplos elementos de processamento compartilham uma memória única.

O paradigma de passagem de mensagens para o desenvolvimento de aplicações paralelas pode ser subdividido em duas classes: *SPMD* e *MPMD* [Geist et al. 1994]:

- ***SPMD*** (*Single Program, Multiple Data*): nesse modelo um único programa executa em mais de um elemento de processamento, ou seja, todos os elementos de processamento executam o mesmo código, porém sobre um conjunto de dados distinto;

- ***MPMD*** (*Multiple Program, Multiple Data*): neste caso programas diferentes executam em elementos de processamentos distintos (cada elemento de processamento com seu próprio conjunto de dados e código).

O paradigma de troca de mensagens oferece uma forma simples e principalmente adaptável em qualquer plataforma para o desenvolvimento de aplicações paralelas. Desta forma, ressalta-se a importância da programação por meio da troca de mensagens e dos ambientes de passagem de mensagens, que se destinam a fornecer maneiras de se utilizar sistemas distribuídos como ambientes de programação paralela, e assim fornecer plataformas flexíveis e de menor custo para a construção e execução de algoritmos paralelos [Souza 1996]. A programação via troca

de mensagens e os ambientes de passagem de mensagens são discutidos na seção 2.5.

2.5 Programação Paralela Utilizando Ambientes de Passagem de Mensagens

Um ambiente de passagem de mensagens consiste basicamente em uma biblioteca (extensão funcional) de comunicação que, juntamente com os comandos oferecidos por uma linguagem seqüencial hospedeira, como C ou Fortran, permite o desenvolvimento de aplicações paralelas. Quealy, em [Quealy 1994], define passagem de mensagens como "um método pelo qual os processos se comunicam quando não compartilham memória". O paradigma de passagem de mensagens pode ser visualizado na figura 2.7.

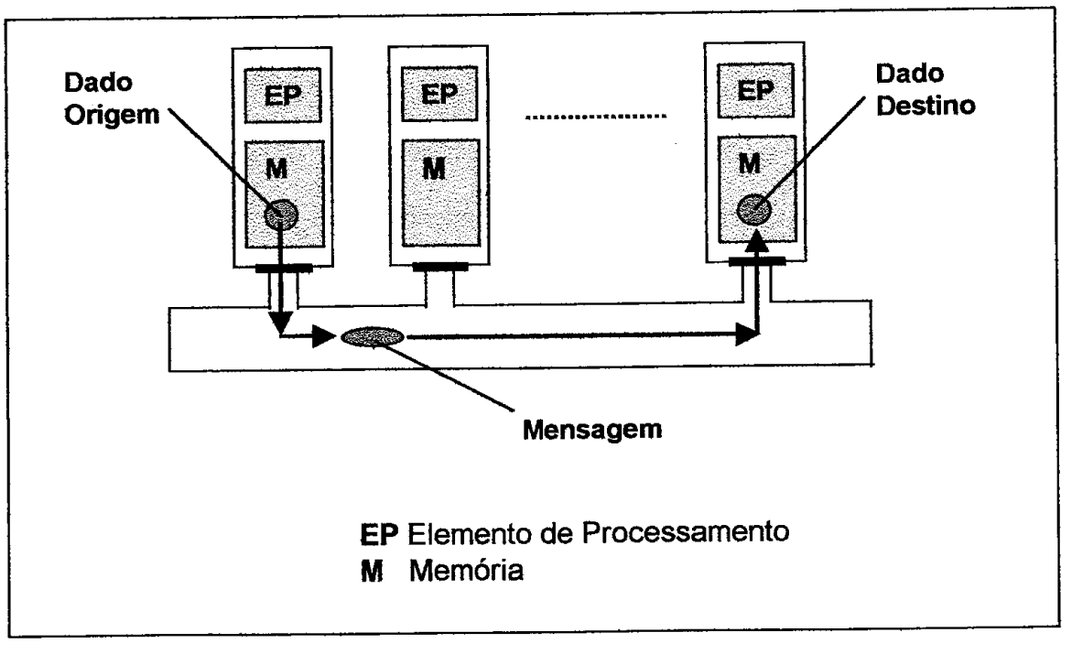


Figura 2.7 Paradigma de Passagem de Mensagem. Um método pelo qual os processos se comunicam quando não compartilham memória.

Inicialmente, os ambientes para passagem de mensagens foram criados pelos próprios fabricantes para os sistemas *MPP's* (*Massively Parallel Processor*). Porém, com a ausência de padrões, aplicações escritas para uma determinada arquitetura não eram portáteis para outras arquiteturas. Os ambientes desenvolvidos, apesar de serem logicamente similares, diferiam na maneira como os serviços principais (sincronização e passagem de dados) eram organizados. Como resultado, obteve-se um grande número de sistemas incompatíveis, dentre os quais podem ser citados *Meiko CS*, *IBM EUI* e *nCUBE PSE* [Geist et al. 1994b].

A solução encontrada para o problema da incompatibilidade foi o desenvolvimento das chamadas plataformas de portabilidade [Mcbryan 1994]. Neste caso, um ambiente de programação é definido e implementado em várias plataformas paralelas (arquiteturas paralelas ou sistemas distribuídos), possibilitando que programas possam ser migrados de maneira fácil e direta. Essas plataformas de portabilidade também participaram ativamente do processo de migração da programação paralela para sistemas distribuídos em virtude da adequação natural desses tipos de plataformas para ambientes heterogêneos, formados pela interligação de diferentes arquiteturas em um ambiente único [Souza 1996].

Muitos ambientes de passagem de mensagens foram desenvolvidos nos últimos anos, tendo como característica básica a portabilidade: os ambientes são constituídos por um conjunto de primitivas que independe da plataforma de *hardware* a ser utilizada.

Porém, a falta de um padrão para o desenvolvimento dessas plataformas deu origem a problemas semelhantes aos ocorridos com os ambientes específicos para determinadas arquiteturas. Com o objetivo de padronizar as plataformas de portabilidade, iniciou-se um processo que incluiu representantes de diversas organizações, resultando no padrão *MPI* [Mcbryan 1994].

Alguns ambientes de passagem de mensagens são apresentados na tabela 2.1 [Hwang; Xu 1998].

Tabela 2.1 Ambientes de Passagem de Mensagens.

Ambiente	Origem	Observação
CMMD	Thinking Machines	Software proprietário
Express	ParaSoft Corporation	Software proprietário
Fortran-M	Argonne National Laboratory	Software domínio público
MPI	MPI Forum	Software domínio público
Nx	Intel	Software proprietário
P4	Argonne National Laboratory	Software domínio público
PARMACS	ANL/GMD	Software domínio público
PVM	Oak Ridge National Laboratory	Software domínio público
PVMe	IBM	Software proprietário

O *CMMD* é utilizado em sistemas *Thinking Machines CM-5 Express*, da *ParaSoft Corporation*, e é constituído por uma coleção de ferramentas e utilitários, incluindo uma interface para passagem de mensagens, para programação paralela em arquiteturas *MIMD* com memória distribuída.

O *Fortran-M* é uma extensão do *Fortran 77*, e foi projetada para fornecer suporte aos paradigmas de passagem de mensagens e de memória

compartilhada. Apenas o suporte para passagem de mensagens está disponível atualmente.

O *MPI* foi desenvolvido com objetivo de padronizar os ambientes para passagem de mensagens em sistemas *MIMD* com memória distribuída.

Nx é um sistema desenvolvido para arquiteturas *Intel MPP's* (*Paragon*). *P4* e *Parmacs* (que herdou muitas das características do *P4*) são bibliotecas de macros e subrotinas desenvolvidas pelo *Argonne National Laboratory*, para a programação em uma variedade de máquinas paralelas. *P4* e *Parmacs* suportam os modelos de programação baseados em memória compartilhada (por meio de monitores) e distribuída (pela troca de mensagens).

O *PVM* é um ambiente de passagem de mensagens desenvolvido pelo *Oak Ridge National Laboratory*, sendo considerado um padrão entre os ambientes de passagem de mensagens. *PVMe* é um ambiente de passagem de mensagens desenvolvido pela *IBM*.

2.5.1 Computação Paralela em Sistemas Distribuídos

Em arquiteturas paralelas, normalmente todos os elementos de processamento são iguais em termos de capacidade, recursos, *software* e velocidade de comunicação. Porém, em um sistema distribuído, podem ser encontradas dificuldades no desenvolvimento de programas paralelos em função da heterogeneidade [Geist et al. 1994a]:

- **Arquitetura**: o conjunto de computadores disponíveis pode incluir uma grande variedade de tipos de arquiteturas, como por exemplo computadores pessoais, estações de trabalho de alto desempenho, multiprocessadores, computadores vetoriais e arquiteturas maciçamente paralelas;

- **Formato de dados**: computadores diferentes podem apresentar formatos de dados distintos, ou seja, incompatíveis, o que exige algum tipo de padronização;

- **Potência computacional**: em um ambiente virtual composto por diferentes tipos de arquiteturas deve-se garantir que nenhuma das máquinas fique ociosa, o que pode levar a um baixo desempenho;

- **Carga de trabalho na rede e em cada máquina**: com muitos usuários a carga de trabalho imposta à rede e até mesmo às máquinas pode degradar o desempenho das aplicações paralelas.

Apesar dessas dificuldades, a computação paralela em sistemas distribuídos oferece diversas vantagens:

- Não existe a necessidade de grandes investimentos adicionais, uma vez que o *hardware* já existente pode ser utilizado;
- Alto desempenho, pois cada tarefa pode ser atribuída para a arquitetura mais apropriada;

- Os recursos da máquina virtual podem aumentar gradativamente, o que permite maior facilidade na assimilação de novas tecnologias.

Além das vantagens citadas, um requisito fundamental para a computação distribuída são as redes de comunicação de alta velocidade, que têm apresentado um grande avanço com o desenvolvimento de novas tecnologias visando maior *throughput* e baixa latência. Podem ser citadas, entre outras, as tecnologias *Ethernet*, *FDDI* (*Fiber Distributed Data Interface*) e *ATM* (*Asynchronous Transfer Mode*). Todas essas características têm incentivado a utilização dos chamados ambientes de passagens de mensagens.

Diversos grupos de pesquisa têm desenvolvido ambientes para passagem de mensagens, visando otimizar a utilização de recursos, reduzir os custos e o tempo de desenvolvimento e possibilitar implementações mais eficientes das aplicações, utilizando a computação distribuída. Uma vantagem adicional dos ambientes de passagem de mensagens é permitir a migração de programas paralelos entre diferentes tipos de plataformas.

2.6 Conclusões

Os últimos anos têm apresentado um avanço tecnológico significativo, o qual tem possibilitado tanto o desenvolvimento de arquiteturas com maior poder computacional quanto o desenvolvimento de novas tecnologias de redes de comunicação, resultando no surgimento de máquinas para

processamento de alto desempenho. Como conseqüência um grande aumento na aceitação e utilização da computação paralela, voltado para a computação científica de alto desempenho e também para as aplicações de propósito geral, vem sendo observado.

Para suportar certas aplicações, existe uma necessidade por maior velocidade para superar as restrições de tempo, devido à complexidade computacional dos algoritmos utilizados e/ou ao tamanho do conjunto de dados a serem processados. Nesse sentido, o aumento da velocidade tem ocorrido às custas de pesquisas e desenvolvimentos em áreas como microeletrônica, arquitetura, software, e periféricos. Apesar dos avanços específicos em cada área, o paralelismo, de maneira geral, é que vem aumentando substancialmente a velocidade dos sistemas computacionais.

O aumento de velocidade decorrente do paralelismo é uma questão de extrema importância em computadores paralelos. Esse aumento de velocidade observado entre um computador paralelo com *p* processadores e um computador seqüencial é denominado *speedup*.

Para que exista uma cooperação, deve haver a possibilidade de sincronismo e comunicação entre os processos que estão sendo executados concorrentemente. A comunicação permite que a execução de um processo influencie na execução de outros processos. A comunicação entre os processos é baseada no uso de variáveis compartilhadas ou por passagem de mensagem [Andrews; Scheneider 1983].

Capítulo 3

Simulação Distribuída

Este capítulo aborda os conceitos que envolvem a realização de uma simulação distribuída. Do ponto de vista do projetista são discutidos os aspectos do esquema básico de simulação distribuída e da modelagem do sistema físico a ser simulado. Os principais protocolos de sincronização para simulação distribuída também são discutidos.

3.1 Introdução

A limitação das técnicas matemáticas, particularmente da teoria de filas, encontra-se no grande número de restrições que devem ser impostas aos sistemas para que os resultados sejam confiáveis. Tais restrições

acabam resultando em análises pouco realistas [Grillo 1989, Chandy; Misra 1981].

A princípio, protótipos eram implementados, correspondendo às vezes, à construção de elementos de hardware complicados e caros, acoplados a um grande esforço de desenvolvimento de software. Atualmente, os protótipos têm sido simulados.

Uma simulação realista, todavia, requer grande esforço computacional e o tempo de processamento necessário, muitas vezes, inviabiliza a simulação. Este cenário tem mudado com o uso de supercomputadores e de ambientes formados por processadores com grande capacidade de comunicação, onde o paralelismo pode ser explorado. As técnicas de simulação também têm evoluído. A maneira tradicional e seqüencial, que era inerentemente lenta, gradativamente tem dado lugar às técnicas de simulação distribuída que exploram convenientemente o paralelismo .

A idéia de simulação distribuída foi proposta por Chandy em 1977 em uma série de conferências na Universidade de Waterloo e, independentemente, por R. E. Bryant [Misra 1986, Nicol; Fujimoto 1995].

A possibilidade da diminuição do tempo de execução da simulação (*speedup*), a grande quantidade de memória disponível em uma arquitetura paralela viabilizando a simulação de modelos mais robustos e o pouco controle global exercido por qualquer máquina são as características que tem incentivado o uso da simulação distribuída [Low et al. 1999].

Além disso, a simulação de um sistema pode ser adaptada à estrutura do hardware disponível; por exemplo, se poucas máquinas estão

disponíveis para a simulação, diversos processos físicos podem ser simulados (seqüencialmente) em um único processador.

3.2 Limitações da Simulação Seqüencial

Usando a simulação seqüencial, modelos de simulação de sistemas complexos ou a simulação de baixo nível podem requerer horas ou mesmo dias de processamento computacional. Este problema é crítico quando estão sendo simulados sistemas computacionais. Um exemplo desse problema ocorre ao serem simuladas as instruções de máquina de um sistema computacional, que passam a gastar centenas de vezes o tempo real de execução. Os computadores mais complexos e os sistemas de comunicação já se apresentam intratáveis matematicamente e por essa razão recorre-se à simulação para a avaliação de seus desempenhos. As técnicas de simulação seqüencial podem ser inadequadas para esses sistemas porque apenas um pequeno número de acontecimentos é simulado por vez [Chandy ;Misra 1981, Misra 1986, Grillo 1989].

Outro exemplo crítico é a simulação de circuitos digitais complexos visando a verificação de sua lógica. Alguns circuitos necessitam de semanas de tempo de processamento para serem simulados. Apesar desta infinidade de tempo, os projetistas não têm outra escolha, uma vez que um projeto é inaceitável caso não tenha sido testado.

Tratando-se de sistemas distribuídos, além da necessidade da análise de diversos parâmetros tornar trágico o tempo de processamento, ainda há

o desconforto de modelar um sistema real distribuído numa simulação seqüencial e centralizada.

O grande problema na conversão de uma simulação seqüencial para uma simulação distribuída está na natureza seqüencial da execução dos eventos de uma simulação. A abordagem clássica para simulação seqüencial aborda a manipulação de uma estrutura de dados chamada *lista de acontecimentos futuros*, de acordo com os valores de uma variável relógio (*clock*) que representa a passagem do tempo na simulação [Fujimoto 2000, Misra 1986, MacDougall 1987].

A natureza do mecanismo da lista de acontecimentos futuros imposta pela simulação seqüencial, onde a cada ciclo da simulação apenas um item é removido da lista de acontecimentos, não torna possível a execução concorrente do algoritmo em máquinas com mais de um elemento processador, uma vez que a lista de acontecimentos não pode ser efetivamente particionada. A existência de uma lista de acontecimentos posicionados cronologicamente impõe uma simulação seqüencial onde apenas um evento é simulado por vez [Garcia 1991].

Esquemas básicos onde mesmo não existindo uma estrutura seqüencial como a lista de acontecimentos futuros mas que ainda assim promovem o avanço coerente do tempo da simulação têm sido propostos e discutidos intensamente. Os principais protocolos de sincronização para simulação distribuída são discutidos nas próximas seções.

3.3 Esquema Básico de Sincronização para Simulação Distribuída Via Troca de Mensagens

Uma simulação distribuída típica, independente de qual mecanismo é utilizado para sincronizá-la, consiste em uma coleção de processos lógicos (*pl*), onde cada *pl* irá simular algum processo físico (*pf*) do sistema real. Muitos trabalhos na literatura utilizam o conceito de *pl* para representar uma unidade seqüencial de processamento em um modelo. Entretanto a decomposição de um modelo em vários *pl's* pode ser dirigida por questões de modularidade, características do software utilizado ou pelo desempenho obtido através do paralelismo.

Em uma simulação distribuída cada *pl* simula um *pf* e é executado em apenas um processador. Este processador pode ser representado por uma máquina em uma rede de computadores ou apenas um nó de uma arquitetura paralela com memória distribuída. Um determinado mecanismo pode ser utilizado para permitir a troca de mensagens entre os *pl's*. Este mecanismo pode ser uma forma explícita de troca de mensagens,uma *chamada de procedimento remoto* (RPC) ou outro mecanismo que irá depender do software utilizado. Na simulação, se pf_i envia mensagens para pf_j, então deverá existir um canal de comunicação entre pl_i e pl_j. Cada processo lógico pode executar códigos seqüencialmente e dois comandos: *recebe* e *envia*. No comando *envia*, um *pl* utiliza um canal de saída e constrói uma mensagem que será enviada através do canal. A

execução do *envia* resulta em um armazenamento da mensagem no canal utilizado; o processo emissor então prossegue com a execução do código (comunicação não bloqueante). Cada mensagem leva um tempo arbitrário para atingir seu destino. As mensagens enviadas pelo canal são recebidas na seqüência pela qual foram enviadas. Em um comando *recebe*, um *pl* escolhe um ou mais canais de chegada para receber suas mensagens. Um *pl* que deseja receber uma mensagem deve esperar que esta atravesse o canal de chegada. Note que este protocolo de comunicação é bastante simples e pode ser implementado em diversas máquinas com diferentes arquiteturas.

Antes de descrever os protocolos de sincronização, implementados a partir do esquema básico descrito anteriormente, deve-se definir alguns conceitos importantes que auxiliam no entendimento de uma simulação distribuída.

3.3.1 *Lookahead*

Desde de que o conceito de *lookahead* foi inicialmente introduzido por Chandy e Misra [Chandy; Misra 1979] ele tem sido interpretado por diferentes maneiras. Para descrever o conceito de *lookahead* será utilizado um exemplo de um sistema simples de fila, conforme representado na figura 3.1.

Neste sistema as tarefas chegam através de um canal de entrada, submetem-se a uma fila (FIFO – *First In First Out*) e eventualmente são

processadas pelo servidor. O tempo de serviço pode ser obtido através de uma distribuição exponencial e é não preemptivo.

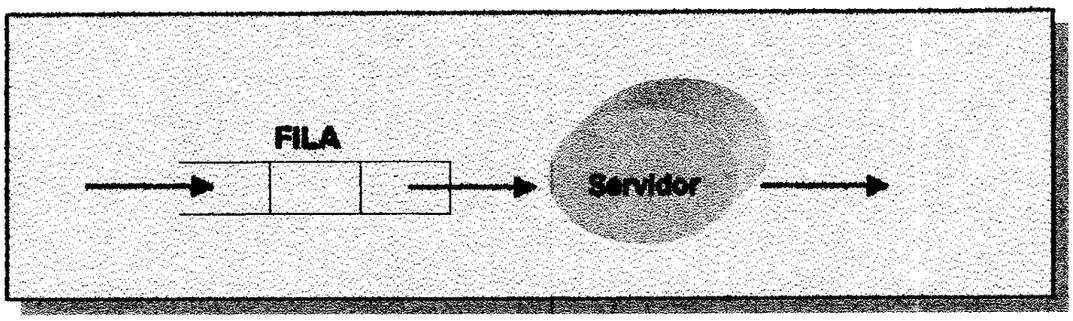


Figura 3.1 - Sistema de Fila FIFO. Política de manipulação na qual o primeiro elemento a chegar na fila é o primeiro elemento a ser executado.

Considere o seguinte cenário. No tempo 30 da simulação uma tarefa *A* inicia seu processamento (com duração de 12 unidades de tempo) e a tarefa *B*, *C* e *D* estão na fila aguardando serem processadas. Os tempos de serviço para cada uma delas é de 3, 5 e 11 respectivamente. No tempo 32 uma nova tarefa (*E*) chega ao sistema com tempo de serviço 9. A tabela 3.1 mostra os tempos de ocorrência, de duração e de término do processamento de cada uma das tarefas.

As questões associadas ao *lookahead* podem ser discutidas através do comportamento deste processo lógico simulando o sistema de fila em relação a tarefa *E*.

Tabela 3.1 - Tempos de Serviço. Tempos de ocorrência, duração e tempos de término de execução.

Tarefa	Início	Duração	Término
A	30	12	42
B	42	3	45
C	45	5	50
D	50	11	61
E	61	9	70

A figura 3.2 mostra a situação do sistema no instante em que a tarefa *E* é recebida (T = 32). Uma vez que o *pl* tenha atingido o tempo de simulação 32, o próximo evento a ser processado é o tratamento da chegada da mensagem associada a tarefa *E*.

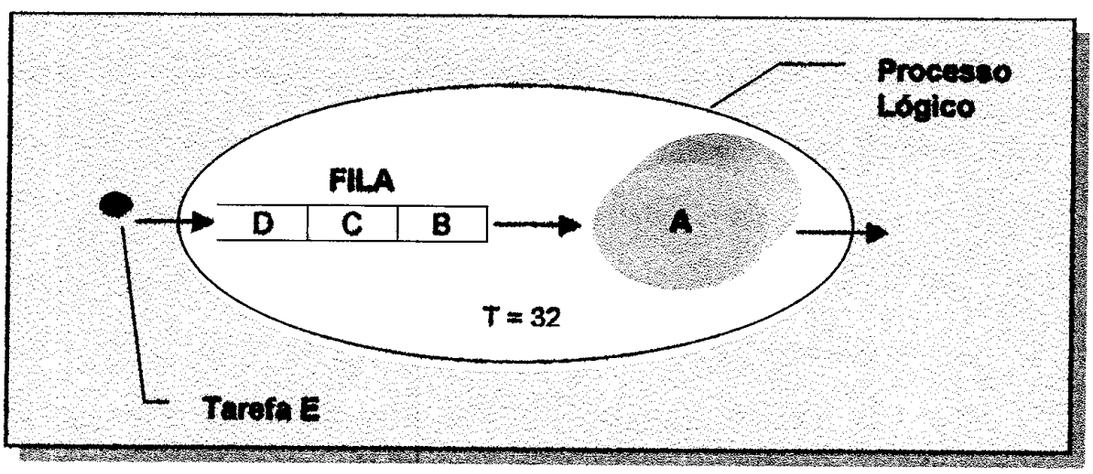


Figura 3.2 - Processo Lógico do Exemplo do Sistema de Fila Simples. Representação do esquema de chegadas de tarefas em um processo lógico.

É possível para o *pl*, uma vez que já tenha avançado para o tempo 32, calcular que o término do processamento da tarefa E terá uma marca de tempo com valor 70, uma vez que são conhecidos os valores do tempo de execução para as tarefas que chegaram anteriormente.

Sendo assim, pode-se definir o *lookahead* como a possibilidade de previsão do tempo em que um determinado evento da simulação irá ocorrer.

Esta estimativa de tempo de envio da mensagem relacionada a tarefa *E* pode ser alterada se a disciplina da fila não for FIFO ou se o servidor for preemptivo.

3.3.2 *Lookahead* e os Tempos Característicos da Passagem de um Evento pelo Processos Lógicos

Uma outra forma de definir o conceito de *lookahead* pode ser feita através da caracterização de três instantes em um *pl* em relação às suas mensagens (ou tarefas):

T_{cause} : Tempo associado ao *pl* quando a decisão de enviar uma mensagem é realizada [Fujimoto 1988]. Entre T_{cause} e o envio efetivo da mensagem Λ, este valor pode ser alterado.

T_{commit} : Próximo tempo pelo qual nenhuma mensagem futura poderá alterar o envio da mensagem em questão (Λ). Note que T_{commit} é o tempo no qual a mensagem Λ é enviada.

T_{effect} : Marca de tempo da mensagem que está sendo enviada (Λ).

A figura 3.3 mostra estes três tempos para uma mensagem típica. Inevitavelmente, o conceito de *lookahead* possui um número bastante alto de variações em seu significado. Em todas essas variações o *lookahead* pode ser considerado como a diferença entre os tempos T_{commit} e T_{effect} (figura 3.3).

A figura 3.4 ilustra, utilizando o mesmo exemplo do servidor de fila única, o conceito de *lookahead* proposto por Chandy e Misra.

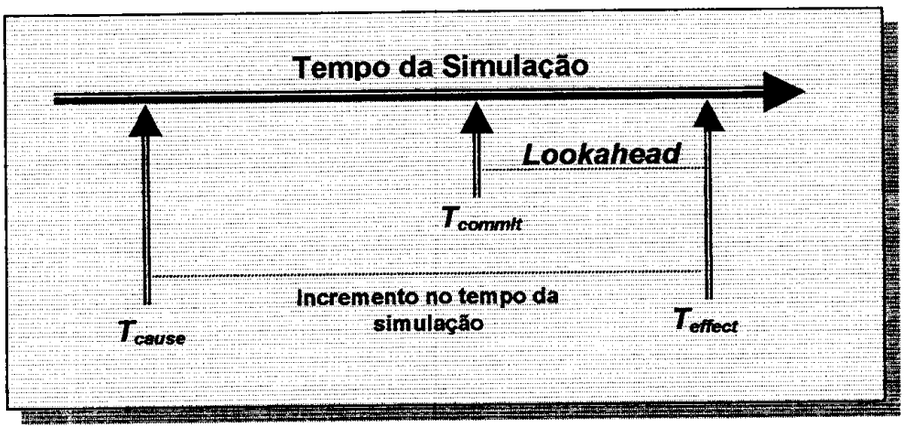


Figura 3.3 - Tempos Característicos para as Tarefas. Representação do *lookahead* através dos tempos de ocorrência da simulação.

Como foi mencionado no exemplo anterior (seção 3.3.1) no tempo de simulação para o processo lógico (LVT) 30 já se pode calcular, através dos tempos de chegada e de serviço de cada mensagem, os tempos de início e fim de processamento para as mensagens *A*, *B*, *C* e *D*. Quando a mensagem *E* chega ao sistema no tempo 32 pode-se, da mesma forma,

saber quando ela será processada (LVT = 61) e quando será concluída (LVT = 70).

Note que valores de T_{cause} (quando a mensagem foi recebida) e T_{commit} (quando foi enviada para a fila) são os mesmos e valem 32.

Com isto tem-se que o valor para o *lookahead*, segundo [Misra 1986] é igual a *70 -32 = 38.*

Na proposta original, Chandy e Misra descrevem o *lookahead* em termos de T_{effect}. Sua definição é:

"Neste caso, mensagens enviadas por um processo i para um processo j no intervalo ([T_{commit}] , [T_{effect}]) são independentes das mensagens recebidas pelo processo i depois de [T_{commit}]. Nós [Chandy e Misra], definimos lookahead para o arco (i,j) como sendo [T_{effect} – T_{commit}]. O valor do lookahead depende apenas de [T_{commit}] e do histórico obtido pelas mensagens no processo i em [T_{commit}] " [Misra 1986].

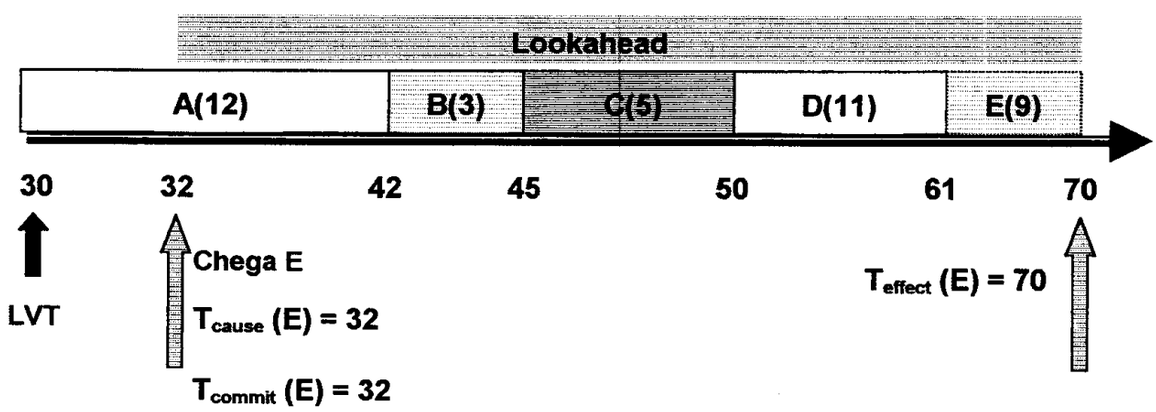


Figura 3.4 *Lookahead* para o Exemplo do Servidor de Fila Única

Em um trabalho mais recente [Fujimoto 1990] Fujimoto estendeu o conceito de *lookahead* considerando o cálculo do *lookahead* para toda mensagem enviada por um *pl*. Baseado nestas colocações, pode-se definir *lookahead* como sendo *o valor* $T_{effect} - T_{commit}$ *calculado quando* T_{commit} *for o menor tempo possível de envio da mensagem.*

Os processos lógicos em uma simulação distribuída, independente de qual protocolo de sincronização está sendo utilizado, podem apresentar diferentes valores de *lookahead*. Pela variação do *lookahead* pode-se determinar seus efeitos no desempenho da simulação distribuída. A forma com que os valores de *lookahead* afetam o desempenho da simulação distribuída é uma característica do protocolo de sincronização utilizado na simulação.

A seguir um panorama geral dos principais protocolos de sincronização e suas variantes são discutidos.

3.4 Protocolos de Sincronização para Simulação Distribuída

Pode-se observar na literatura três classes de protocolos de sincronização: conservativos [Misra 1986], otimistas [Jefferson 1985] e mistos [Jha; Bagrodia 1994], onde nesta última classe estão incluídos os modelos que podem ser executados utilizando o modo conservativo ou o modo otimista [Bagrodia 1996].

Tradicionalmente, uma simulação distribuída pode ser *Conservativa* ou *Otimista* [Spolon et al. 1999].

Um protocolo de sincronização conservativo não tolera erros de causalidade. Esse protocolo garante que, entre todas as mensagens futuras, nenhuma possuirá uma marca de tempo *menor* que o valor do clock local do processo (*LVT*). Essa condição essencial de cronologia é fiel às definições originais propostas por Chandy [Chandy ; Misra 1979]:

"Se um pl envia uma seqüência de mensagens $<...(t_i,m_i),(t_{i+1},m_{i+1})...>$ *para outro pl, então* $t_i < t_{i+1}...$ *. Isso implica que se* pl_i *recebe* (t,m) *de* pl_j*, então ele conhece todas as mensagens que* pf_i *recebe de* pf_j *até, e inclusive, o tempo t, porque qualquer mensagem futura terá um componente t maior."*

Desta forma, em uma simulação distribuída conservativa, toda vez que uma mensagem chega em um determinado *pl* sua marca de tempo é comparada com o valor do clock do processo lógico. Se esta marca de tempo for maior que o valor do clock então a mensagem é escalonada para seu próximo evento normalmente, caso contrário, se sua marca de tempo for menor que o valor do clock do processo, a simulação é interrompida.

Por outro lado em uma simulação distribuída, que utiliza um protocolo de sincronização otimista, a chegada de uma mensagem com marca de tempo *menor* que o valor do clock do processo (um evento defasado no tempo de simulação) não causa a interrupção da simulação. Nesta classe

de protocolos de sincronização as mensagens são recebidas sem uma prévia verificação da sua marca de tempo, entretanto o protocolo deve detectar e corrigir as eventuais violações de causalidade.

Os protocolos que consideram alguns subconjuntos do modelo a ser simulado utilizando-se das duas formas de sincronização vistas anteriormente são denominados de protocolos de sincronização mistos.

A seguir os principais protocolos de sincronização conservativos e otimistas e suas variantes são discutidos.

3.4.1 Protocolos de Sincronização Conservativos

Esta seção descreve os principais algoritmos de sincronização que utilizam o princípio conservativo.

3.4.1.1 Sincronização Lock-Step Time-Driven

A primeira e mais simples técnica de sincronização é a Lock-Step [Steinman 1995]. Esse protocolo força o processamento de todos os eventos em determinados períodos de tempo com um tempo fixo de incremento.

O processamento poderá ocorrer, por exemplo, nos tempos 0.0, 7.5, 15.0, 22.5, ... até o fim da simulação. Juntos cada elemento de processamento (EP) realiza seu processamento no tempo global da simulação e então espera que todos os EP's tenham realizado seus

processamentos também. Quando todos os outros EP's tiverem realizado seus processamentos no tempo corrente da simulação, eles trocam mensagens e então incrementam seus relógios em unissonância. O processamento continuará novamente no próximo intervalo de tempo.

Existem vários problemas com esse protocolo. O principal refere-se ao fato de que os eventos no mundo real não acontecem em tempos regulares. Os eventos ocorrem em tempos completamente arbitrários e não necessariamente sincronizados. Diminuir os intervalos de tempo ocasionaria um grande aumento do número de sincronizações necessárias, comprometendo o desempenho da simulação.

Mesmo quando a simulação for construída com a mais eficiente forma, ela não escapará do problema da necessidade de múltiplas sincronizações.

3.4.1.2 Sincronização Fixed Time-Bucket

Uma variante mais poderosa do protocolo Lock-Step é o protocolo Fixe Time-Bucket. Neste, os eventos são gerados com intervalos de tempo arbitrários (eventos discretos) porém com uma restrição. Os eventos nunca poderão gerar novos eventos para processos que estejam em diferentes EP's perto de tempos especificados por um valor global de *lookahead* (vamos chamá-lo de T).

Como acontecia no Lock-Step cada EP conhece o tempo global de simulação (t) mas ao invés de processa-los em um determinado valor de

tempo, os eventos são processados assincronamente na ordem de tempo correta até o tempo t+T.

Devido a esta restrição, cada EP deve ser assegurado de que nunca receberá uma mensagem de evento com uma marca de tempo menor que t+T. Isso garante que a simulação será processada logicamente correta. Quando cada EP atingir o tempo t+T, ele pára para realizar a sincronização e trocar mensagens com os outros EP's. O processamento irá então prosseguir no próximo ciclo.

O principal benefício obtido pelo protocolo Fixed Time-Bucket (em relação ao Lock-Step) é um mecanismo mais eficiente para fornecer o processamento de eventos discretos. Agora os eventos podem ocorrer nos seus valores de tempo natural ao invés de serem fixos.

3.4.1.3 Sincronização Chandy - Misra – Bryant (CMB)

O protocolo básico de sincronização CMB oferece uma alternativa ao Fixed Time Bucket no que diz respeito a não obrigatoriedade de um valor global de *lookahead* para a sincronização. Ao contrário, cada processo na simulação é forçado a conhecer quais outros processos interagem com ele próprio. Uma característica importante é que os pares de processos sempre se interagem em tempos não decrescentes. Em outras palavras, se um processo *A* interage com um processo *B*, mensagens de *A* para *B* são sempre geradas (e recebidas) em tempos com valores crescentes.

Existem muitas variantes para o protocolo CMB. Na sua forma básica, cada processo recebe suas mensagens em portas (entradas) separadas, considerando-se uma porta para cada processo. Cada processo deverá processar aquela mensagem que possui o menor tempo de ocorrência entre todas as mensagens armazenadas em suas entradas. Quando um processo possuir uma ou mais entradas vazias, ele não poderá processar seguramente nenhum de seus eventos pois não poderá prever a chegada de mensagens defasadas. Portanto, o processo ficará bloqueado até que esta situação seja resolvida.

As situações de *deadlock* podem ocorrer especialmente em simulações onde o número de troca de mensagens é bastante alto comparado com o número de portas de entrada por processo.

O protocolo CMB resolve este impasse através da utilização do conceito de mensagens nulas (mensagens com função apenas de sincronização). Estas mensagens são utilizadas para atualizar os tempos nas filas de entradas dos processos.

A exploração do conceito de *lookahead* pode melhorar o desempenho do protocolo CMB. Se um processo possuir uma entrada vazia mas, pode prever que não receberá nenhuma mensagem de um outro processo até um determinado tempo futuro, ele poderá seguramente processar alguns de seus eventos. Na prática, através do *lookahead* pode-se obter um bom desempenho com este protocolo de sincronização.

O protocolo CMB é mais adequado para simulações executando em plataformas com topologia de comunicação fixa, isto é, cada processo sabe quais processos interagem com ele mesmo.

3.4.1.3.1 Resolução de *Deadlock* Usando Mensagens Nulas

Tendo em vista o esquema básico de simulação distribuída, apresentado nas seções anteriores, é necessário redefinir *deadlock* conforme o contexto apresentado.

Um conjunto *D* de *pl*'s está em *deadlock* em um mesmo ponto da execução se todas as seguintes condições forem satisfeitas:

1. Todo *pl,* pertencente à *D*, está esperando uma mensagem ou terminou sua execução;
2. No mínimo um *pl,* pertencente à *D*, está esperando uma mensagem;
3. Um pl_i qualquer, pertencente à *D*, está esperando receber uma mensagem de algum pl_j, com pl_j também pertencendo à *D*, e não existe nenhuma mensagem em trânsito de pl_j para pl_i.

Essas condições asseguram que nenhum *pl* em *D* irá executar qualquer tipo de computação, uma vez que cada um deles está esperando a realização de algum evento pelo outro processo.

Para resolver o problema de *deadlock* pode-se utilizar um novo tipo de mensagem no simulador, denominada *mensagem nula*: $(t,nula)$, que será enviada no tempo *t* por um pl_i a um pl_j. Esta mensagem significa que pf_i não envia nenhuma mensagem a pf_j entre o valor corrente do *clock* do canal (de pl_i a pl_j) e o tempo *t*. Isto é, qualquer futura mensagem de pl_i para pl_j terá o valor do componente tempo excedendo *t*. É óbvio que essas mensagens nulas não possuem nenhuma correlação com o sistema físico. Uma mensagem nula é utilizada para indicar a ausência de mensagens. Essa ausência de mensagens no sistema físico no tempo *t* é reconhecida pela não emissão de mensagens neste instante *t*.

O próximo passo a ser seguido é modificar o esquema básico apresentado na seção anterior introduzindo o conceito de mensagens nulas.

Como visto, T_i denota o valor de *clock* de pl_i. Quando pl_i recebe uma mensagem, ele próprio atualiza T_i, e se T_i é alterado de valor, então pl_i avança a simulação de pf_i até T_i. Neste ponto pl_i prediz para cada canal de saída uma seqüência de mensagens que pf_i possa ter enviado. Assim pl_i gera $<(t_{j1},m_{j1}),(t_{j2},m_{j2}),...>$ para transmitir para o pl_j, para cada *j* no qual exista um canal de saída. Algumas dessas seqüências podem estar vazias, nesse caso nenhuma mensagem é enviada ao correspondente *pl*.

Suponha que pl_i possa futuramente predizer que após a transmissão dessa seqüência de mensagens pf_i não irá enviar mais nenhuma

mensagem a pf_j até o tempo t_j. Então, nesse novo esquema, pl_i envia $(t_j, nula)$ a pl_j depois de ter enviado a seqüência genuína de mensagens. Uma vez que pl_i conheça o estado do correspondente pf_i até o tempo T_i, ele pode predizer todas as mensagens (que estão sendo enviadas) e também detectar a ausência de mensagens, no mínimo até o tempo T_i. Consequentemente, todo canal de saída possuirá a última mensagem com o componente tempo (*t*) igual ou maior que T_i.

Note que, em qualquer iteração, apenas a última mensagem enviada através do canal pode ser uma mensagem nula.

A recepção de uma mensagem nula é tratada da mesma maneira como na recepção de qualquer outra mensagem: ela fará com que *pl* atualize seu estado interno atualizando o valor do *clock*.

Suponha que um sistema físico seja simulado até um tempo *T*. Todo processo fonte (que não recebe mensagens) deve enviar mensagens até que o componente do tempo *(t)* da última mensagem iguale à *T*; se nenhuma mensagem não nula existir com essa propriedade, então finalmente, *(T,nula)* deve ser enviada.

3.4.2 Protocolos de Sincronização Otimistas

Nesta seção são discutidas as principais técnicas de sincronização de simulação distribuída utilizando os protocolos otimistas.

3.4.2.1 Sincronização Time Warp

Nas seções anteriores, técnicas de sincronização exclusivamente conservativas foram discutidas. Nelas os eventos são processados apenas quando existir certeza que nenhuma outra mensagem, com tempo defasado em relação ao tempo da simulação, será recebida. Todas as variações das técnicas conservativas (algumas delas discutidas na seção anterior) seguem sempre os mesmos princípios (passos de tempos fixo, exploração de *lookahead* global e/ou local, topologias fixas, mensagens nulas, etc..). Uma desvantagem das técnicas conservativas é o fato destas não oferecerem uma completa escalabilidade. Serão vistas agora algumas técnicas otimistas que permitem, entre outras características, uma maior escalabilidade. A mais conhecida refere-se ao algoritmo *Time Warp*.

No Time Warp os eventos são processados com a esperança de que nenhuma mensagem com tempo de ocorrência defasado chegue ao processo. Uma boa implementação do Time Warp possui mecanismos de gerenciamento de eventos e uma infra-estrutura que garante o mecanismo de *rollback* para cada processo lógico. O tempo de simulação para cada processo (LVT) é definido como o tempo de ocorrência do último evento processado. Quando um processo recebe uma mensagem com seu tempo no passado (defasada), o estado da simulação é retornado para o último estado consistente antes da chegada desta mensagem. Um exemplo de um processo enviando uma mensagem defasada para outro processo está representado na figura 3.5. Note, através da figura, que o processo 0 envia

uma mensagem ao processo 1 com o tempo de ocorrência menor que o tempo local do processo 1.

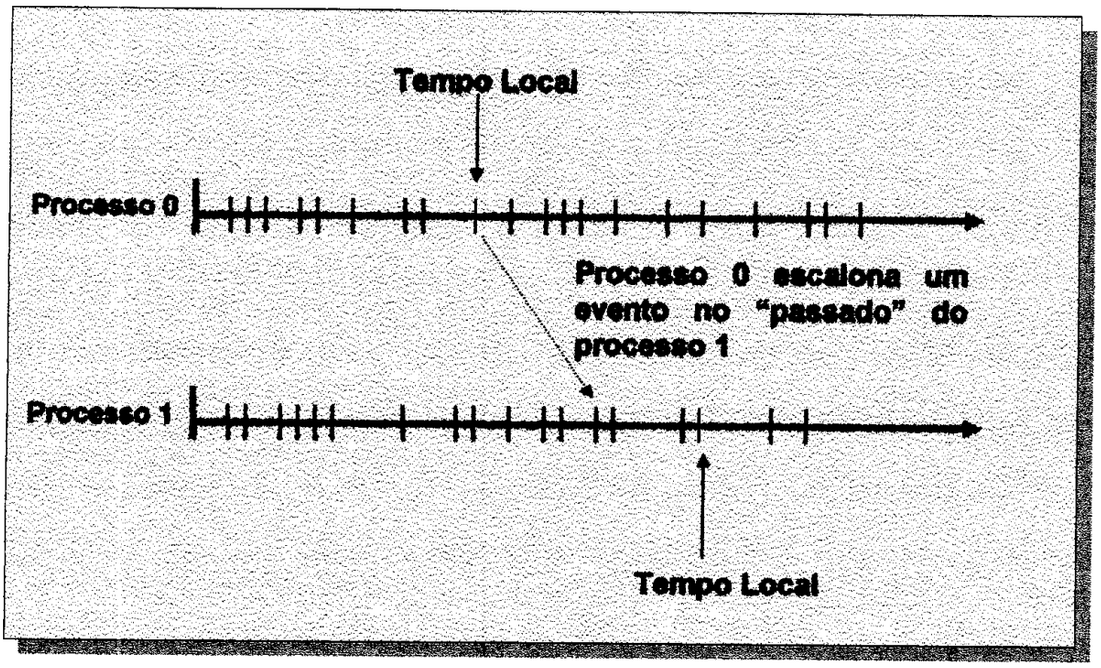


Figura 3.5 Processamento Otimista de Eventos. Representação do escalonamento de uma tarefa em um tempo passado da simulação.

Para viabilizar o *rollback*, um mecanismo de armazenamento deve ser implementado para recuperar o estado de um processo da simulação (i.e., o conjunto de variáveis que descrevem seu estado). Uma das técnicas normalmente utilizada consiste em salvar uma cópia de todo o estado do processo antes que cada evento seja processado. Esta técnica é chamada de *full state saving*. A execução de um *rollback* devido a chegada de uma mensagem atrasada deve ser acompanhada pela retomada dos valores de estado do processo anteriores a chegada desta mensagem. Esta técnica pode ser bastante complicada quando estiverem sendo utilizadas estruturas

de dados dinâmicas (como listas encadeadas) ou quando são necessárias alocações dinâmicas de memória.

Um método mais eficiente consiste em se utilizar *técnicas de salvamento de estados com base em incrementos.* Nestas técnicas os eventos devem ser retornados na ordem reversa aquela que foram processados. Essas técnicas são fornecidas diretamente pelo programador sacrificando a transparência da simulação.

Um dos conceitos mais importantes com relação a simulações otimistas diz respeito ao *Global Virtual Time* (*GVT*). GVT é definido como o menor valor entre os LVT's que compõem a simulação distribuída. Normalmente não se deve permitir que os processos escalonem novos eventos com marcas de tempos no passado, ou seja, eventos com marcas de tempo menores que o GVT. Em outras palavras, os eventos com marcas de tempo menores que a do GVT foram corretamente processados e nunca levarão a erros de causalidade. Por esse motivo deve-se determinar sempre que possível o valor do GVT sem prejudicar o processamento da simulação.

Outra razão para se realizar freqüentes atualizações no GVT é que existem simulações com interatividade bastante freqüentes. Mensagens válidas de um evento podem ser processadas seguramente quando o evento é confiável e para saber se um evento é confiável deve-se saber o valor do GVT. Se o valor do GVT for calculado muito freqüentemente (por exemplo a cada 50 ms) em uma simulação interativa em tempo real, então o tempo de resposta da interatividade podem ser menores.

Quando um evento é processado, ele pode gerar novos eventos. Estes novos eventos são escalonados através de envios de mensagens. Quando um evento é retornado , o processo da simulação deve ser reiniciado com seus valores originais e qualquer mensagem gerada deve ser cancelada através do envio de suas correspondentes antimensagens (isso significa que o Time Warp deve manter informações referentes às mensagens que foram geradas por determinado evento).

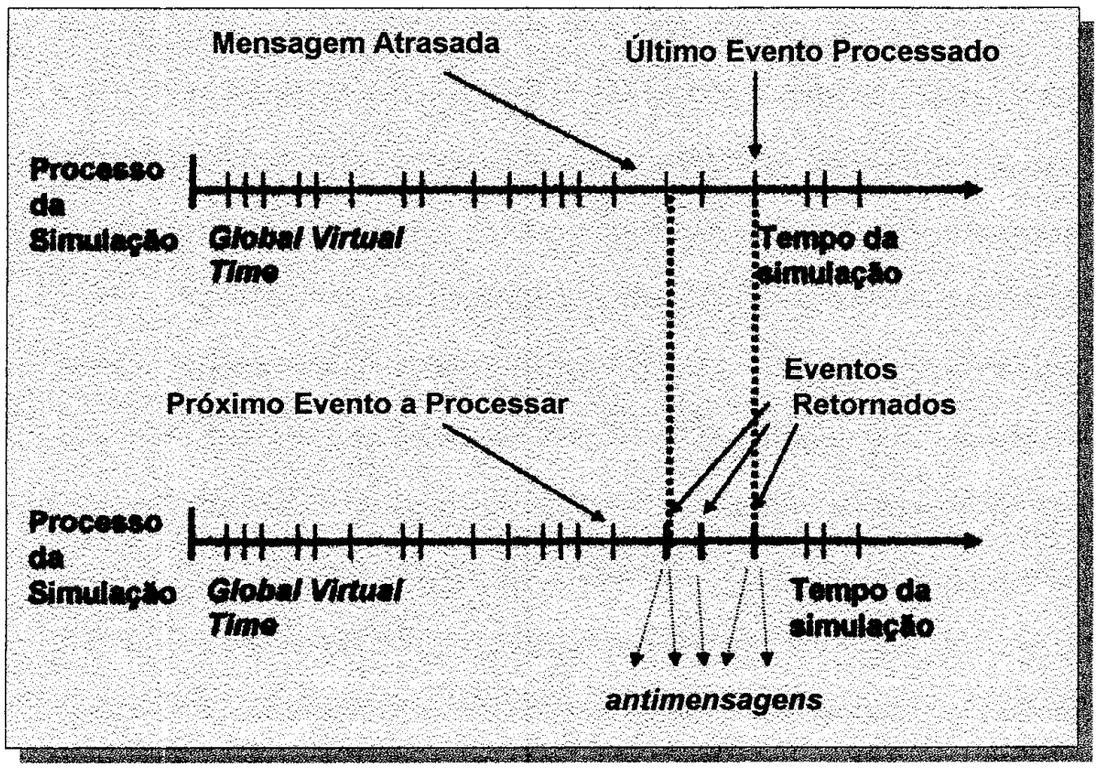


Figura 3.6 *RollBack* no Time Warp. Representação gráfica do processo de emissão de mensagens durante o *rollback*.

As figuras 3.6 e 3.7 mostram os passos envolvidos quando uma mensagem atrasada é recebida e quando uma antimensagem é escalonada por um processo. O processo de simulação é retornado ao evento da simulação que não deveria ter sido escalonado. Para corrigir este erro o

evento deverá ser retirado da lista de eventos futuros da simulação. Se por acaso este evento tenha emitido erroneamente novas mensagens, estas mensagens também deverão ser anuladas através da emissão de novas antimensagens. Esta situação poderá acarretar em uma explosão em cascata de antimensagens.

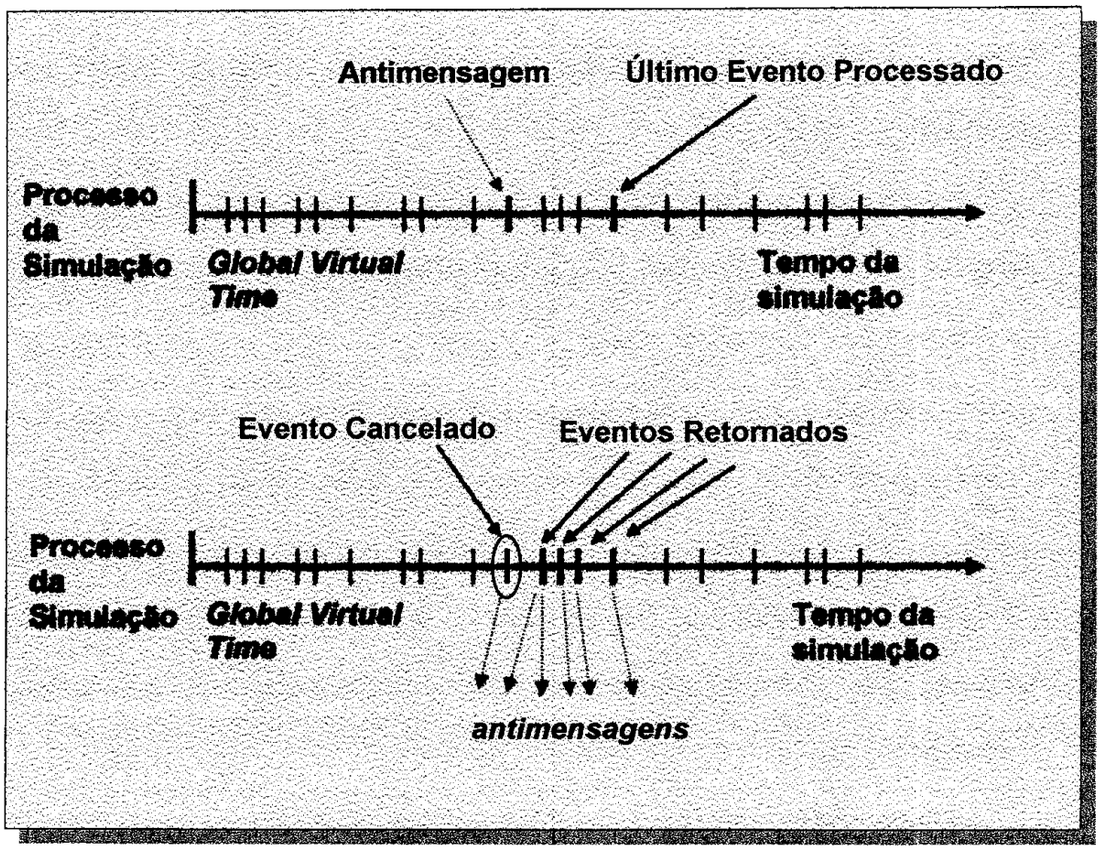


Figura 3.7 Utilização de antimensagens no Time Warp.

3.4.2.2 Sincronização Breathing Time Buckets

O Breathing Time Buckets [Steinman 1995] é um algoritmo intermediário entre o Fixed Time Buckets e o Time Warp. Assim como o

Time Warp, este algoritmo processa os eventos de forma otimista. Entretanto, ao contrário do Time Warp, as mensagens que são geradas pelos eventos não são processadas até que se tenha certeza que elas são válidas.

O conceito fundamental do algoritmo Breathing Time Buckets diz respeito ao *horizonte de eventos.* Este é um conceito também utilizado em simulações seqüenciais. Considere, por exemplo, a simulação seqüencial mostrada na figura 3.8.

Quando os eventos são processados, estes podem eventualmente gerar novos eventos. Se estes novos eventos fossem armazenados em uma lista de eventos temporária, chegaria um momento na simulação em que o próximo evento a ser processado estaria nesta nova lista temporária. O tempo de ocorrência deste próximo evento é chamado de horizonte de eventos. Este é o tempo no qual os eventos gerados pelos eventos anteriores (já processados) devem retornar à lista de eventos futuros. É realizado então uma concatenação entre os eventos gerados e os eventos pendentes.

Nota-se neste ponto que os eventos processados em cada ciclo podem ser processados em paralelo, uma vez que por definição, nenhuma mensagem chegará com seu tempo de ocorrência perto dos tempos de ocorrência dos eventos do ciclo. A questão portanto é adaptar a determinação do horizonte global de eventos assim que os eventos são processados. Para garantir esta determinação deve-se utilizar um conjunto de operações de comunicação bloqueantes e não-bloqueantes.

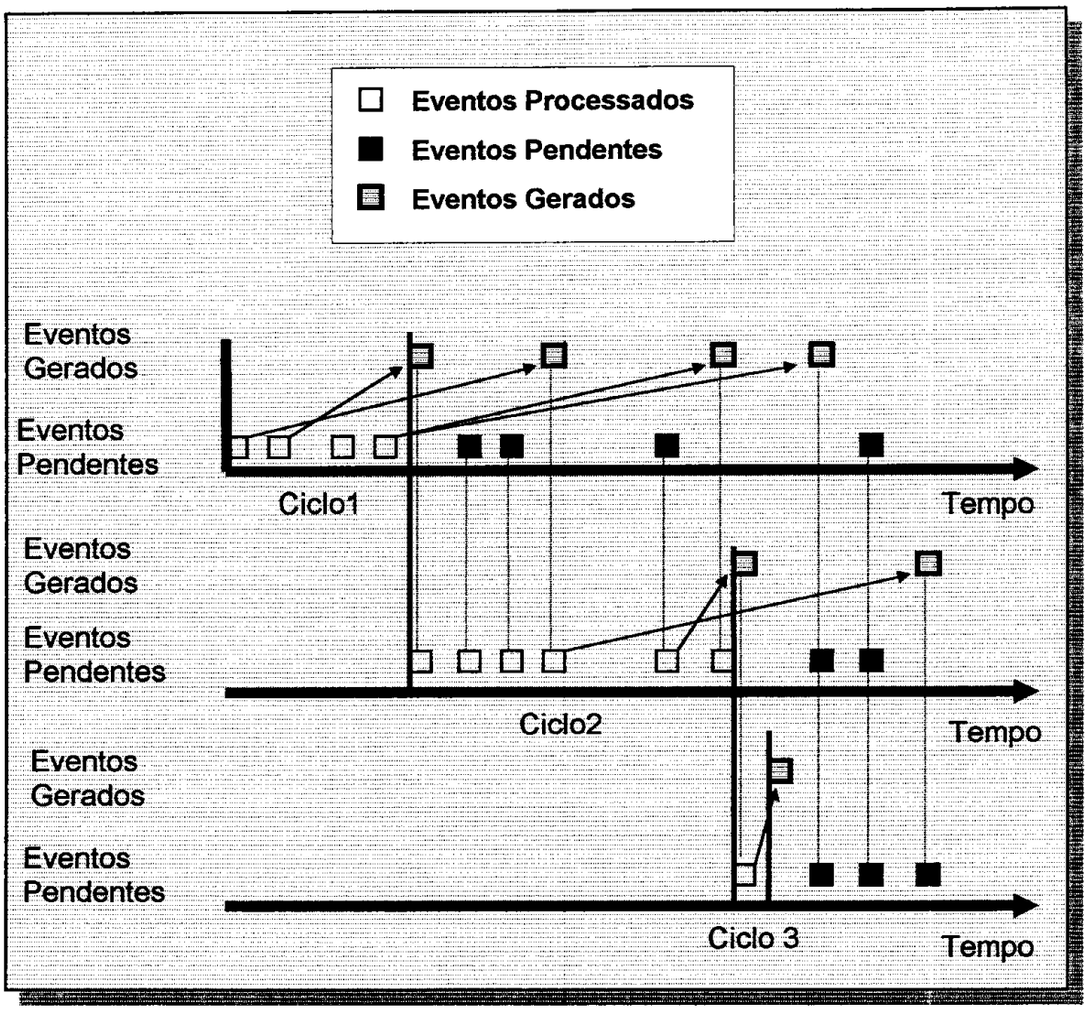


Figura 3.8 Horizonte de Eventos. A cada ciclo são considerados os eventos pendentes e os eventos gerados.

Uma das desvantagens deste algoritmo é a possibilidade da não obtenção de uma boa média de eventos a serem processados em um ciclo, prejudicando assim a eficiência da simulação.

IFSC-USP SERVIÇO DE BIBLIOTECA E INFORMAÇÃO

3.4.2.3 Sincronização Breathing Time Warp

O último algoritmo discutido nesta seção consiste em uma união das características existentes nos algoritmos Time Warp e Breathing Time Buckets fornecendo um controle direto sobre os fatores de risco e controle de fluxo em uma simulação distribuída. Se o valor do *lookahead* global for conhecido, o Fixed Time Buckets pode ser utilizado para processar os eventos da simulação com tempos de ocorrência menores que t+T com baixa sobrecarga de comunicação, de uma forma mais conservativa.

É importante ressaltar que o principal problema do Time Warp vem do fato de que as mensagens são processadas com um alto risco. O efeito cascata das antimensagens é bastante provável quando EP's "fugitivos" acabam se distanciando do resto da simulação. Um EP "fugitivo" envia uma grande quantidade de mensagens inválidas através de eventos processados com tempos de ocorrências que estão bastante adiantados em consideração ao resto da simulação. Entretanto, eventos processados perto do GVT provavelmente não irão ser afetados por mensagens vindas de EP mais lentos (atrasados).

Como regra geral, eventos processados em um tempo muito avançado em relação ao GVT tendem mais facilmente a ser retornado do que os eventos processados em um instante perto do GVT. Em alguns casos seria uma boa idéia parar o processamento do EP muito adiantado.

3.5 Conclusões

A simulação paralela, também chamada de simulação distribuída, refere-se a execução de um único programa de simulação em um computador paralelo . Esta técnica tem atraído um interesse considerável nos últimos anos. A partir de um ponto de vista pragmático, este interesse surgiu pelo crescimento da utilização da simulação na engenharia, ciências da computação, economia e aplicações militares [Fujimoto 2000, Nicol; Fujimoto 1995].

Acredita-se que a simulação distribuída representa uma técnica promissora no que diz respeito ao aumento da velocidade da simulação. A teoria básica tem sido desenvolvida; ela surge com experimentos realizados utilizando-se várias heurísticas para garantir que um bom desempenho possa ser obtido em relação a simulação seqüencial [Misra 1986].

A abordagem clássica para simulação seqüencial aborda a manipulação de uma estrutura de dados chamada de *lista de acontecimentos futuros*, de acordo com os valores de uma variável chamada *relógio* que representa a passagem do tempo na simulação [Misra 1986, MacDougall 1987].

A natureza do mecanismo da lista de acontecimentos futuros imposta pela simulação seqüencial, onde a cada ciclo da simulação apenas um item é removido da lista de acontecimentos, não torna possível a execução concorrente do algoritmo em máquinas com mais de um elemento

processador, uma vez que a lista de acontecimentos não pode ser efetivamente particionada.

Misra [Misra 1986] propõe um esquema básico onde, mesmo não existindo uma estrutura seqüencial, como a lista de acontecimentos futuros, a execução dos acontecimentos na simulação distribuída obedece a ordem seqüencial imposta pelo tempo no sistema real. O problema da ocorrência de *deadlock* e sua solução constituem o núcleo da questão do desempenho em uma simulação distribuída.

Capítulo 4

Mecanismo para Troca Dinâmica de Protocolos de Sincronização

Este capítulo aborda os conceitos que envolvem a implementação de um mecanismo de troca de protocolos de sincronização. As características dos protocolos CMB e Time Warp são apresentadas detalhadamente assim como os processos que envolvem o estudo do protocolo CMB como parte do mecanismo de troca dinâmica.

4.1 Introdução

No paradigma de simulação distribuída, o sistema modelado é representado como um conjunto de processos físicos que interagem durante

a simulação, e cada processo físico é representado na simulação por um processo lógico. Toda a interação entre os processos físicos é representada através de mensagens contendo os eventos da simulação e seus respectivos tempos de ocorrência, enviadas entre os processos lógicos por canais de comunicação. Para garantir o relacionamento de causa e efeito entre os eventos, são necessários protocolos para sincronizar os processos lógicos [Ulson 1999].

De uma forma geral, como foi mencionado no capítulo 3, estes protocolos podem ser classificados em *conservativos, otimistas* e *híbridos*.

Uma grande quantidade de pesquisas têm sido realizadas para avaliar o desempenho dos protocolos conservativos e otimistas (e suas variantes) em diferentes tipos de aplicações. Um consenso emergente tem definido o *lookahead* , a plataforma de comunicação entre os processos e a granulosidade do modelo como as principais características que definem qual tipo de protocolo a ser utilizado. Para muitos sistemas pode ocorrer destas características se contradizerem em relação a determinadas partes do sistema. Ou seja, uma parte do modelo seja eficientemente executada através de um protocolo conservativo e outra parte através de um protocolo otimista. Assim, as características que determinam qual protocolo deve ser utilizado, além de serem difíceis de serem caracterizadas, alteram-se dinamicamente [Jha 1994].

Protocolos que apresentam características unificadas dos protocolos de sincronização conservativos e otimistas têm sido propostos (*protocolos híbridos ou mistos*), entretanto, a quantidade de parâmetros a serem

ajustados previamente pelo modelador torna a utilização destes protocolos pouco viável para um modelador não familiarizado [Jha 1994].

Protocolos como o CMB [Chandy; Misra 1979] e Time Warp [Fujimoto 1990] têm sido, ao longo destes anos, implementados por uma grande quantidade de pessoas ligadas direta ou indiretamente a simulação distribuída. A praticidade e a flexibilidade destes algoritmos têm contribuído para essa condição.

Estes protocolos têm apresentado diferentes níveis de desempenho quando características como plataforma de execução e *lookahead* são modificadas.

Neste capítulo é apresentado um mecanismo que permite, durante a execução de uma simulação distribuída, a troca automática e dinâmica entre os protocolos CMB e Time Warp para realizar o sincronismo dos eventos da simulação. Desta forma, utilizando-se de dois protocolos bastante conhecidos, o sistema de simulação proposto poderá decidir, através do comportamento da simulação, qual protocolo de sincronização melhor se adapta ao contexto em que a simulação distribuída está sendo realizada.

O mecanismo para a troca de protocolos de sincronização pode ser dividido nas seguintes etapas:

1. Detectar a necessidade da troca de protocolos;
2. Garantir de manutenção da consistência da simulação no momento da troca;
3. Adaptar as estruturas para a troca de protocolos de sincronização.

Na etapa 1 deve-se determinar quando o mecanismo de troca de protocolos de sincronização deve ser considerado. Esta etapa, por sua vez, pode ser dividida em dois momentos.

O primeiro momento é quando o protocolo CMB está sendo utilizado e deve-se detectar que o Time Warp seria mais eficiente. O segundo momento é quando o Time Warp está sendo executado e o CMB seria mais eficiente. A questão de quando converter de CMB para Time Warp é tratada nos capítulos 5 e 6 desta tese. A questão relativa a detectar quando o protocolo Time Warp não está sendo eficiente é considerada em uma outra tese em andamento no Grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP.

Para tornar possível a realização do chaveamento entre os protocolos de sincronização CMB e Time Warp é necessário a manutenção da consistência da simulação e das estruturas utilizadas nos dois protocolos (etapas 2 e 3). Para definir o mecanismo de troca torna-se necessária a identificação das estruturas e processos que caracterizam a implementação dos dois protocolos considerados na construção do mecanismo de troca dinâmica.

A próxima seção deste capítulo apresenta as principais características dos dois protocolos utilizados para a construção do mecanismo de troca dinâmica de protocolos de sincronização. Baseando-se nas características apresentadas, todas as etapas realizadas para a implementação do mecanismo de troca de protocolos de sincronização em tempo de execução são discutidas na seção 4.3.

4.2 Esquema Básico para Simulação Distribuída Via Troca de Mensagens

Uma questão importante na implementação de uma simulação em uma máquina paralela, é como será simulado o tempo real do sistema , ou seja, como será garantida a coerência temporal uma vez que cada processador estará executando uma parte da simulação.

Para que se possa entender com detalhes como os protocolos CMB e Time Warp sincronizam os eventos de uma simulação distribuída é necessário discutir alguns conceitos fundamentais.

Uma abordagem é ter-se um único *clock* (variável relógio) que dirija todos os processadores da simulação. Esta abordagem é usada em simulação *dirigida por tempo*. Seu problema é que o uso de um único processo para dirigir a simulação acaba gerando um "gargalo" no sistema [Grillo 1989].

Uma abordagem alternativa é aquela na qual a simulação acontece de uma forma totalmente assíncrona, cada processo mantém seu próprio relógio local sem nenhum mecanismo de sincronização global. A comunicação entre os processos é feita através de troca de mensagens.

Para simular qualquer sistema físico (real), constrói-se um sistema lógico associando um processo lógico (*pl*) para cada processo físico (*pf*); pl_i irá simular as ações do seu correspondente pf_i.

Um *pl* pode simular as ações de um *pf* até o momento *t* se *pl* tiver conhecimento do estado inicial e de todas as mensagens que o

correspondente *pf* recebe até o momento *t*. Nenhuma mensagem futura, isto é, recebida após *t*, poderá afetar o comportamento de *pf* no tempo *t* (propriedade de *realizability*). *Realizability* indica que uma mensagem enviada por um processo no tempo *t* é uma função de seu estado inicial, e das mensagens que ele recebeu até o instante *t*. Isto garante a propriedade de causa - efeito instantâneo. Este conceito pode ser esclarecido com um pequeno exemplo. Considere um servidor típico não preemptivo e com política de atendimento FIFO, com tempo de serviço exatamente igual a 10 unidades de tempo. Assuma que um cliente chegue no tempo T quando seu servidor está vazio. A partir destas informações sobre as chegadas das mensagens até o tempo T, pode-se predizer o comportamento do servidor até o tempo T + 10: ele não irá produzir nenhuma saída entre os tempos T e T+10, mas irá produzir uma mensagem de saída no tempo T+10, enviando o serviço realizado para seu próximo destino.

A partir dessas observações, pode-se construir um algoritmo para simulação distribuída. Note que os tempos nos quais as mensagens são enviadas pelos *pf*'s devem ser incluídos nas mensagens que são enviadas pelos *pl*'s. Se uma mensagem *m* é enviada pelo pf_i ao pf_j no tempo T, a mensagem *(T,m)* será enviada pelo pl_i ao pl_j no mesmo ponto durante a simulação. Defini-se *o valor do clock de um canal* (*vcc*) como sendo o componente *t* da última mensagem recebida através daquele canal; o *vcc* terá valor zero se nenhuma mensagem tiver sido recebida através deste canal.

Todo pl_i conhece todas as mensagens recebidas por seu correspondente pf_i até o tempo $T_i = min_j\{t_j\}$, onde os t_j's são os *vcc*'s de todos os canais que chegam em pl_i, e o valor mínimo é pego entre todos esses valores. Chama-se T_i de *valor do clock* de pl_i. Por exemplo, veja a figura 4.1. Os valores de *clock* dos canais de comunicação do processo lógico são 2 e 4. O menor valor de tempo de recebimento de mensagens corresponde à mensagem recebida no tempo 2 , portanto o valor de *clock* do *processo* é 2. Consequentemente, pl_i pode seguramente simular pf_i até o tempo 2 (T_i); isto é, ele pode executar toda mensagem que pf_i envia até o tempo T_i.

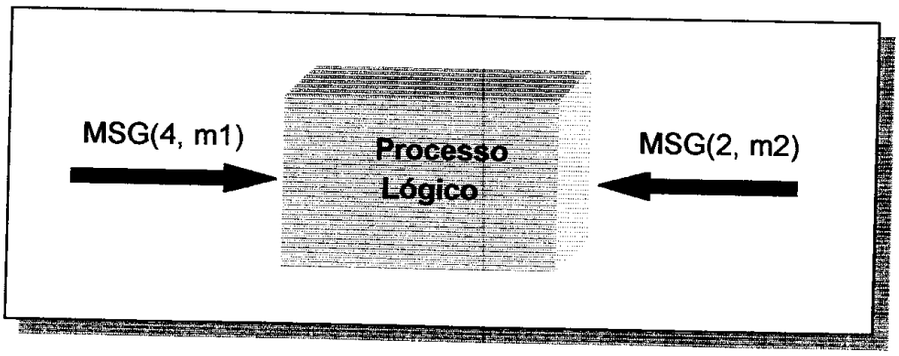


Figura 4.1 - Valor do clock de um processo. Exemplo da chegada de duas mensagens ao processo lógico. Neste caso o valor do clock do processo assume o valor da segunda mensagem.

Na simulação seqüencial pode-se declarar que o simulador completou a simulação até o tempo fornecido pelo *clock*: todos os processos físicos devem ter sido simulados até este tempo. Com a simulação distribuída não

ocorre o mesmo, porque cada processo lógico pode ter simulado seu correspondente processo físico em um ponto diferente do tempo.

O *valor do clock do simulador* (*T*) é o menor valor de *clock* de todos os processos lógicos. Pode-se declarar que em um determinado ponto da simulação, o sistema físico tenha sido simulado até o tempo marcado por *T*, ainda que algum processo lógico tenha simulado seu correspondente processo físico até um tempo maior que *T*.

O algoritmo descrito anteriormente considera um protocolo genérico para sincronizar as mensagens de uma simulação distribuída. Todo protocolo de sincronização segue de uma forma ou outra os princípios básicos definidos neste algoritmo.

A maneira com que o protocolo de sincronização reage frente a uma infração da propriedade de causa – efeito serve como parâmetro principal na sua classificação entre protocolo *Otimista* ou *Conservativo*.

A seguir são descritas as implementações de dois protocolos bastante utilizados (CMB e Time Warp) para sincronizar uma simulação distribuída e que constituem os principais representantes destas categorias.

4.2.1 Protocolo CMB

A principal característica dos protocolos conservativos é a ausência da possibilidade de um erro de causalidade. No protocolo CMB um processo lógico não pode executar um evento a menos que haja a certeza de que não chegará nenhuma mensagem com um tempo de ocorrência menor.

Determinar quando é seguro processar um evento ou não é uma questão complexa, uma vez que depende das características do modelo simulado. Nesses protocolos, a capacidade de *lookahead* do modelo (a capacidade de prever o comportamento futuro e determinar a ocorrência dos eventos) constitui o elemento chave na determinação do desempenho do algoritmo. Em função da importância da determinação do *lookahead* para o desempenho da simulação conservativa, muitos protocolos foram desenvolvidos para auxiliar no processo de determinação do *lookahead* [Bagrodia et al. 1998, Fujimoto 1990, Nicol 1996].

Com o intuito de facilitar a comparação entre as estruturas utilizadas nos dois protocolos de sincronização desenvolveu-se uma estrutura lógica para cada um dos protocolos, apresentadas nas figuras 4.2 e 4.3.

No protocolo CMB as mensagens são recebidas através de canais de comunicação fixos (topologia estática). Para cada canal (ou porta) de chegada de mensagens existe uma estrutura de dados ([1]*IB[i]- Input Buffer*) responsável pelo armazenamento das mensagens que chegam ao processo (Figura 4.2). Da mesma forma a estrutura OB[j] (*Output Buffer)* encarrega-se de armazenar as mensagens que deverão ser enviadas a outros processos. As mensagens recebidas são armazenadas por ordem do tempo de ocorrência e o valor do menor tempo de ocorrência é armazenado em uma estrutura de dados chamada *clock do canal* (*CC[i] – Channel Clock*). A cada ciclo da simulação a mensagem correspondente ao menor valor entre os CC[i]'s é inserida na lista de ventos futuros (*EVL – Event List*). Para facilitar a

---

[1] As siglas serão mantidas em suas versões em inglês por ser a forma usual de utilização empregada pela comunidade ligada a simulação distribuída.

identificação do menor valor entre os CC[i]'s este valor é calculado e armazenado na variável LVTH (*Local Virtual Time Horizon*).

A função da EVL é armazenar os eventos que serão processados em ordem do seu tempo de ocorrência. Cada evento (ou mensagem) retirado da EVL corresponde a um avanço no relógio do processo (*LVT – Local Virtual Time).*

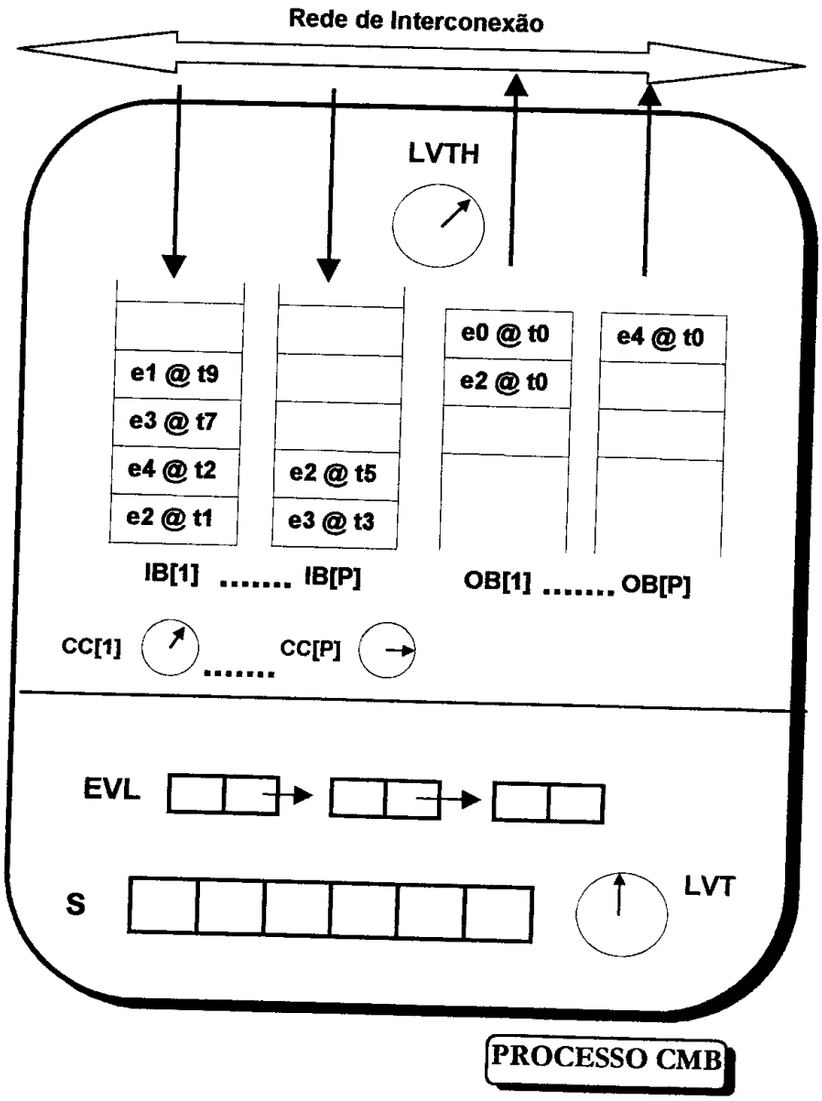


Figura 4.2 Um Processo Lógico CMB. Esquema de representação de um processo CMB com suas estruturas e variáveis de tempo.

A estrutura de dados S (*Status*) encarrega-se de armazenar as informações estatísticas relacionadas ao contexto do processo lógico.

Durante a execução da simulação os processos envolvidos no modelo podem, por um período indeterminado de tempo, não possuir mensagens a serem enviadas. Esta condição é suficiente para que ocorra um *deadlock* e assim a simulação fique bloqueada. Para evitar essa situação o protocolo CMB pode se utilizar do mecanismo de *mensagens nulas*.

Uma mensagem nula enviada por um pl_i a um pl_j, significa que pf_i não envia nenhuma mensagem a pf_j entre o valor corrente do *clock* do canal (de pl_i a pl_j) e o tempo *t*; por essa razão, qualquer futura mensagem de pl_i para pl_j terá o valor do componente tempo excedendo *t*. É óbvio que essas mensagens nulas não possuem nenhuma correlação com o sistema físico. Uma mensagem nula é utilizada para indicar a ausência de mensagens. Essa ausência de mensagens no sistema físico no tempo *t* é reconhecida pela não emissão de mensagens neste instante *t*.

Desta forma a marca de tempo da mensagem nula irá causar uma atualização no valor do processo lógico que a receberá, permitindo desta forma o avanço do tempo da simulação.

4.2.2 Protocolo Time Warp

Ao contrário do que ocorre com o protocolo CMB, no protocolo de sincronização Time Warp existe a probabilidade de erros de causalidade na execução dos eventos. O conceito otimista assume que nenhuma

mensagem com tempo de ocorrência menor que o LVT (*Local Virtual Time*) local será recebida. Entretanto quando essa situação ocorre, um erro de causalidade é detectado e o processo lógico é obrigado a restaurar um estado anterior (consistente) desfazendo-se assim as alterações provocadas pela mensagem prematura (mecanismo de *rollback)* [Damian 1987]. Desta forma, após isso, os eventos são executados novamente na ordem correta de tempo de ocorrência. No protocolo Time Warp o menor valor entre os tempos de ocorrência das mensagens processadas é chamado de *Global Virtual Time* (*GVT*). Nenhum evento com tempo de ocorrência menor que GVT será, por conseqüência da chegada de uma mensagem defasada, executado novamente. As estruturas IB (*Input Buffer*) e OB (*Output Buffer*) são responsáveis pelo recebimento e envio das mensagens através da rede de comunicação (figura 4.3).

Para permitir o processo de *rollback* o protocolo Time Warp conta com uma estrutura de dados onde são armazenados os valores de estados do processo. Essa estrutura de estados (*SS*) permite que a simulação possa ser restaurada a um estado consistente caso haja um erro de causalidade. O processo de restauração de um estado consistente também envolve a atualização da estrutura S (*Status)* que mantém dados estatísticos referentes a simulação do modelo da aplicação. O objetivo do armazenamento temporário das mensagens pelas estruturas *IQ* (*Input Queue*) e *OQ* (*Output Queue*) é corrigir, através das antimensagens, um possível engano causado pelo processamento de uma mensagem fora da ordem cronológica.

Para cada mensagem existe uma *antimensagem* cujo conteúdo é idêntico com exceção ao campo de *sinal* (mensagem possui sinal + e antimensagem possui sinal -) [Jefferson 1985]. Uma antimensagem tem como objetivo informar ao processo lógico que a sua mensagem correspondente está contrariando o princípio de causalidade e por isso o processo deve desfazer os efeitos causados pela mensagem infratora.

Uma vez discutidas as características dos algoritmos dos protocolos CMB e Time Warp, a seção 4.2.3 discute as vantagens e desvantagens destes protocolos.

4.2.3 Protocolos Conservativos Versus Otimistas e Protocolos Híbridos

A questão relativa aos méritos dos protocolos conservativos e otimistas tem sido freqüentemente abordada. Entretanto existe um consenso geral de que a formulação de regras para determinar a superioridade de um protocolo em relação a outro é uma tarefa bastante difícil [Ferscha 1995].

Embora alguns resultados analíticos tendam a favor dos protocolos otimistas, os protocolos conservativos podem arbitrariamente superar o desempenho dos protocolos otimistas em determinadas aplicações.

A possibilidade de explorar mais convenientemente o paralelismo e utilizar uma topologia de comunicação dinâmica são características que favorecem os protocolos otimistas. Por outro lado a necessidade de armazenamento dos estados de cada processo lógico representa um grande obstáculo a ser transposto e que tem despertado muito interesse entre os

pesquisadores. Apesar de até o momento haver uma restrição causada pela necessidade da utilização de uma rede de comunicação estática, estudos mostram que a conectividade dinâmica também é possível para os protocolos conservativos [Jha 1994]. Por outro lado a utilização dos protocolos conservativos é altamente dependente do *lookahead*, uma garantia de que o processo lógico não irá interagir com outros processos lógicos por um período específico de tempo. Calcular o valor ou obter o comportamento do *lookahead* na maioria das vezes é uma tarefa bastante difícil.

Esta questão é puramente teórica e representa uma visão para alguns casos extremos. Além disso as análises são baseadas em suposições como custos bem comportados de *rollback* e quantidade zero de mensagens nulas [Das 1996].

Alguns protocolos que assumem uma política mista entre os protocolos otimistas e conservativos têm sido propostos (*protocolos híbridos*). Intuitivamente estes protocolos utilizam um "otimismo calculado" que permite explorar os benefícios dos protocolos otimistas sem suas inconveniências. Entretanto, é difícil para o usuário da simulação, que não é intimamente familiarizado com os protocolos de sincronização e com as arquiteturas paralelas, ajustar os parâmetros apropriados para obter um bom desempenho.

Como foi mencionado anteriormente, a proposta deste trabalho é fornecer condições ao usuário da simulação distribuída alterar, durante a execução da simulação e através de um mecanismo que reuna informações

a respeito do desempenho da simulação, o protocolo utilizado para a sincronização da simulação distribuída.

A descrição do mecanismo de troca, assim como as alterações necessárias em cada um dos protocolos participantes (CMB e Time Warp) são descritas na seção 4.3.

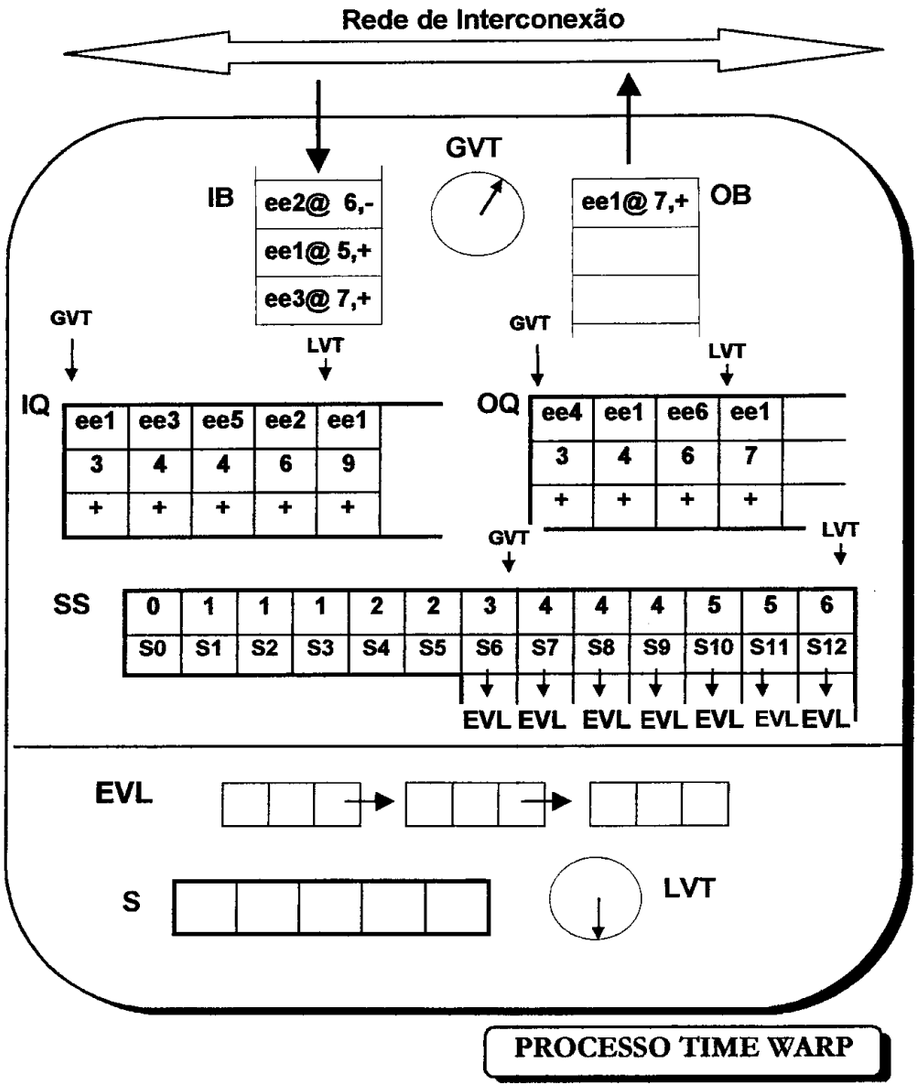


Figura 4.3 Um Processo Lógico Time Warp. Estruturas e variáveis de um processo Time Warp.

4.3 Mecanismo de Troca de Protocolos de Sincronização para Simulação Distribuída

Durante a execução da simulação distribuída, independente de qual protocolo esteja sendo utilizado, é necessário que as informações relativas às variáveis que medem o desempenho da simulação (abordadas detalhadamente nos capítulos 5 e 6) sejam obtidas e analisadas para que se possa decidir se existe a necessidade da troca de protocolos.

Para que isto seja possível foram definidos três processos lógicos cuja função desempenhada está intimamente relacionada com o mecanismo de troca de protocolos de sincronização não tendo nenhuma participação no contexto da simulação do modelo em questão.

Sendo assim, um sistema de simulação que utilize o mecanismo de troca de protocolos de sincronização deve possuir uma implementação do protocolo otimista Time Warp, uma implementação do protocolo conservativo CMB, um processo *gerenciador*, um processo *conversor* e um processo *observador* [Morselli et al. 2000]. As funções de cada um destes três processos adicionais são descritas a seguir.

1. *Processo Observador.*

Para que o momento da troca de protocolos de sincronização seja identificado é necessário reunir informações a respeito do desempenho da simulação em cada um dos processos lógicos envolvidos na simulação distribuída. A quantidade de mensagens nulas e completas transmitidas ou

recebidas, a porcentagem de mensagens bloqueadas quando atingem a cabeça da lista de eventos futuros, os valores dos LVT's de cada um dos processos, o número de *rollbacks* e o número de antimensagens são exemplos de informações que podem ser obtidas pelo processo observador no contexto geral da simulação para que uma possível troca de protocolos seja considerada.

As informações que o processo observador deverá obter depende da política adotada para definir a conversão de protocolos de sincronização. Esta política, por sua vez, depende do protocolo que está sendo executado. Para o caso do protocolo CMB, a política de troca é analisada nos capítulos 5 e 6 desta tese. Para o caso do protocolo Time Warp, um estudo está sendo efetuado em outra tese em andamento.

O processo observador deve estar sendo executado concorrentemente com cada processo lógico e a cada intervalo de tempo, deverá enviar estes valores para o processo gerenciador.

2. *Processo Gerenciador*

O processo gerenciador tem a função de receber essas informações dos processos observadores e através da identificação do comportamento do protocolo utilizado, decidir pela realização ou não da troca de protocolos.

3. *Processo Conversor*

O processo conversor é responsável pela verificação da consistência na simulação e pela manipulação das estruturas de dados e variáveis de tempo dos processos lógicos no momento da troca de protocolos.

Uma instância deste processo deve estar executando concorrentemente com cada processo lógico participante da simulação.

A verificação da consistência refere-se a deixar a simulação em um estado a partir do qual a conversão de protocolos possa ser considerada. Esta necessidade é uma consequência da diferença existente entre os paradigmas (conservativo e otimista) adotados pelos protocolos CMB e Time Warp respectivamente. Um exemplo de inconsistência é quando o Time Warp está sendo utilizado e as antimensagens precisam ser eliminadas e os LVT's igualados. A seção 4.3.1 detalha as formas de se manter a consistência.

A conversão das estruturas de dados refere-se às modificações dos dados responsáveis pelo armazenamento das mensagens, utilização das variáveis GVT e LVTH, etc..A figura 4.4 mostra a estrutura dos processos envolvidos no sistema de simulação utilizando a troca dinâmica de protocolos.

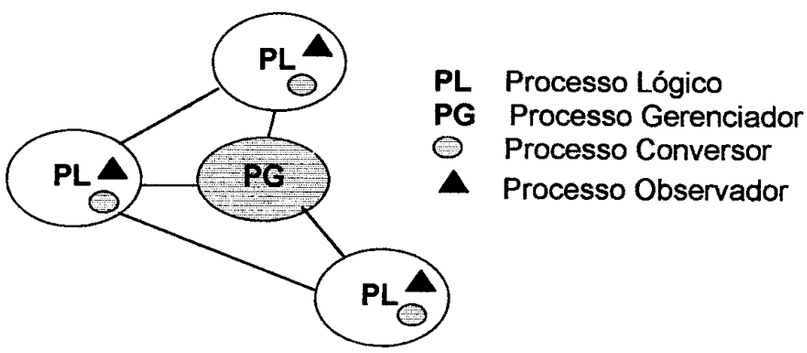


Figura 4.4 Configuração dos processos da simulação distribuída utilizando o mecanismo de troca dinâmica de protocolos. A função do processo observador é enviar as informações relativas ao desempenho de cada processo lógico da simulação ao processo gerenciador. O processo conversor é responsável pelas manipulações do conjunto de estruturas e variáveis de cada protocolo no momento da troca.

Desta forma, uma instância do processo conversor e uma instância do processo observador para cada processo lógico estão ativos durante a execução da simulação. No momento em que uma conversão é necessária, o processo conversor é ativado (figura 4.5).

As próximas seções detalham o processo conversor, considerando o processo para garantir a consistência da simulação (seção 4.3.1) e o mecanismo para conversão das estruturas de dados (seção 4.3.2).

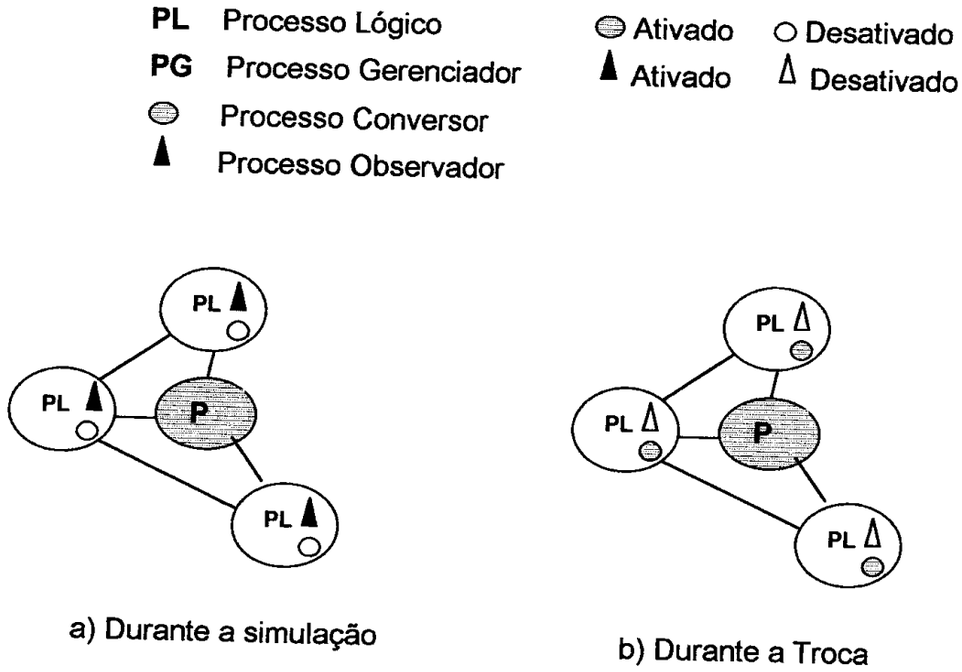


Figura 4.5. Mecanismo de ativação dos processos envolvidos no Mecanismo de Troca. Durante a execução da simulação os processos observadores transmitem informações ao processo gerenciador informando o desempenho de cada processo lógico (a). No momento da troca de protocolos os processos conversores são ativados para realizar a manipulação das estruturas de dados e variáveis.

4.3.1 Garantia da Consistência

Para que não ocorra erros de causalidade, o mecanismo de troca de protocolos de sincronização deve prever a mudança de protocolo apenas quando a simulação estiver em um estado consistente. Entende-se por estado consistente a impossibilidade de ocorrência de um processo de recuperação de estados (*rollback*- Time Warp) ou bloqueio (*deadlock* – CMB) para que a troca seja efetivamente realizada.

- **Garantia de consistência na conversão CMB para Time Warp**

Uma simulação distribuída sincronizada por um protocolo conservativo estará consistente caso não esteja bloqueada, desta forma a utilização do mecanismo de mensagens nulas garante a consistência de uma simulação distribuída sincronizada pelo protocolo CMB em qualquer momento durante sua execução.

Essa garantia fornecida pelo mecanismo de mensagens nulas possibilita a troca de CMB para Time Warp em qualquer instante da simulação.

- **Garantia de consistência na conversão Time Warp para CMB**

Ao contrário do que ocorre com uma simulação distribuída sincronizada pelo protocolo CMB, onde as mensagens nulas oferecem a garantia de consistência dos processos durante toda a simulação, o paradigma otimista adotado pelo protocolo Time Warp não garante que , em qualquer instante a simulação esteja consistente. Um protocolo otimista considera a execução

de todas as mensagens independentemente dos seus valores de tempo de ocorrência. Partindo deste princípio, a execução de uma mensagem cronologicamente incorreta deve disparar um processo de reexecução (*rollback*) de todas as mensagens recebidas com tempo de ocorrência menor que o tempo de ocorrência da mensagem defasada. A possibilidade de ocorrer a necessidade de reexecução das mensagens impede que o processo de troca de protocolos seja realizado.

Existem duas maneiras nas quais a garantia de consistência durante a fase Time Warp pode ser obtida. A primeira consiste em avançar a simulação até que os processos lógicos atinjam o valor do maior LVT. Para a implementação desta abordagem, as diversas instâncias do processo observador devem incluir o LVT de seu processo lógico no envio das informações para o processo gerenciador.

Ao detectar uma necessidade de conversão do protocolo Time Warp para o protocolo CMB, o processo gerenciador deve ativar o processo conversor e informar-lhe o maior valor de LVT recebido.

O processo conversor permite que o processo lógico execute até seu LVT atingir o valor recebido do processo gerenciador e só então bloqueia o processo lógico.

Esta primeira opção apresenta algumas desvantagens. Além de ocorrer uma sub-utilização do processo lógico que possui o maior LVT (causada pela espera de que os outros processos lógicos atinjam sua marca) os outros processos lógicos teriam que aguardar as chegadas das mensagens futuras para que seus LVT's fossem incrementados até o maior LVT. Neste período de espera três situações podem ocorrer:

1. A chegada de uma ou mais mensagens fora da ordem cronológica e ocorrer um ou mais rollbacks;
2. Uma demora muito grande para as chegadas das mensagens que atualizam os LVT's;
3. Pelo princípio de causa-efeito, existe a possibilidade de que um desses processos lógicos, que deveriam avançar seus LVT's, necessitar de um determinado evento que o processo lógico que possui o maior LVT deveria causar, mas este estaria parado aguardando que todos os PI's atingissem a sua marca de LVT para que ocorresse a troca.

A ocorrência de uma ou mais dessas situações provocaria uma sobrecarga de processamento muito grande prejudicando o desempenho do mecanismo de troca.

A segunda maneira de garantir a consistência da simulação durante a fase Time Warp parte da própria definição do GVT (*Global Virtual Time*). O GVT é a variável de tempo cujo valor é o menor entre todos os LVT (*Local Virtual Time*) da simulação. Desta forma o valor do GVT indica qual período de tempo o modelo lógico (a simulação) simulou o modelo físico e que é impossível a chegada de mensagens com valores de tempo de ocorrência menores que o GVT. Sendo assim a realização de um *rollback* intencional de todos os processos Time Warp até a marca de tempo fornecida pelo GVT garante que a simulação atingirá um estado consistente.

Para realizar a implementação desta abordagem, uma vez que cada processo lógico da simulação possui o valor do GVT, seria necessário apenas que os processos conversores, ao receberem a ordem de conversão

do protocolo Time Warp para o protocolo CMB, transmitissem uma antimensagem com o valor do GVT aos processos lógicos da simulação. Desta forma, o próprio mecanismo de *rollback* do Time Warp se encarregaria de levar todos os processos lógicos até o valor do GVT.

Dependendo do estado geral da simulação, a sobrecarga de processamento exigida pelos *rollbacks* realizados pelos processos otimistas pode, em função do baixo desempenho da simulação apresentada pelo protocolo Time Warp e/ou de uma curta diferença entre o valor do GVT em relação aos LVT's, não interferir no desempenho final da simulação como interferiria a primeira opção discutida anteriormente.

Desta forma, a realização intencional dos *rollbacks* para todos os processos lógicos até o valor do GVT para garantir o estado consistente e permitir a realização da troca de Time Warp para CMB parece ser mais adequada. No entanto, uma quantificação dos prós e contras de cada uma delas deverá ser abordada em um trabalho futuro.

4.3.2 Mecanismo de Conversão

Para cada classe de protocolos existe um conjunto de características que devem ser consideradas no momento da troca de protocolo. E desta forma três pontos devem ser tratados na conversão:

a) As diferenças entre as estruturas de dados necessárias nos dois protocolos considerados (seção 4.3.2.1);

b) A adaptação da topologia lógica da rede de comunicação utilizada pelos protocolos de sincronização (seção 4.3.2.2);

c) A necessidade da compatibilização das variáveis de tempo empregadas para manter a ordem cronológica da execução dos eventos (seção 4.3.2.3);

4.3.2.1 Estruturas de Dados

Uma das principais características com relação a implementação dos protocolos de sincronização refere-se às estruturas de dados que auxiliam no gerenciamento das mensagens utilizadas na simulação. Para cada protocolo de sincronização existe um conjunto de variáveis e estruturas de dados diferente, ou seja, que faz parte de um determinado protocolo mas que não possuem similares no outro protocolo.

A troca de protocolos durante a execução da simulação exige que estruturas que não existiam anteriormente em um processo lógico CMB, por exemplo, passem a fazer parte deste processo e vice e versa. De um forma geral, esta tese propõe o emprego de um mecanismo complementar de utilização de estruturas de dados [Morselli et al. 2000]. Isto é, a maior parte das estruturas de dados foram reunidas em apenas um conjunto de estruturas. Sendo assim, um processo Time Warp, por exemplo, irá conter além de suas estruturas de dados originais, as estruturas de dados de um processo lógico CMB. Estas estruturas deverão ser declaradas inicialmente e ativadas (utilizadas) em tempo de execução no momento da troca de protocolos.

A seguir são analisadas cada estrutura de dados utilizada nos protocolos considerados e como são manipuladas durante a conversão.

Lista de Eventos Futuros (EVL)

A lista de eventos futuros contém os eventos escalonados (local ou remotamente) para serem executados pelo processo lógico. Em uma simulação distribuída cada processo lógico contém sua própria lista de eventos. Nela os eventos estão ordenados pelo seu tempo de ocorrência e a cada iteração da simulação o evento com o menor tempo de ocorrência é retirado da lista e processado.

- **Transição CMB → Time Warp**

Durante a execução da simulação no modo CMB, a estrutura da lista de eventos futuros estará em permanente estado de consistência pois, como foi mencionado anteriormente, este estado é garantido pelo mecanismo de mensagens nulas. Desta forma, a lista de eventos futuros deverá manter, após a troca de protocolos, o conjunto de eventos armazenados durante a execução no momento anterior ao processo de troca. Essa operação não exige nenhum esforço computacional uma vez que a estrutura da EVL é idêntica nos dois protocolos.

- **Transição Time Warp → CMB**

O paradigma otimista utilizado pelo protocolo de sincronização Time Warp proporciona sucessivos períodos de consistência na simulação. Ou seja, após um período de estabilidade pode ocorrer um período de inconsistência provocado pela execução de um *rollback* em função da chegada de uma mensagem defasada.

Lista de Eventos Futuros (EVL)

A lista de eventos futuros contém os eventos escalonados (local ou remotamente) para serem executados pelo processo lógico. Em uma simulação distribuída cada processo lógico contém sua própria lista de eventos. Nela os eventos estão ordenados pelo seu tempo de ocorrência e a cada iteração da simulação o evento com o menor tempo de ocorrência é retirado da lista e processado.

- **Transição CMB → Time Warp**

Durante a execução da simulação no modo CMB, a estrutura da lista de eventos futuros estará em permanente estado de consistência pois, como foi mencionado anteriormente, este estado é garantido pelo mecanismo de mensagens nulas. Desta forma, a lista de eventos futuros deverá manter, após a troca de protocolos, o conjunto de eventos armazenados durante a execução no momento anterior ao processo de troca. Essa operação não exige nenhum esforço computacional uma vez que a estrutura da EVL é idêntica nos dois protocolos.

- **Transição Time Warp → CMB**

O paradigma otimista utilizado pelo protocolo de sincronização Time Warp proporciona sucessivos períodos de consistência na simulação. Ou seja, após um período de estabilidade pode ocorrer um período de inconsistência provocado pela execução de um *rollback* em função da chegada de uma mensagem defasada.

Todo processo de *rollback* é iniciado quando uma antimensagem atinge a cabeça da lista de eventos futuros e posteriormente é executada. Sendo assim, mesmo com a simulação em um estado consistente pode existir uma ou mais antimensagens escalonadas na lista de eventos futuros.

Já foi mencionado anteriormente que para que a troca de Time Warp para CMB seja realizada é necessário retroceder os relógios de todos os processos ao valor da variável GVT para garantir a entrada do protocolo CMB com a simulação em um estado consistente. Esta volta das variáveis de tempo dos processos, realizado pelo processo conversor, garante que nenhuma antimensagem fará parte da lista de eventos futuros utilizada pelo protocolo CMB e caso ocorra a chegada de uma mensagem defasada no tempo da simulação o protocolo CMB se encarregará de bloquear a simulação.

Com relação a estrutura das mensagens utilizadas pelo mecanismo de troca deve-se considerar que, se por um lado as mensagens do protocolo CMB possuem informações sobre o identificador da tarefa, o tempo em que a tarefa deverá ser executada (tempo de ocorrência) e o número do evento a ser processado, por outro lado, as mensagens que fazem parte da implementação do protocolo Time Warp possuem, além destas informações, um campo para o sinal de identificação (mensagem ou antimensagem), identificador do processo remetente e identificador do processo destinatário. Por esta razão a estrutura de mensagens utilizada pelo protocolo Time Warp foi mantida para a implementação do mecanismo de troca (Figura 4.6).

Na conversão de CMB para Time Warp o processo conversor deverá percorrer a EVL e inicializar os seguintes campos:

- *Processo Origem.* Esta informação não está disponível, normalmente, no CMB, logo deverá ser incluída;
- *Processo Destino.* Esta informação é necessária para o envio de antimensagens. Como até o momento da conversão de protocolos, a não ocorrência de erros de causalidade estava garantida, esta informação não era utilizada mas após a troca deve ser considerada;
- *Sinal de Identificação.* Todos os eventos da EVL devem ser identificados como positivos, uma vez que ainda não existem antimensagens.

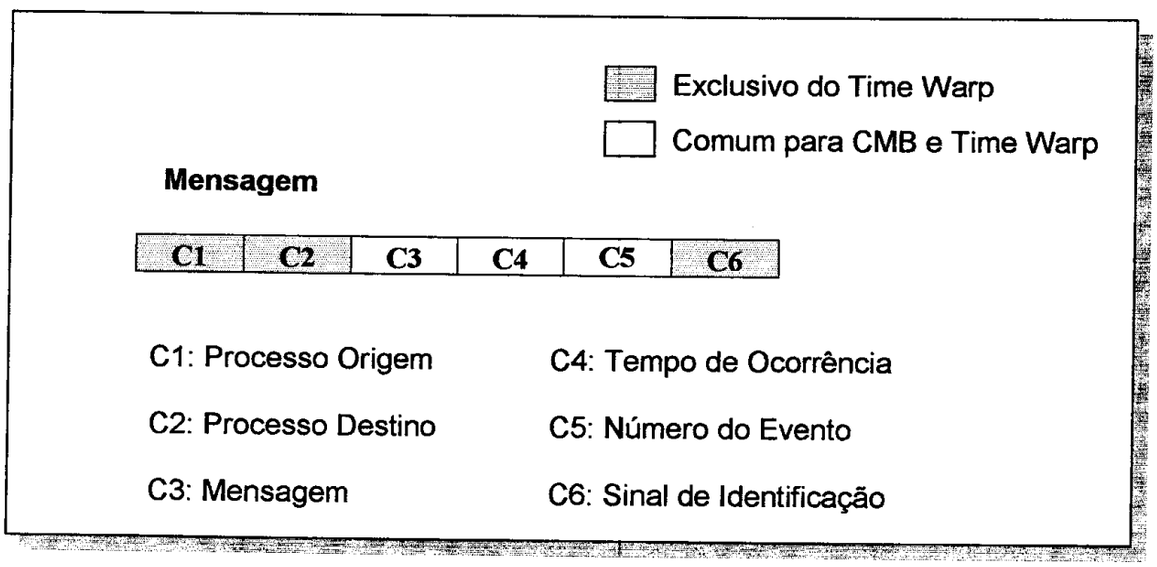


Figura 4.6 Mensagem Utilizada pelo Mecanismo de Troca de Protocolos de Sincronização. Utilização de uma mensagem que permite a compatibilização entre os dois protocolos de sincronização.

Portas de Entrada e Saída de Mensagens

As estruturas do protocolo CMB *input buffer (IB[i])* e *output buffer (OB[j])*, também chamadas de *portas*, são responsáveis pelo armazenamento temporário (em ordem cronológica) das mensagens recebidas/enviadas por um determinado canal de comunicação. A variável IB[i] armazena as mensagens recebidas pelo processo através do canal i e a variável OB[j] armazena as mensagens que deverão ser enviadas pelo canal j. No Time Warp estas variáveis são substituídas pelas estruturas *input buffer* (*IB*) e *output buffer* (*OB*). Todas as mensagens recebidas pelo processo Time Warp são inseridas em IB e as mensagens que o processo Time Warp deve enviar são armazenadas em OB (não necessariamente em ordem cronológica). Nota-se que essas estruturas, tanto no protocolo CMB como no protocolo Time Warp, exercem as mesmas funções (armazenar temporariamente as mensagens que chegam ou partem do processo lógico). Porém existe uma incompatibilidade consequente do número de estruturas empregadas em cada um dos protocolos. Sendo assim, todos os processos lógicos utilizados pelo mecanismo de troca manterão todas as estruturas IB[i] e OB[j] utilizadas pelo protocolo CMB, porém quando o protocolo Time Warp estiver em vigor, fica determinado que apenas a primeira posição de cada uma delas seja utilizada (simulando a função das estruturas IB e OB do Time Warp) .

- **Transição CMB → Time Warp**

No momento da troca CMB para Time Warp todas as mensagens armazenadas na estrutura IB[i] deverão ser transferidas para a estrutura IB[1] (Figura 4.7). Da mesma forma as mensagens armazenadas na estrutura OB[j] passarão a pertencer a estrutura OB[1] e a partir deste momento o protocolo Time Warp estará utilizando apenas essas duas estruturas para armazenar temporariamente as mensagens recebidas e as que serão enviadas.

- **Transição Time Warp → CMB**

Quando a troca de protocolos for no sentido Time Warp para CMB as estruturas IB[i] e OB[j] deverão ser utilizadas plenamente (não apenas as posições IB[1] e OB[1]). Sendo assim, o processo conversor se encarregará de habilitar os canais de comunicação que fazem a comunicação com as portas IB[i-1] e OB[j-1]. Neste momento cada processo lógico é informado que as mensagens que deverão ser escalonadas remotamente (de um processo lógico para outro) devem ser direcionadas para os meios de comunicação que fazem a ligação com o processo lógico destino. Desta forma, a partir deste momento cada processo lógico passará a armazenar suas mensagens em cada uma das posições de IB[i] e OB[j].

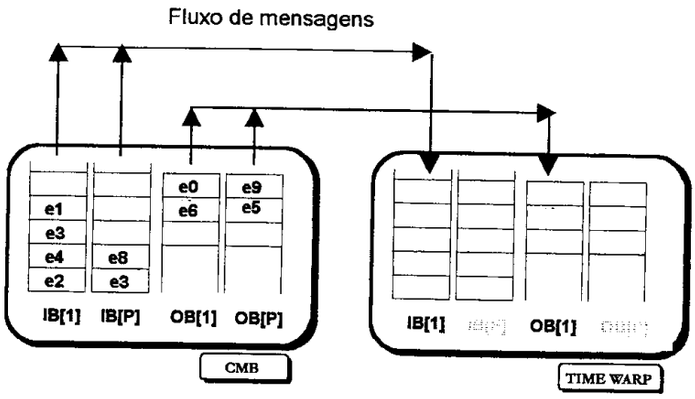


Figura 4.7 Fluxo de Mensagens durante a Troca CMB para Time Warp. As estruturas IB[1] e OB[1] recebem as mensagens armazenadas nas estruturas IB[i] e OB[j].

Estruturas IQ (*Input Queue*) e OQ (*Output Queue*)

Todas as mensagens enviadas ou recebidas por um processo Time Warp são armazenadas por um determinado período de tempo nas estruturas IQ e OQ. Isto ocorre porque quando uma mensagem cronologicamente defasada é recebida por um processo Time Warp, este processo deve transmitir uma ou mais antimensagens para que as alterações provocadas pelas mensagens emitidas antes do recebimento da mensagem defasada sejam novamente processadas na ordem cronológica correta.

Desta forma, o armazenamento das mensagens recebidas e transmitidas permite a correção de um possível erro de causalidade causado pelo processamento de uma mensagem fora da ordem cronológica.

As mensagens mantidas nas estruturas IQ e OQ são aquelas com tempo de ocorrência entre GVT e o LVT do processo.

- **Transição CMB → Time Warp**

Como as estruturas IQ e OQ não fazem parte dos processos lógicos no modo CMB, o processo conversor deverá habilitar essas estruturas logo após a troca para o protocolo Time Warp. A partir do momento em que essas estruturas estejam habilitadas, as estruturas IQ e OQ passam a receber as mensagens contidas nas estruturas IB e OB do processo Time Warp. Deve-se lembrar que, como foi mencionado anteriormente, neste momento as mensagens das estruturas IB[i] e OB[j] do protocolo CMB já foram transmitidas para as estruturas IB e OB do protocolo Time Warp (Figura 4.7).

- **Transição Time Warp → CMB**

Na troca de protocolos Time Warp para CMB as mensagens, após a troca, são retiradas das estruturas IB[i] e OB[j] e escalonadas diretamente na lista de eventos futuros sem a necessidade de serem armazenadas temporariamente, sendo assim, as estruturas IQ e OQ são desativadas pelo processo conversor.

Como no instante da conversão optou-se por executar *rollbacks* nos processos até que os LVT's igualem-se ao valor do GVT, neste instante as estruturas IQ e OQ estarão vazias.

S - Controle de Variáveis da Simulação

A estrutura de dados *S* (conjunto de registros) mantém armazenados os valores das variáveis locais e os dados estatísticos que no final da

simulação serão analisados. Tempo médio de utilização de um recurso, tamanho médio das filas, quantidade de tempo simulado e número de preempções são exemplos de informações armazenadas nesta estrutura.

Para que haja a troca de protocolos não é necessário, uma vez que a troca é realizada em um estado consistente e esta estrutura faz parte dos dois protocolos considerados nesta tese (Figura 4.2 e 4.3), nenhum tipo de manipulação na estrutura S.

SS – Estrutura de Controle de Contexto do Time Warp

Para que a realização do mecanismo de *rollback* seja possível a estrutura SS armazena os valores que descrevem o contexto da simulação Time Warp durante o período de tempo considerado instável (eventos ocorridos entre o GVT e o LVT). Esses dados incluem *clock* do processo, elementos da lista de eventos futuros e a estrutura que armazena as estatísticas da simulação. O sistema de coleta de lixo do Time Warp é responsável pela eliminação das instâncias mais antigas e já consideradas seguras. Essas instâncias seguras são definidas como aquelas que possuem *timestamp* menor que o GVT (*clock* global).

- **Transição CMB → Time Warp**

Como o protocolo CMB não possui um mecanismo de reexecução em caso de infração da ordem cronológica, não existe a necessidade de

manutenção da estrutura SS. O processo conversor deverá ativar esta estrutura no instante da mudança de protocolos e inicializá-la com os valores de LVT, EVL e S.

Uma vez que a conversão será realizada em um estado consistente, os valores de LVT, EVL e S, presentes em cada processo lógico CMB, poderão ser utilizados pelo protocolo Time Warp.

- **Transição Time Warp → CMB**

Quando a troca de protocolos for no sentido oposto a estrutura SS, típica do protocolo Time Warp, poderá ser desativada uma vez que no protocolo CMB não poderá ocorrer uma violação da ordem cronológica de execução das mensagens.

4.3.2.2 Canais de Comunicação

O protocolo de sincronização CMB clássico, proposto por Chandy e Misra, não prevê um esquema de suporte para alocação dinâmica de canais de comunicação. Estudos posteriores mostram que a conectividade dinâmica também é possível para os protocolos conservativos (embora apresentando restrições) sendo suficientes para suportar uma plataforma paralela [Jha 1994].

Além disto, o protocolo CMB não prevê a necessidade de incluir nas mensagens os identificadores de processo fonte e processo destino. A identificação dos processos é obtida através do canal pelo qual a mensagem

está sendo recebida ou enviada. Este tipo de identificação, permitida pela topologia estática, não está incluída no protocolo Time Warp (topologia dinâmica). No Time Warp a identificação dos processos é necessária para que o mecanismo de antimensagens seja efetuado.

Para solucionar esta incompatibilidade o mecanismo de troca de protocolos de sincronização utiliza mensagens que permitem a inclusão do identificador de processos e desta forma os dois protocolos poderão gerenciar o mecanismo de alocação de canais seguindo seus princípios originais.

- **Transição CMB → Time Warp**

Como a topologia da rede de comunicações que conecta os processos lógicos de uma simulação sincronizada pelo protocolo CMB deve ser estática é necessário que os canais de comunicação já estejam alocados de acordo com o modelo de aplicação que foi implementado.

No protocolo CMB todo processo lógico possui um conjunto de canais de comunicação pelos quais as mensagens são recebidas e armazenadas em suas portas de entrada (IB[i]) e cada porta de saída (OB[j]) possui um canal de comunicação que levará a mensagem até os outro processo. No momento da troca CMB → Time Warp deverá ser mantido apenas o canal de comunicação a quem estão conectadas as portas IB[1] e OB[1] e que realizam as funções das estruturas IB e OB no protocolo Time Warp. Sendo assim, a partir da troca de protocolos, cada processo lógico (agora processos Time Warp) irá se comunicar com os outros processos da

simulação pelos canais que estão conectados com às portas IB[1] e OB[1] (Figura 4.8).

- **Transição Time Warp → CMB**

Quando o protocolo que será substituído é o Time Warp o processo conversor de cada processo lógico do mecanismo de troca deverá habilitar os canais de comunicação que realizam as conexões com as portas de entrada e saída que estavam desabilitadas durante a sincronização Time Warp.

É importante notar que a topologia da rede de comunicação que será utilizada deverá ser prevista para a realização da simulação durante a fase CMB.

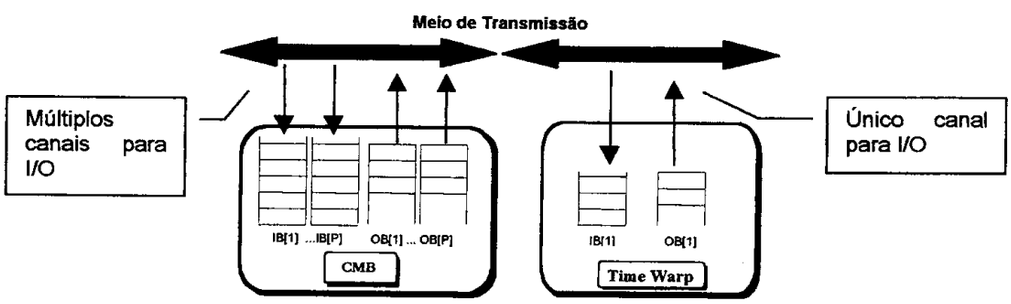


Figura 4.8 Troca de Protocolos Considerando os Canais de Comunicação. Representação dos canais de comunicação dos processos Time Warp e CMB.

4.3.2.3 Variáveis de Tempo

Os protocolos de sincronização utilizam variáveis de programação para armazenar os valores de tempo que controlam as execuções dos eventos da simulação em sua ordem cronológica correta e para representar o avanço do tempo real do modelo simulado. As variáveis utilizadas pelos protocolos CMB e Time Warp estão representadas na figura 4.9.

Estruturas de Tempo Clock Channel (CC[i])

A estrutura de dados IB[i] do protocolo CMB mantém as mensagens que foram recebidas ordenadas pelo seu tempo de ocorrência na simulação. Desta forma, a mensagem que ocupa a primeira posição da fila é aquela que possui o menor valor do tempo de ocorrência recebida por aquele canal de comunicação. Este valor é armazenado em uma das estruturas CC[i].

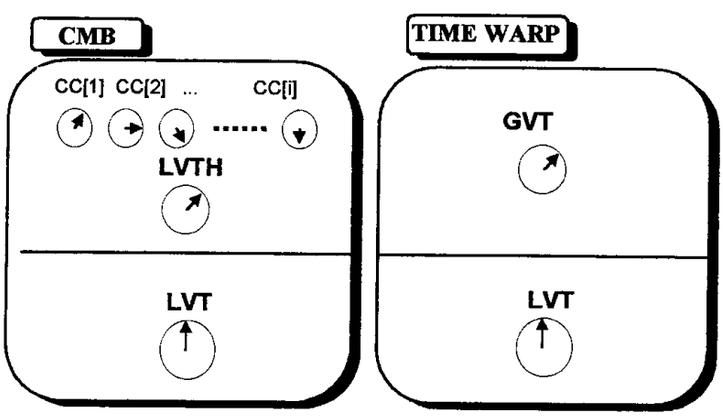


Figura 4.9 Variáveis de Tempo Utilizadas nos Protocolos CMB e Time Warp. Existe um conjunto diferente de variáveis para cada protocolo de sincronização.

Para que o controle local do processo possa decidir qual mensagem deverá ser escalonada (inserida na lista de eventos futuros) é escolhida a mensagem com menor valor entre as estruturas CC[i].

As estruturas CC[i] não são utilizadas pelo protocolo Time Warp. Isso ocorre porque no Time Warp as mensagens recebidas por um processo lógico são inseridas em apenas uma estrutura (IQ), assim sendo, basta ao controle local do processo Time Warp escalonar a mensagem que ocupa a primeira posição da estrutura IQ.

Desta forma as estruturas CC[i] não são necessárias no modo Time Warp.

- **Transição CMB → Time Warp**

Como no protocolo Time Warp o conjunto de estruturas IB[i] foi reduzido à IB[1] bastará ao mecanismo de controle local escalonar a mensagem que ocupa a cabeça da estrutura IB[1]. Desta forma, como as estruturas CC[i] só possuem significado no protocolo CMB, para o cálculo de LVTH, estas estruturas devem ser desativadas.

- **Transição Time Warp → CMB**

Quando a troca de protocolos for no sentido de Time Warp para CMB as mensagens serão armazenadas nas estruturas IB[i] e assim as estruturas de tempo CC[i] passam a armazenar novamente os valores do menor tempo de ocorrência de cada canal de chegada. Isto é, a variável CC[i] será

inicializada, pelo próprio mecanismo de controle local do CMB, com o menor tempo de ocorrência entre todos os eventos armazenados no buffer IB[i].

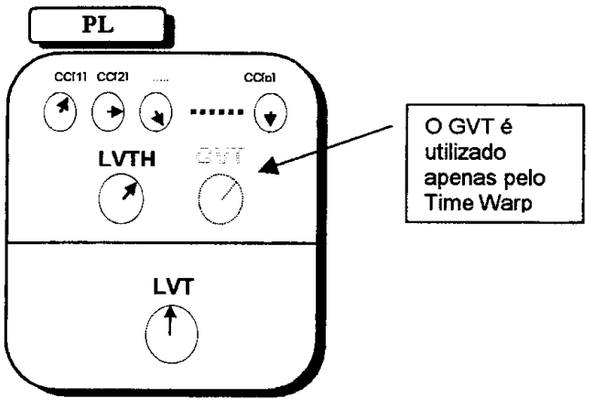


Figura 4.10 Mecanismo de Troca: Variáveis de Tempo Utilizadas no Modo CMB. O processo lógico adaptado para o mecanismo de troca recebe os valores necessários para a sincronização nos dois protocolos.

Variável de Tempo *Local Virtual Time Horizon* (LVTH)

A variável LVTH, utilizada exclusivamente pelo protocolo CMB, armazena o *menor* valor entre as mensagens que estão armazenadas nas portas de entrada (CC[i]) para cada canal de comunicação mencionada no item anterior. O evento que possui esta marca de tempo deverá ser escalonado na lista de eventos futuros. Da mesma forma que a estrutura CC[i], a variável LVTH só tem significado no protocolo CMB.

- **Transição CMB → Time Warp**

No protocolo Time Warp este valor não tem relevância, uma vez que todas as mensagens são armazenadas em uma única fila (IQ). Sendo assim o processo conversor desabilitará esta variável.

- **Transição Time Warp → CMB**

Com a chegada das mensagens pelas estruturas IB[i], as estruturas CC[i] começam a armazenar os valores dos menores tempos em cada IB[i]. Sendo assim torna-se necessário a habilitação da variável LVTH, que será inicializada com o menor valor de tempo de ocorrência entre os valores de CC[i].

LVT (Local Virtual Time)

A variável LVT, tanto para o protocolo CMB como para o protocolo Time Warp corresponde a marca de tempo do último evento processado naquele processo lógico. Este valor, denominado *clock do processo*, possui uma grande importância pois representa a quantidade de tempo virtual que o processo lógico simulou o processo físico correspondente. Além disto, é através do valor de LVT que se determina quando uma mensagem está fora da sua ordem cronológica.

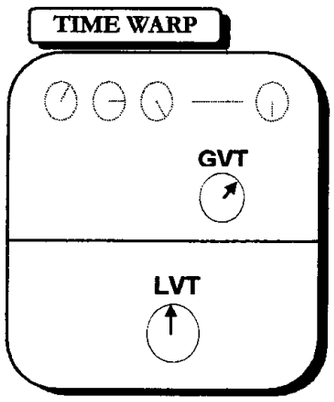


Figura 4.11 Mecanismo de Troca: Variáveis de Tempo Utilizadas no Modo Time Warp.

A forma de calcular os LVT's nos dois protocolos considerados é diferente, mas seu significado e seu valor é exatamente o mesmo. Desta forma esta variável será mantida após o momento da troca de protocolos não importando qual protocolo esteja sincronizando a simulação no instante anterior a troca.

Variável de Tempo Global Virtual Time (GVT)

Pode-se afirmar que um sistema de simulação simulou o comportamento de um sistema real até o valor de tempo armazenado em GVT. Este valor é obtido através do cálculo do menor LVT entre todos os processos do sistema de simulação.

No protocolo Time Warp a coleta de lixo (eliminação dos estados da simulação armazenados para a necessidade de efetuar um *rollback*) é realizada através do valor do GVT disponível no processo lógico, ou seja, qualquer alteração efetuada no estado da simulação antes do valor de GVT é considerada estável.

O protocolo CMB, ao contrário, não prevê a existência deste valor armazenado em cada processo lógico (Figura 4.10).

Sendo assim, para que os processos lógicos Time Warp tenham acesso ao valor do GVT logo após a troca de CMB para Time Warp (Figura 4.11), o processo gerenciador deverá calcular, através dos valores dos LVT's enviados pelos processos observadores, o menor LVT e enviá-lo aos

processos conversores. No sentido oposto, ou seja, a troca do Time Warp pelo CMB, as variáveis GVT são desativadas.

4.4 Algoritmo dos Processos do Mecanismo de Troca de Protocolos

Nas seções anteriores foi discutido o mecanismo de troca de protocolos através das manipulações das estruturas de dados, das variáveis de tempo e da topologia de comunicação realizadas durante a troca de protocolos. Haja vista as manipulações necessárias para a realização da troca de protocolos, a seguir são apresentados os algoritmos dos processos envolvidos no mecanismo de troca.

4.4.1 Algoritmo do Processo Observador

O processo observador deve estar sendo executado concorrentemente com cada processo lógico e a cada intervalo de tempo deverá enviar informações a respeito do desempenho do processo lógico ao processo gerenciador. A seguir é descrito o algoritmo do processo observador:

1. *Início*
2. *Enquanto não terminar a simulação:*
   3. *Atualizar variáveis de desempenho da simulação (% mensagens nulas, probabilidade de bloqueio, LVT, etc.);*

4. Transmitir ao Processo Gerenciador;

5. Voltar ao passo 2;

6. Fim.

4.4.2 Algoritmo do Processo Gerenciador

O processo gerenciador recebe as informações transmitidas pelos processos observadores e decide se a troca de protocolos deve ser realizada.

1. Início

2. Enquanto a simulação não terminar:

3. Agrupar e analisar as informações recebidas;

{Com base nas análises obtidas através do estudo do comportamento do protocolo em execução e decidir se existe a necessidade da troca de protocolos}

4. Se não existir a necessidade de realização da troca, ir para o passo 2;

5. Efetuar a troca de protocolos:

A) Troca CMB para Time Warp

- *Calcular o valor do GVT através dos LVT's recebidos;*
- *Transmitir o identificador de troca CMB para Time Warp aos processos Conversores junto com o valor do GVT;*
- *Ativar processos Conversores;*

B) Troca Time Warp para CMB

- *Transmitir identificador de troca Time Warp para CMB aos processos Conversores;*
- *Ativar processos Conversores;*

6. Voltar ao passo 2;

7. Fim.

4.4.3 Algoritmo do Processo Conversor

Os processos conversores, após terem sido ativados pelo processo gerenciador, efetuam as manipulações nas estruturas de dados, variáveis de tempo e de comunicação dos protocolos envolvidos. O algoritmo realizado por estes processos é descrito a seguir:

1. Início

2. Se a troca for CMB para Time Warp, ir para passo 3 senão ir para passo 10;

{ Troca CMB para Time Warp }

3. Desativar portas de entrada e saída (IB[i-1] e OB[i-1]) e variáveis CC[i];

4. Desativar variável de tempo LVTH;

5. Ativar as estruturas IQ e OQ;

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

6. Ativar estrutura SS e inicializá-la com os valores de LVT, EVL e S;

7. Percorrer a EVL e atualizar os campos Processo Origem, Processo Destino, Sinal de Identificação;

8. Atualizar GVT

9. Ir para o passo 16;

{ Troca CMB para Time Warp }

10. Promover a consistência do processo simulação através do envio de uma anti-mensagem com o valor do GVT do processo lógico;

11. Ativar portas de entrada e saída (IB[i-1] e OB[i-1]) e atualizar variáveis CC[i];

12. Atualizar variável de tempo LVTH;

13. Desativar as estruturas IQ e OQ;

14. Desativar estrutura SS ;

15. Percorrer a EVL e atualizar os campos Processo Origem, Processo Destino, Sinal de Identificação;

16. Transmitir finalização do processo de troca do protocolo;

17. Fim.

4.5 Conclusões

Uma grande quantidade de pesquisas têm sido realizadas para avaliar o desempenho dos protocolos conservativos e otimistas (e suas variantes) em diferentes tipos de aplicações. As características que determinam qual protocolo deve ser utilizado, além de serem difíceis de serem caracterizadas, alteram-se dinamicamente [Jha 1994].

Protocolos que apresentam características unificadas dos protocolos de sincronização conservativos e otimistas têm sido propostos, entretanto, a quantidade de parâmetros a serem ajustados previamente pelo modelador torna a utilização destes protocolos pouco viável para um modelador não familiarizado.

Foi apresentado neste capítulo um mecanismo que permite, durante a execução de uma simulação distribuída, a troca automática e dinâmica entre os protocolos CMB e Time Warp para realizar o sincronismo dos eventos da simulação. Desta forma, utilizando-se de dois protocolos bastante conhecidos, o sistema de simulação proposto poderá decidir, através do comportamento da simulação, qual deles se adapta melhor ao contexto em que a simulação distribuída está sendo realizada.

A troca dinâmica de protocolos de sincronização envolve duas questões principais: como realizar a troca considerando as características distintas de cada protocolo e quando realizar a troca, isto é, determinar o melhor momento para a troca.

A maneira pela qual o momento da troca de protocolos será detectado será discutido no capítulo 5. No capítulo 4 foi apresentada a forma com que

as diferentes estruturas de dados, variáveis de tempo e tipos de topologias utilizadas pelos protocolos envolvidos são manipuladas durante a realização da troca.

Antes que a troca de protocolos de sincronização seja iniciada é necessário que a simulação esteja em um estado consistente. Esta garantia é necessária porque a realização da troca de protocolos em um estado transitório poderia causar um erro de causalidade na execução das mensagens.

No protocolo de sincronização CMB o mecanismo de mensagens nulas garante um estado de consistência permanente. Entretanto, no protocolo Time Warp é necessário restabelecer os estados de todos os processos no momento marcado pela variável GVT, que por sua própria definição garante um estado consistente.

Três processos lógicos especiais são utilizados para manipular o mecanismo de troca. O processo gerenciador, com base nas informações sobre o desempenho da simulação, monitorada pelos processos observadores, determina qual o momento em que a troca de protocolos deverá ser realizada.

Após o momento da troca ser detectado, os processos conversores, localizados dentro de cada processo lógico da simulação, iniciam as operações de chaveamento das estruturas e variáveis de cada processo.

Os processos lógicos utilizados pelo mecanismo de troca foram definidos através da união entre as estruturas de dados utilizadas pelos dois protocolos (Figura 4.12).

Desta forma, todas as estruturas de dados utilizadas pelos processos lógicos nos protocolos CMB e Time Warp estão presentes nos processos lógicos do mecanismo de troca. Dependendo de qual protocolo de sincronização será utilizado, após o momento da troca, apenas um subconjunto dessas estruturas é utilizado.

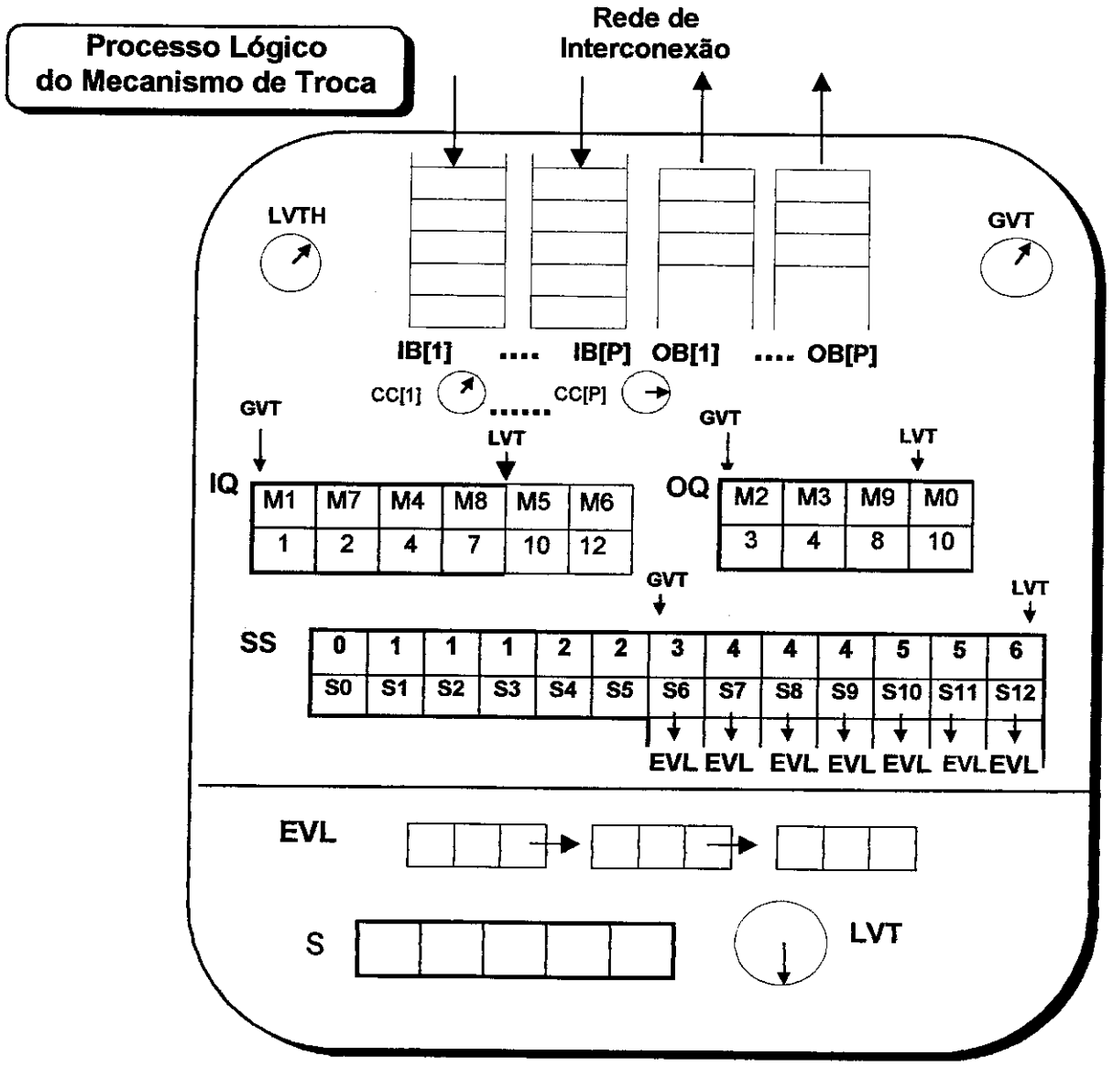


Figura 4.12 Processo Lógico Utilizado pelo Mecanismo de Troca. Cada processo lógico é formado pelas estruturas de dados e variáveis de tempo utilizadas pelos dois protocolos de sincronização.

Com relação à topologia da rede de comunicação utilizada o protocolo de sincronização CMB, ao contrário do protocolo Time Warp, não prevê um esquema de suporte para alocação dinâmica. Por esse motivo o protocolo CMB não considera a necessidade de incluir nas mensagens os identificadores de processo fonte e processo destino. A identificação dos processos é obtida através do canal pelo qual a mensagem está sendo recebida ou enviada. No Time Warp a identificação dos processos é necessária para que o mecanismo de antimensagens seja eficiente.

Para solucionar esta incompatibilidade o mecanismo de troca de protocolos de sincronização propõe uma estrutura de mensagens que permite a inclusão do identificador de processos e desta forma os dois protocolos poderão gerenciar o mecanismo de alocação de canais seguindo seus princípios originais.

Da mesma maneira com que foram consideradas as estruturas de dados, as variáveis de tempo que compõem os dois protocolos de sincronização também foram consideradas para atuarem em um único processo lógico. Foi discutido que o conjunto de variáveis de tempo utilizado por um protocolo de sincronização não é o mesmo utilizado pelo outro protocolo. O GVT, por exemplo, é utilizado no protocolo Time Warp para a realização da coleta de lixo (eliminação dos estados da simulação armazenados para a necessidade de efetuar um *rollback*) uma vez que qualquer alteração efetuada no estado da simulação antes do valor de GVT é considerada estável. O protocolo CMB, ao contrário, não prevê a existência deste valor armazenado em cada processo lógico. Sendo assim foi definido um mecanismo que, através dos valores dos LVT's fornecidos

pelos processos observadores, permite o processo gerenciador calcular o valor do GVT e transmiti-los aos processos conversores durante a troca do CMB para o Time Warp.

No capítulo 5 será abordada a maneira pela qual o momento da troca de protocolos é detectado.

Capítulo 5

Modelo e Implementação de um Sistema de Simulação para o Estudo do Comportamento do Protocolo CMB

Este capítulo aborda os conceitos que envolvem a implementação do sistema de simulação utilizado para obter as características do comportamento do protocolo CMB. As características dos modelos das plataformas representativas, do protocolo CMB e da aplicação são apresentadas.

5.1 Introdução

Definir qual o melhor protocolo de sincronização para um determinado modelo não é uma tarefa simples. Além das características iniciais do modelo deve-se considerar que o comportamento do sistema simulado pode ser modificado dinamicamente durante sua execução. A proposta de ajustar o protocolo de sincronização de acordo com o comportamento da simulação é bastante atraente.

Entretanto o modo com que a troca de protocolos é realizada não é a única questão a ser estudada. Deve-se considerar também qual o melhor momento para a realização da troca de protocolos para atingir um maior desempenho da simulação. Além do protocolo de sincronização, existem vários fatores que influenciam no comportamento de uma simulação distribuída. Em Ulson [Ulson 1999] o comportamento do protocolo CMB foi observado através de técnicas de aferição, manipulando-se características como granulosidade, particionamento dos processos, alocação, características do modelo e sobrecarga de mensagens nulas.

Além da análise do comportamento do protocolo CMB em relação a estas características, a possibilidade de analisar o comportamento do CMB em um conjunto de plataformas com características distintas torna a técnica de modelagem uma ferramenta bastante atraente. A não disponibilidade de algumas plataformas pode ser resolvida através da modelagem e da alta flexibilidade característica das técnicas de simulação.

Assim, para realizar o estudo do comportamento do protocolo CMB e detectar em quais características este protocolo influencia negativamente o

desempenho da simulação, foi implementado um modelo de simulação do protocolo de sincronização CMB, utilizando a extensão funcional SMPL [MacDougall 1987], onde são consideradas as características do hardware utilizado para a execução e da aplicação utilizada como estudo de caso para a simulação. Desta forma, o objetivo da simulação desenvolvida é definir o momento em que o protocolo CMB deixa de ser interessante e a troca para o protocolo Time Warp deve ser considerada. A seção 5.2 apresenta as características da extensão SMPL utilizada na implementação da simulação. A seção 5.3 apresenta o modelo e o programa de simulação desenvolvido.

5.2 Simulation Program Language (SMPL)

SMPL é uma extensão funcional baseada em uma linguagem de programação hospedeira de propósito geral (nesse caso a linguagem C) proposta por MacDougall [MacDougall 1987]. Essa extensão apresenta uma biblioteca (subsistema *smpl*) na qual, em conjunto com a linguagem hospedeira, compõem uma linguagem de simulação orientada a eventos. A implementação dos modelos de simulação em SMPL é semelhante aos programas implementados na linguagem hospedeira: as operações de simulação são executadas via chamadas a procedimentos.

O subsistema de simulação *smpl* é apenas uma parte do ambiente chamado SMPL. O SMPL ainda inclui ferramentas adicionais como: depurador, coleta de dados, funções que analisam os resultados da simulação e uma interface interativa para a execução de um modelo de

simulação, denominada *mtr*. Com o auxílio da *mtr* o usuário pode, em tempo de execução, inspecionar a execução da simulação na tela do computador e também alterar os parâmetros de entrada da simulação[MacDougall 1987].

5.2.1 Estruturas do SMPL

Existem três tipos de entidades presentes no subsistema *smpl:* as *facilities* (recursos), os *tokens* (tarefas) e os *eventos*.

Estaticamente um sistema compreende uma coleção de *facilities* interconectadas. Uma *facility* representa, no modelamento, um recurso que oferece determinado serviço no sistema real, tal como uma CPU em um sistema de computação ou um meio de transmissão em um modelo de rede local. O SMPL fornece funções para criar, reservar, liberar, causar preempção e para observar o estado das *facilities*.

Os *tokens* representam as entidades ativas do sistema. O comportamento dinâmico do sistema é modelado pelo movimento dos *tokens* através do conjunto de *facilities*. Um *token* pode representar uma tarefa em um modelo de sistema de computação ou um pacote na comunicação de uma rede local [MacDougall 1987].

Existem dois tipos básicos de operação para controlar o fluxo dos *tokens* através do sistema de simulação: um *token* pode reservar uma *facility* ou pode ser escalonado para uma determinada atividade durante um certo período de tempo. Se no instante em que o *token* tenta reservar uma *facility,*

esta estiver sendo ocupada por outro *token*, o *token* requisitante ficará suspenso em uma fila de espera até que a *facility* seja liberada.

Quanto às estruturas de dados do SMPL, uma certa compactação foi obtida através do uso de um único elemento básico de dados para representar todas as estruturas de dados do SMPL. Cada elemento possui cinco campos nomeados conforme ilustra a figura 5.1. O SMPL mantém um *pool* desses elementos. Os elementos são alocados para servirem de descritores para as *facilities*, entradas para as filas e entradas para a lista de eventos futuros. Os valores armazenados nos campos dependem da razão pela qual o elemento básico é alocado. Por exemplo, se o elemento estiver sendo alocado para constituir um descritor de uma *facility* então ele deverá armazenar em seus campos informações como: número de servidores ocupados, tamanho da fila, tempo da última mudança na fila, número de preempções, e outras informações relativas ao funcionamento da *facility*.

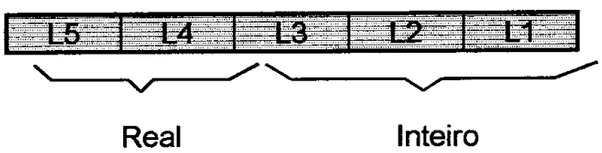


Figura 5.1 Elemento básico de dados. Unidade de dados utilizadas pela extensão SMPL para a manipulação das informações utilizadas na simulação.

Uma mudança de estado de qualquer entidade do sistema, ativa ou passiva, é provocada por eventos. Suponha que uma tarefa após completar a sua execução na CPU seja, logo após, liberada. Pode-se considerar que

dois eventos teriam ocorrido: o primeiro evento seria o fim da atividade de processamento e o segundo a mudança de estado da CPU (desocupada).

- **Primitivas do SMPL[1]**

Um conjunto de primitivas, disponíveis na extensão SMPL, auxiliam na implementação de simulações orientadas a eventos. Dentre elas destacam-se:

Smpl()

A primitiva smpl() tem por objetivo iniciar todo o subsistema de simulação para que a simulação seja executada. Ela atribui valores iniciais às estruturas de dados e ao tempo de simulação.

Facility()

Essa primitiva cria, nomeia e retorna um descritor para cada *facility*. O descritor é usado para especificar a *facility* em outras operações. Cada *facility* possui uma fila e um número n de servidores paralelos.

Request()

Usada para requisitar o atendimento de um dos servidores de uma *facility*. Se existir um servidor desocupado, esse servidor é reservado para o

---

[1]Todas as funções (código em C) pertencentes ao SMPL podem ser encontradas em [MacDougall 1987].

token, caso contrário a primitiva *request()* retorna um valor indicando que a *facility* está ocupada.

Release()

Libera um servidor de uma determinada *facility*. O servidor após ser liberado terá sua estatística atualizada e o número de servidores ocupados daquela *facility* é então decrementado.

Schedule()

Um evento é escalonado via a chamada da rotina *schedule*. Essa primitiva possui três parâmetros de entrada: o número do evento *(ev)*, o tempo de ocorrência *(te)* e o número do *token* que será escalonado *(tkn)*. Com esses parâmetros a primitiva *schedule* cria uma entrada na lista de eventos futuros.

Cause()

A rotina *cause* remove o elemento da cabeça da lista de eventos futuros e avança o tempo da simulação igualando-o ao tempo de ocorrência do elemento removido. Corresponde a realização do evento por parte do *token*.

Além das primitivas descritas nesta seção, o SMPL possui outras primitivas que manipulam operações com filas ou listas (*enqueue(), getelm(),* etc.) e primitivas que fornecem medidas e distribuições estatísticas.

Conforme pode ser observado na descrição do SMPL, apresentada nesta seção, esta extensão funcional possui todos os requisitos necessários para avaliar o protocolo de sincronização CMB. Além disso, esta ferramenta

é largamente utilizada no grupo de Sistemas Distribuídos e Programação Concorrente do ICMC – USP, tendo-se um completo domínio de sua utilização. A utilização do SMPL para simulação de diversos sistemas, permitiu ao grupo comprovar sua flexibilidade, eficiência e bom funcionamento. Por estas razões, SMPL foi escolhido para o desenvolvimento do programa de simulação do protocolo CMB. A próxima seção descreve o modelo e a simulação do protocolo CMB.

5.3 Modelo da Simulação Utilizada para Análise do Comportamento do Protocolo CMB

O desenvolvimento do programa de simulação utilizado para analisar o protocolo CMB foi dividido nas seguintes etapas:

- Construção do modelo do *protocolo* CMB, levando em consideração as premissas originais propostas na definição do mesmo. Diversas características do protocolo CMB (variáveis, estruturas de dados e mecanismo de envio de mensagens nulas) foram abstraídas e inseridas no modelo;

- Em seguida, as informações relativas à *plataforma* de processamento foram inseridas no contexto da simulação através da modelagem das suas características, como velocidade de processamento, topologia da rede e tempo de comunicação;

- A etapa final consistiu na construção dos modelos de aplicação que foram utilizados como estudos de caso na análise do comportamento da simulação distribuída.

A seção 5.3.1 descreve a estrutura global da simulação desenvolvida. Em seguida, são detalhadas as características consideradas no modelamento e simulação do protocolo CMB (seção 5.3.2), das plataformas (5.3.3) e das aplicações (5.3.4).

5.3.1 Estrutura Global do Modelo de Simulação

Uma simulação orientada a eventos consiste em um conjunto de eventos que devem ser executados seguindo uma ordem cronológica. Esses eventos são abstraídos pelo modelador baseando-se nas características físicas do sistema real simulado.

No modelo geral construído para analisar o protocolo CMB além dos eventos relacionados ao sistema real (aplicação) foram incluídos os eventos que representam a plataforma e aqueles referentes ao protocolo CMB.

Todos esses eventos, apesar de se relacionarem a modelos diferentes (aplicação, plataforma e protocolo) coexistem em um mesmo espaço de código referente ao corpo principal da simulação cuja delimitação ocorre através de uma estrutura de repetição que controla a simulação.

A extensão SMPL apresenta, na sua configuração original, algumas limitações quanto ao tamanho das estruturas de dados disponíveis para o gerenciamento da simulação. Desta forma, foram necessárias algumas modificações no código original da extensão SMPL para que o modelo final fosse executado com uma quantidade de eventos e tarefas suficientemente capaz de fornecer resultados estatísticos representativos.

Entre as modificações realizadas na extensão SMPL a principal refere-se ao tamanho do *pool* de elementos. Esta estrutura é utilizada pelo SMPL para armazenar as informações relativas aos recursos (*facility*) utilizados no modelo, as medidas estatísticas que serão fornecidas pelo relatório final e, principalmente, para o armazenamento das tarefas e das mensagens (completas e nulas) que cada processo lógico recebe durante a simulação.

Desta forma, devido a quantidade de informações necessárias, o valor inicial do tamanho do *pool* foi alterado para seu valor máximo limitado pelo compilador da linguagem C e pela quantidade de memória disponível na plataforma hospedeira.

Além desta modificação houve a necessidade de alterar o tipo inicial de algumas variáveis da extensão SMPL cuja capacidade de armazenamento de determinadas medidas (principalmente estatísticos) estava sendo extrapolada (*overflow*). Esta tarefa consiste em uma atividade mais complexa pois além de se tratar de um código não-proprietário a extensão SMPL apresenta uma interatividade muito forte entre suas primitivas de simulação.

Após terem sido realizadas essas alterações é que foi possível a inclusão dos três modelos em um único programa de simulação SMPL e

assim, através de uma estratégia de execução, analisar o comportamento do protocolo CMB sob diferentes aspectos. A figura 5.2 apresenta a estrutura geral do processo de análise do comportamento do protocolo CMB.

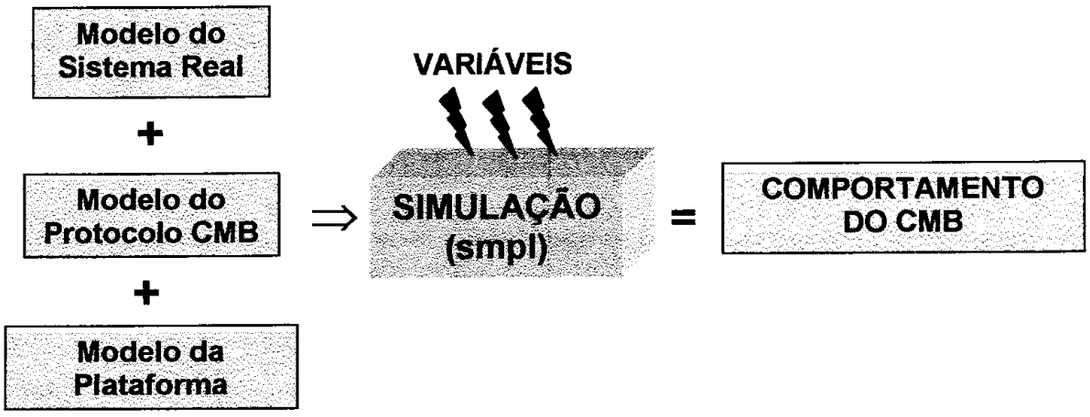


Figura 5.2 Composição do Modelo Geral para Análise do CMB. Após o processo de modelagem do sistema real, do protocolo CMB e da plataforma a simulação é executada com diferentes configurações.

Para o desenvolvimento da simulação, todas as características do protocolo CMB, tais como diferentes tipos de mensagens utilizadas, mecanismo de mensagens nulas e variáveis simuladoras de tempo pertencentes à proposta original dos autores [Misra 1986] foram consideradas.

A seção 5.3.2 apresenta a simulação do protocolo CMB através da descrição das primitivas desenvolvidas.

5.3.2 Modelagem e Simulação das Características do Protocolo CMB

De acordo com a proposta do protocolo CMB pode-se identificar 5 situações durante a execução da simulação distribuída e que foram consideradas no processo de construção do modelo do protocolo CMB:

Situação 1. Um processo, que não estava bloqueado, atinge um estado de ausência de processamento (bloqueado).

Neste caso, segundo a implementação original do protocolo CMB, o processo lógico envia uma mensagem nula a todos os processos que fazem parte da simulação distribuída. Esta mensagem nula contém o valor do tempo de ocorrência da próxima mensagem que o processo bloqueado deverá processar. Ou seja, o processo anuncia aos demais processos da simulação o período de tempo em que não existirá eventos a serem retirados da sua lista de ventos futuros.

Esta abordagem desconsidera o envio de mensagens nulas por *demanda.* Quando o mecanismo de mensagens nulas trabalha sob o regime de demanda existe um envio de mensagens nulas apenas quando houver uma solicitação por parte dos processos ou quando o valor do tempo de ocorrência da próxima mensagem for superior a um valor predeterminado.

Situação 2. Um processo bloqueado recebe uma mensagem completa

Nesta situação o processo lógico não está executando nenhum evento e recebe uma mensagem completa. Desta forma a mensagem completa recebida pelo processo lógico é apenas armazenada na sua lista de eventos futuros e uma nova mensagem nula, caso haja a necessidade, com o mesmo conteúdo da mensagem emitida anteriormente é enviada para os processos da simulação.

Situação 3. Processo não está bloqueado e recebe uma mensagem completa

Nesta situação a mensagem completa recebida pelo processo lógico é apenas armazenada na sua lista de eventos futuros.

Situação 4. Processo não está bloqueado e recebe uma mensagem nula

Quando um processo, em qualquer circunstâncias (bloqueado ou não), recebe uma mensagem nula, esta mensagem é inserida em uma das portas de entrada do processo (IB[i]) e posteriormente deve "morrer" com o processo, isto é, ela não é repassada para outros processos. Na implementação desta simulação foi criado um evento na simulação exclusivamente para excluir essas mensagens do sistema.

A marca de tempo trazida por uma mensagem nula atualizar o LVT do processo mesmo não existindo a necessidade uma vez que o processo não está bloqueado.

Situação 5. Processo está bloqueado e recebe uma mensagem nula

Quando um processo bloqueado recebe uma mensagem nula seu LVT é atualizado, após a mensagem Ter sido processada, com o valor do tempo de ocorrência contido na mensagem nula. Esta atualização do valor da variável LVT pode causar um avanço no *clock* local do processo descaracterizando o processo como um processo bloqueado.

Além disto, a chegada de uma mensagem nula a um processo lógico proporciona a eliminação de todas as mensagens nulas armazenadas na estrutura de armazenamento de mensagens recebidas (porta).

Todas as situações descritas anteriormente foram implementadas no sistema de simulação para o estudo do comportamento do protocolo CMB.

Uma vez tendo visto essas situações pode-se descrever o procedimento que o protocolo CMB realiza toda vez que um processo lógico recebe uma mensagem:

1. O princípio conservativo é verificado para descobrir se a mensagem está defasada;

2. A mensagem é processada de acordo com uma das 5 situações relatadas anteriormente;

3. Determina-se se a mensagem, caso seja completa, irá ou não bloquear o processo. Esta etapa consiste em atribuir ou não (probabilisticamente) um estado de ausência de processamento após a execução do evento relativo a mensagem recebida;

4. O relógio do processo (LVT) é atualizado;

5. O valor do clock global (GVT) é atualizado.

A seguir são detalhadas as primitivas implementadas neste trabalho para realizar as tarefas do protocolo CMB e que são importantes para a avaliação deste protocolo.

- ***Primitiva Bloqueia_ou_não()***

Para que o modelo construído apresentasse um comportamento semelhante ao modelo original CMB foi desenvolvida uma rotina chamada *bloqueia_ ou_não_processo()*. A função desta rotina é, além de permitir a representação de modelos com diferentes características de bloqueio, causar o mesmo efeito de quando um processo lógico permanece um determinado tempo sem possuir eventos a serem processados. Desta forma, o processo não emitirá nenhuma mensagem nula ou completa por um

período de tempo. Esse comportamento, no qual o processo fica incomunicável durante certo período de tempo, é comum nas simulações distribuídas, o que é alterado é a quantidade de tempo em que o processo permanece neste estado.

Esta rotina funciona como uma variável semáforo. De acordo com a ocorrência de uma probabilidade o processo entra ou não em estado de bloqueio. Um processo sai do estado de bloqueio quando seu relógio (LVT) é atualizado e torna-se maior ou igual ao tempo de ocorrência do seu próximo evento a ser executado. A atualização do LVT, neste caso, é realizada pela chegada de uma mensagem nula ao processo lógico.

- ***Primitiva Atualizar_gvt()***

Como foi mencionado anteriormente, o *Global Virtual Time* (GVT) representa o tempo no qual o modelo de simulação já simulou o sistema físico correspondente.

Toda vez que ocorre um avanço no valor do LVT de um determinado processo lógico o cálculo do GVT deve ser realizado e, se for necessário, atualizado. Considerando como exemplo um modelo de aplicação composto por dois processos lógicos (denominados *cpu* e *disco*) a atualização do GVT para este modelo pode ser realizada através da rotina apresentada na figura 5.3. Note que as variáveis *lvt_cpu* e *lvt_disco* representam os valores dos LVT's para os processos *cpu* e *disco,* respectivamente. Desta forma, o valor de GVT é obtido atribuindo-se a esta variável o menor valor entre os clocks locais.

```
void atualizar_gvt()
{
if (lvt_cpu < lvt_disco) then
                    {gvt = lvt_cpu;}
                else
                    {gvt = lvt_disco;}
}
```

Figura 5.3 Rotina atualizar_gvt(). O GVT é sempre atualizado com o menor valor entre os *clocks* dos processos.

- ***Implementação do Princípio Conservativo***

A principal característica dos protocolos conservativos encontra-se na verificação dos tempos de ocorrência das mensagens recebidas antes de efetivamente serem executadas. Caso a mensagem possua um tempo de ocorrência menor que o LVT do processo a simulação fica bloqueada.

Para a implementação destas condições foi adicionado em cada processo lógico um código que, executado antes da realização do evento correspondente à mensagem recebida, compara o valor do tempo de ocorrência da mensagem com o valor do clock local (LVT). No trecho de código da figura 5.4 os valores do clock do processo CPU e tempo de ocorrência da mensagem (evento) estão representadas respectivamente pelas variáveis *clock_cpu* e *l5[i]+clock*.

```
// testando mensagem defasada!
if (clock_cpu > (l5[i]+clock))
  then
      { printf("\n Mensagem Defasada!
      Contrariando Principio Conservativo.  Simulação Bloqueada!.");
      x=getchar();
      }
```

Figura 5.4 Código que garante o princípio conservativo. Com a chegada de uma mensagem defasada o processo é bloqueado.

- ***Envio/Recebimento de mensagens***

Em uma simulação sincronizada pelo protocolo CMB e com mecanismo de prevenção de *deadlock* pelo uso de mensagens nulas existe dois tipos de mensagens: as mensagens completas e as mensagens nulas.

As mensagens completas referem-se àquelas nas quais os processos lógicos da simulação trocam informações a respeito da simulação. Ou seja, nelas são armazenados o número do evento, o identificador da tarefa e o tempo de ocorrência do evento.

Por outro lado, as mensagens nulas contém apenas o valor do tempo de ocorrência do próximo evento a ser executado pelo emissor da mensagem nula e o identificador de mensagens nulas (com valor igual a –1).

A principal característica dos protocolos conservativos refere-se a verificação dos tempos de ocorrência das mensagens recebidas antes de efetivamente serem processadas. Caso a mensagem possua um tempo de ocorrência menor que o LVT do processo a simulação fica bloqueada.

Para que o mecanismo de recebimento e envio de mensagens fosse possível foram implementadas as portas de entrada (IB[i]) para o armazenamento das mensagens recebidas pelos processos lógicos. Cada porta de entrada é representada por uma estruturas *FIFO* que armazena as mensagens recebidas e depois são inseridas na lista de eventos futuros.

As estruturas OB[i] foram implementadas utilizando-se o próprio *pool* de elementos fornecido pela extensão SMPL. Neste *pool* as mensagens que deixam os processos lógicos são armazenadas automaticamente pelo mecanismo de escalonamento do SMPL e logo após são enviadas para outros processos lógicos.

5.3.3 Modelo da Plataforma

Uma das grandes vantagens da utilização da simulação para o estudo do comportamento do protocolo CMB é a possibilidade de, através da construção de um modelo, representar a execução da simulação em diferentes plataformas paralelas.

Desta forma, foi construído um modelo no qual características como o número de elementos de processamento, a velocidade de transmissão da

rede de comunicação e o desempenho dos processadores fossem utilizadas como parâmetros adicionais à simulação.

Para representar os recursos do modelo da plataforma (elemento de processamento e meio físico) foram utilizadas as *facilities* do SMPL (Figura 5.5). Sendo assim, os processadores e o meio físico representam os recursos que serão utilizados na simulação distribuída e desta forma são simulados no SMPL. Assim, quando uma mensagem é transferida de um processador *p1* a outro processador *p2* a seqüência SMPL pode ser resumida da seguinte forma:

- mensagem (*token*) em *p1* requisita *facility meio físico* (MF);
- mensagem utiliza MF durante um determinado período de tempo;
- mensagem deixa *facility* MF e requisita *facility p2*;
- mensagem utiliza *facility p2*.

Para cada *facility* que compõe o modelo da plataforma existe um conjunto de parâmetros que determinam o desempenho do recurso simulado. O meio físico, por exemplo, é especificado através dos valores que determinam o tempo de transmissão de uma mensagem e a quantidade de servidores SMPL utilizados para representá-lo. O valor do tempo de transmissão de uma mensagem modela as características de desempenho das transmissões realizadas pelo meio físico. Por outro lado, o valor do número de servidores indica a natureza do meio de comunicação com relação a sua forma de acesso. Quando o meio de comunicação é compartilhado existe a necessidade da espera pela desocupação do meio,

para que as mensagens possam ser transmitidas. Esse tipo de comportamento do meio físico foi simulado através da utilização de apenas *um* recurso SMPL com *um* servidor para modelar o meio de comunicação. Desta forma as mensagens são obrigadas a esperar pela desocupação do meio de comunicação para serem transmitidas (Ex. : Ethernet).

Da mesma forma, quando o meio físico não é compartilhado foram utilizados mais servidores SMPL na modelagem do meio de comunicação, impedindo desta forma a necessidade de espera para a transmissão (Ex.: Rede de chaveamento).

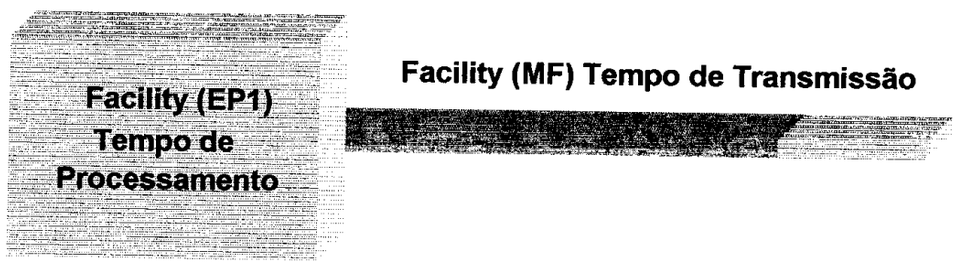


Figura 5.5 Construção do Modelo da Plataforma. Os processadores e os meios de comunicação são representados na simulação por recursos SMPL (*facilities*). Os valores de tempo de processamento e tempo de transmissão foram utilizados como parâmetros para o período de utilização de cada recurso.

5.3.3.1 Plataformas Representativas

Para realizar a simulação de uma aplicação utilizando o protocolo CMB em diferentes plataformas optou-se pela abordagem representativa de arquiteturas com características de desempenho diferentes.

A utilização de plataformas específicas para a realização deste trabalho foi desconsiderada haja vista a grande quantidade de plataformas disponíveis e o grande número de variáveis que envolve a execução de uma simulação em uma determinada plataforma (versão do compilador utilizado, natureza do *benchmark* utilizado para obtenção dos valores de latência e *bandwidth*, medidas de grandeza utilizada para quantificar o desempenho do elemento processador e a taxa de transmissão, etc.).

Desta forma, foram utilizados parâmetros fictícios para as plataformas. Apesar de não representarem uma plataforma em particular os parâmetros utilizados são característicos das principais arquiteturas disponíveis atualmente.

O objetivo da avaliação realizada neste trabalho não é avaliar o comportamento do protocolo CMB para uma plataforma específica mas sim para diversas classes de arquiteturas.

Como ponto de partida foram escolhidas as arquiteturas SP2 IBM (PVMe) e Cray T3D (PVM). Estas plataformas foram consideradas por possuírem desempenhos que podem ser considerados intermediários. A partir destas arquiteturas de referência foram consideradas plataformas que sejam mais ou menos eficientes em termos de processamento e de comunicação.

As arquiteturas consideradas apresentam valores de latência e *bandwidth* situados em uma faixa intermediária dentre as arquiteturas analisadas na literatura. A tabela 5.1 mostra os valores, obtidos através de um algoritmo *ping-pong*, das medidas de latência e bandwidth para as duas arquiteturas referenciais.

Tabela 5.1 Arquiteturas IBM e CRAY: Latência e Bandwidth.

	Latência, µs	Bandwidth, MB/s
CRAY T3D PVM	136,7	25,1
IBM SP2 PVMe	209	16,5

Alem destas arquiteturas foram criadas outras quatro cuja denominação atribuída a cada uma delas refere-se ao desempenho apresentado pelo processador e pelo meio físico. Esta denominação segue a seguinte norma:

P *<desempenho do processador> <desempenho do meio físico>*

Os desempenhos dos processadores e do meio físico são representados pelas letras **A** (**A**lto desempenho) e **B** (**B**aixo desempenho). Assim, pode-se dizer que a plataforma *PAB* representa uma plataforma cujos processadores apresentam um alto desempenho e o meio físico apresenta um baixo desempenho em relação às plataformas intermediárias (SP2 e CRAY). Assim sendo, as plataformas utilizadas para o estudo do comportamento do protocolo CMB são relacionadas na tabela 5.2.

O objetivo da criação das plataformas representativas, como foi mencionado no início desta seção, objetiva a análise do comportamento do protocolo CMB executado sob diferentes aspectos relativos aos desempenhos das plataformas. A figura 5.6 mostra graficamente a relação entre as plataformas utilizadas nesta análise.

Tabela 5.2 Arquiteturas Representativas.

Plataforma	Desempenho do Processador	Desempenho do Meio Físico
PAA	Alto	Alto
PAB	Alto	Baixo
CRAY	Médio	Médio
SP2	Médio	Médio
PBA	Baixo	Alto
PBB	Baixo	Baixo

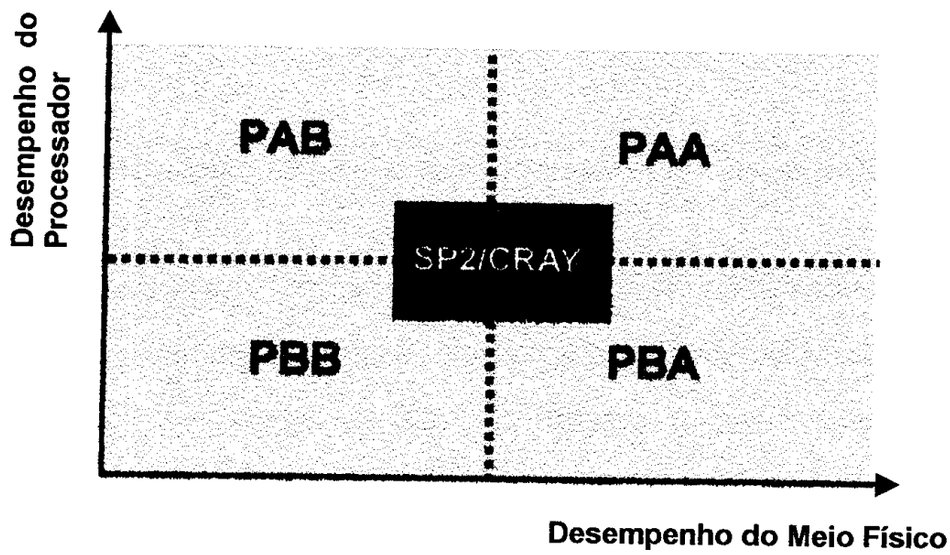


Figura 5.6 Relação entre o desempenho das Plataformas Utilizadas.

Os passos realizados para determinar os parâmetros de desempenho para as plataformas intermediárias e representativas são descritos a seguir:

- **Representação do Tempo de Transmissão (Meio Físico)**

Considerando as informações armazenadas em uma mensagem completa utilizada pelo protocolo CMB (tabela 5.3), pode-se calcular através

da equação *(1)* o tempo gasto para a transmissão desta mensagem utilizando-se dos valores da tabela 5.2.

Tabela 5.3 Tamanho das informações de uma mensagem completa do protocolo CMB

Informação	Tamanho (byte)
Token	2
Evento	2
Porta	2
Tempo de Ocorrência	4
Tamanho da mensagem	10

Desta maneira foi possível calcular o tempo que uma mensagem CMB leva para ser transmitida de um processo lógico a outro conforme as características do meio físico utilizado.

A tabela 5.4 mostra os valores de tempo de transmissão obtidos para cada plataforma utilizada nesta etapa do trabalho.

$$T_{transmissão} = t_{latência} + 1/B_{bandwith} * \text{tamanho da mensagem}; \quad (1)$$

Pode-se notar, pela tabela, que as plataformas com meio físico de baixo desempenho (PAB e PBB) foram modeladas com os parâmetros Ethernet (TCP/IP) e às plataformas que possuem meio de comunicação com alto desempenho foram atribuídos valores correspondentes ao meio físico da plataforma Cray T3D (SHMEM).

- **Representação do Tempo de Execução (Processador)**

A classificação dos elementos de processamento de uma plataforma pode ser feita utilizando-se diferentes características ou ferramentas para avaliação de desempenho. A utilização de características dos elementos de processamento, tais como relógio do processador e quantidade de memória disponível, pode fornecer uma idéia de seu desempenho. No entanto, devido à complexidade dos elementos de processamento, essas características nem sempre levam a uma classificação exata.

Tabela 5.4 Tempo de Transmissão de uma Mensagem CMB

	Meio Físico	
Plataforma	**Latência(μs) e Bandwidth (MB/s)**	**Transmissão (s)**
PAA	Cray T3D SHMEM (1,5; 58,5)[2]	0,00000166
PAB	Ethernet TCP/IP	0,003788
SP2	SP2 PVMe (209; 16,5)	0,0002096
CRAY T3D	CRAY T3D PVM (136,7; 25,1)	0,000137
PBA	Cray T3D SHMEM (1,5; 58,5)	0,00000166
PBB	Ethernet TCP/IP	0,003788

Uma forma de comparar sistemas computacionais existentes oferecendo resultados confiáveis é a utilização de *benckmarks*. O problema dessa abordagem é determinar qual *benchmark* melhor representa o tipo de aplicação que será executada no sistema computacional.

---

2 [Marenzoni 1998]

Neste trabalho, para simular o desempenho do elemento de processamento de uma arquitetura foi utilizado como parâmetro o tempo necessário desde que um processo lógico recebeu uma mensagem e finalizou a execução do evento correspondente no modelo de aplicação utilizado.

Para cada modelo de aplicação simulado deve-se atribuir o valor do tempo gasto para que as tarefas sejam realizadas por um determinado recurso (*facility*) do sistema simulado. Desta forma, considerando-se diferentes valores de tempo de execução dessas tarefas pode-se utilizar este valor como medida de desempenho do processador.

Esta alteração no modelo da aplicação não prejudica o comportamento do protocolo CMB. Deve-se evidenciar que as medidas estatísticas relacionadas ao comportamento do protocolo é que são analisadas no final da execução da simulação. As variáveis estatísticas que referem-se ao modelo da aplicação simulada possuem participação apenas no processo de validação da simulação.

A tabela 5.5 mostra os tempos de processamento das mensagens CMB recebidas pelos processos lógicos *cpu()* e *discos()* que compõem um dos estudos de caso analisados neste trabalho e que adiante serão relatados.

Os tempos da tabela 5.5 foram obtidos através da relação entre os valores de desempenho dos processadores (MFlop/s). A partir do valor original dos tempos de execução para cada tarefa da simulação, fornecida na especificação do modelo, foi calculada a quantidade de tempo proporcional ao desempenho do processador.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Nota-se pela tabela 5.5 que as plataformas representativas PAA e PAB são as que possuem valores maiores para a representação do desempenho dos processadores.

Entre as inúmeras características de implementação relacionadas ao modelo da plataforma pode-se destacar a forma com que foi simulado o acesso ao meio físico. Como foi demonstrado pela figura 5.4 os recursos que compõem o modelo da plataforma (meio físico e processadores) também foram representados por *facilities* SMPL. Nesta representação deve-se atribuir a quantidade de servidores SMPL que farão parte de uma determinada *facility*.

Sendo assim foram utilizadas *facilities* com mais de um servidor SMPL para representar canais de comunicação não compartilhados (CRAY T3D e SP2 IBM) que utilizam um *switch* de alto desempenho para a comunicação entre os elementos de processamento. A utilização de vários servidores para representar um recurso (meio físico) possibilitou a representação de um canal de comunicação cuja probabilidade de ocorrência de filas é praticamente nula. Por outro lado, foram utilizados recursos SMPL com apenas um servidor para simular uma política de acesso ao meio físico compartilhado.

Neste tipo de meio de comunicação o tempo de transmissão obtido é decorrente das características do meio de comunicação utilizado, com baixa capacidade de transmissão (10*Mbits/s* em meio de comunicação utilizando *Ethernet*).

Tabela 5.5 Tempo de utilização dos processos *cpu* e *discos* em relação ao desempenho dos elementos de processamento.

	Elemento de Processamento	
Plataforma	**Tempo Cpu e Discos (s)**	**[3](MFlop/s)**
PAA	1,503 4,509	CRAY C90 CF77 5.0 479
PAB	1,503 4,509	CRAY C90 CF77 5.0 479
SP2	10 30	SP2 XLF 3.2 72
CRAY T3D	18,94 56,84	CRAY T3D Craft Cf77 38
PBA	72 216,0	SUN Sparc 1000 55Mhz 10
PBB	72 216,0	SUN Sparc 1000 55Mhz 10

5.3.4 Modelo da Aplicação

Nesta seção são descritos os modelos de aplicações utilizados na análise do comportamento do protocolo CMB utilizando as plataformas descritas na seção 5.3.3.

Uma vez que o objetivo principal desta etapa do trabalho está concentrado no comportamento do protocolo CMB com relação às

---

[3] Obtidos em [Dongarra 2000]

plataformas com características de desempenho diferentes optou-se por utilizar dois modelos de estudo de caso que apresentem características comuns para as classes de modelos de sistemas de filas abertos e fechados.

Define-se um modelo de sistemas de filas *fechado* quando a quantidade de tarefas (ou *tokens*) é constante durante toda a simulação. Nesta classe de modelos estipula-se a quantidade inicial de tarefas que serão executadas durante a simulação e a partir deste momento nenhuma outra nova tarefa é inserida no modelo.

Ao contrário do modelo fechado, um sistema de filas *aberto* contém um número de tarefas flutuante, ou seja, durante a execução do modelo as tarefas podem sair ou chegar novas tarefas dependendo de uma variável de distribuição estatística.

5.3.4.1 Modelos de Aplicação Utilizados

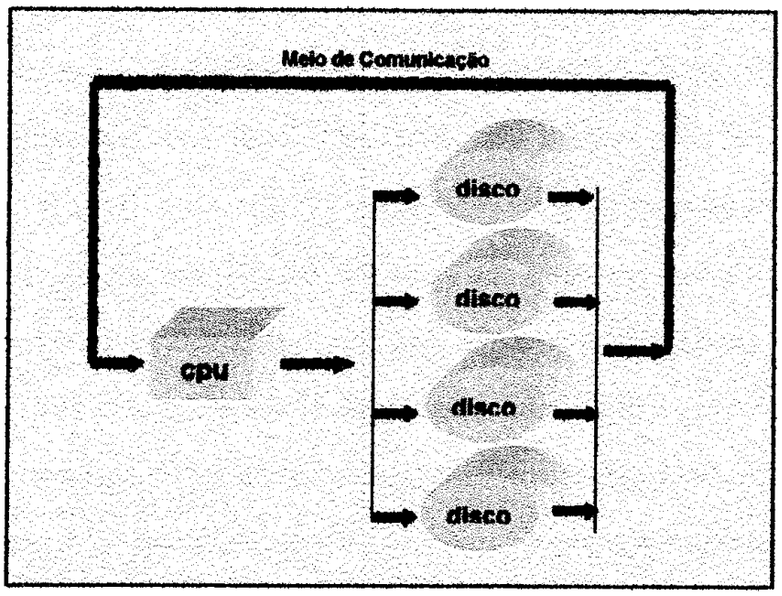
Modelo 1: Servidor Central

Esse modelo representa um sistema computacional constituído por um elemento de processamento e quatro unidades de disco. As tarefas requisitam a utilização do elemento de processamento e, após o término do processamento, uma das unidades de disco é selecionada [MacDougall 1987]. Esse tipo de modelo é bastante utilizado na literatura para a avaliação da simulação seqüencial e também da simulação distribuída, fator que motivou sua escolha [Ayani; Berkman 1993, MacDougall 1987]. Nesse

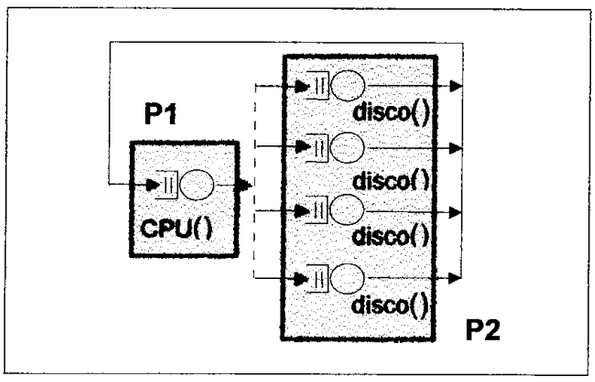
modelo são utilizados cinco recursos que representam a *CPU* e as unidades de disco. Como o modelo é fechado, um número fixo de tarefas (clientes) utiliza os recursos disponíveis no sistema.

Inicialmente, as tarefas são escalonadas para a execução. Uma tarefa requisita a utilização da *CPU*, que executa o processamento quando estiver disponível. Após o término do processamento, uma das unidades de disco é determinada aleatoriamente e essa requisição é escalonada (a solicitação é inserida na fila do recurso que vai executá-la assim que esse fique disponível). Encerrado o processamento na unidade de disco, uma nova requisição para utilizar a *CPU* é escalonada e o ciclo de processamento se repete. As figuras 5.7.a) e 5.7.b) apresentam o modelo de redes de filas para o Modelo do Servidor Central utilizando duas visões diferentes. Na figura 5.7.a) o modelo é apresentado em uma visão mais descritiva com relação ao seus componentes e a dinâmica de execução. Na figura 5.7.b) o mesmo modelo é apresentado com suas características de implementação através das estruturas SMPL.

A parametrização inicial do Modelo do Servidor Central foi obtida em [MacDougall 1987]. As execuções da simulação utilizando o protocolo CMB foram realizadas inicialmente com os parâmetros originais do modelo apresentados pelo autor e, posteriormente, foram realizadas alterações no número de tarefas escalonadas inicialmente para a utilização do sistema. Para representar cada um dos componentes dos modelos utilizados (meio físico e processadores) foi utilizado uma *facility* SMPL. Desta forma, as filas pertencentes aos recursos são geradas automaticamente na definição das *facilities* no início da simulação.



(a) Nessa configuração pode-se observar os recursos utilizados pelo modelo (*CPU* e *discos*) e o fluxo para transmissão de mensagens.



(b) Alocação dos processos lógicos (*CPU* e *discos*) nos processadores *P1* e *P2*. As setas contínuas indicam os canais físicos de comunicação e as setas tracejadas indicam os canais de comunicação lógicos.

Figura 5.7 – Modelo de Redes de Filas: Servidor Central.

Modelo 2: Sistema Computacional Simplificado

Esse modelo apresenta uma visão macroscópica de um sistema computacional, composto por uma unidade de processamento (*CPU*) e duas unidades de disco, representando uma simplificação do Modelo do Servidor Central. Entretanto, nesse modelo a população do sistema, ou seja, o número de tarefas presentes, não é fixa como ocorre no modelo do servidor central. Através dos testes efetuados com o modelo foram determinadas as variações relativas aos tempos entre chegadas de tarefas e a taxa de probabilidade de uma tarefa deixar o sistema, com o intuito de verificar a influência desses parâmetros no comportamento da simulação distribuída.

As tarefas para este modelo são escalonadas para a utilização da *CPU* de acordo com a taxa de chegada (essas requisições são inseridas na fila da *CPU* se a mesma não estiver disponível). Quando disponível, a *CPU* retira uma requisição da fila e a executa. Após o término do processamento, a tarefa pode deixar o sistema ou solicitar (aleatoriamente) os serviços de uma das duas unidades de disco.

Neste caso, uma requisição é escalonada para utilização da unidade de disco correspondente e após a execução, a tarefa retorna para utilização da *CPU*.

O modelo de redes de filas para o sistema computacional simplificado é apresentado nas figuras 5.8.a) e 5.8.b).

5.3.4.2 Características de Implementação do Modelo da Aplicação

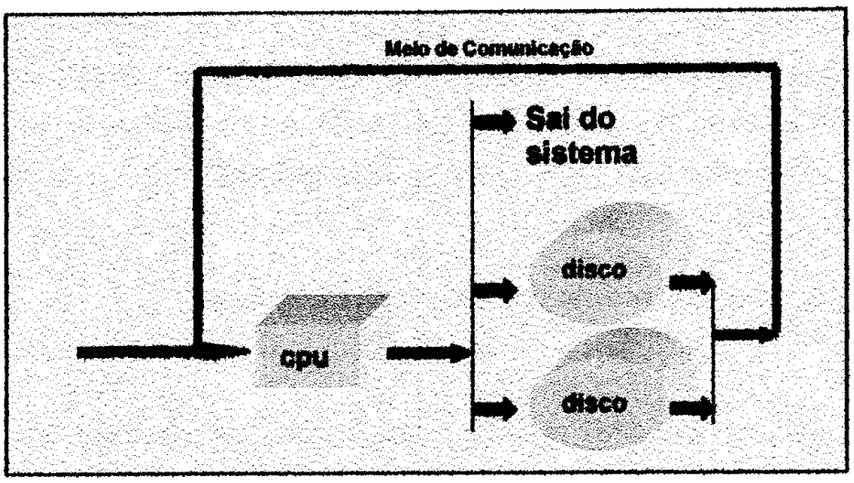
A seguir as principais características de implementação dos modelos utilizados na simulação são detalhadas.

- **Característica Bloqueante de um Modelo**

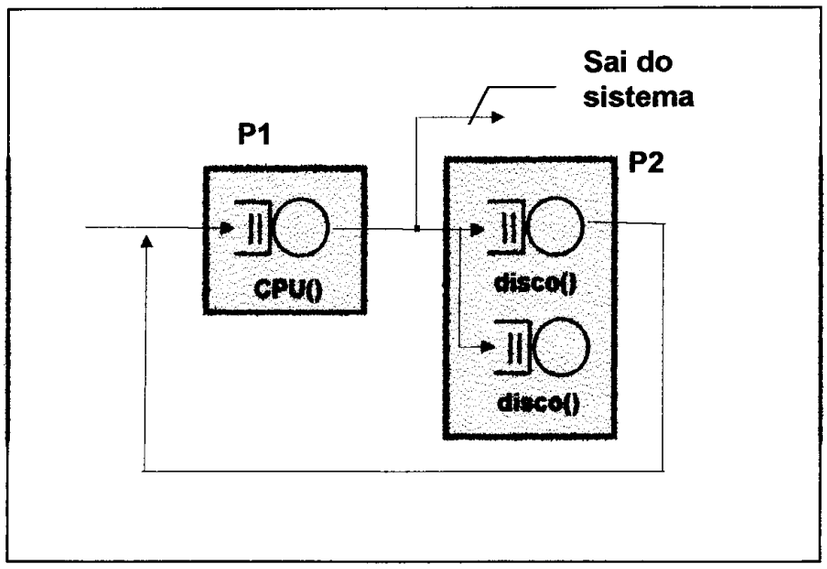
Além dos dois modelos considerados na seção anterior foi implementado um mecanismo que simula características de bloqueio com diferentes intensidades em um único modelo.

A característica de bloqueio de um modelo refere-se a freqüência com que as tarefas são executadas imediatamente após ter atingido a cabeça da lista de eventos futuros. Quando uma mensagem atinge a cabeça da lista de eventos futuros de um processo lógico ela deve aguardar o avanço no clock do processo para que seja executada. Esse avanço no clock do processo depende do menor valor de CC[i], que contém o tempo de ocorrência das mensagens que encabeçam as portas de chegada ao processo. Desta forma, dependendo da quantidade de mensagens recebidas pelo processo lógico esse avanço é realizado em intervalos de tempo diferente.

A variável *bloqueio,* utilizada na implementação dos modelos de aplicação neste trabalho , atribui uma probabilidade às mensagens recebidas pelo processo que indica se esta mensagem irá bloquear ou não o avanço no clock do processo até a chegada de uma mensagem nula. Desta forma, pela alteração no comportamento da execução das tarefas, pode-se



(a) Recursos e Fluxo de mensagens para o modelo Sistema Computacional Simplificado



(b) Alocação de processos e estruturas do modelo Sistema Computacional Simplificado

Figura 5.8 – Modelo de Redes de Filas: Sistema Computacional Simplificado. Depois de utilizar o recurso *cpu* existe a probabilidade da tarefa deixar o sistema.

considerar que para cada um dos modelos descritos anteriormente foi analisado um conjunto (com 10 diferentes valores de probabilidade de bloqueio) de modelos variantes, com a mesma representação de estruturas e recursos, mas com comportamento de execução de tarefas diferentes.

- **Implementação dos Processos Lógicos**

Os recursos CPU e Disco, mencionados na descrição dos modelos, foram representados como processos lógicos no sistema de simulação implementado. Esses processos, representados pelas rotinas *cpu()* e *discos()* foram alocados em 2 processadores separadamente e conectados através de um meio de comunicação. Deve-se mencionar que a análise do comportamento do protocolo CMB com relação a forma de alocação dos processos lógicos nos processadores não está incluída na proposta dessa tese haja vista que esse estudo já foi considerado em [Ulson 1999].

Os processos lógicos implementados podem ser divididos funcionalmente em três partes. A primeira consiste na verificação cronológica das tarefas recebidas. Ou seja, logo que o processo lógico recebe uma mensagem um teste de verificação é realizado para certificar se a mensagem está ou não defasada em relação ao *clock* do processo.

A segunda parte consiste em determinar probabilisticamente se a tarefa, caso seja uma mensagem completa, irá ou não bloquear o processo lógico. As possíveis situações que envolvem a chegada de uma mensagem ao processo lógico foram descritas na seção 5.3.2.

A terceira e última parte envolve as operações de atualização das variáveis de estado e de tempo do processo e do clock da simulação.

5.4 Métodos Utilizados para a Validação

A etapa de validação do modelo de simulação utilizado para analisar o protocolo CMB pode ser dividido em duas partes. A primeira parte refere-se a validação do modelo do protocolo CMB e a segunda à validação da simulação considerando os modelos da plataforma e da aplicação.

5.4.1 Validação do Modelo para o Protocolo CMB

A proposta original do protocolo CMB prevê um comportamento bem característico para um conjunto específico de situações que podem ocorrer durante a simulação.

Desta forma o modelo do protocolo CMB foi submetido intencionalmente a esse conjunto de situações e de acordo com o comportamento observado pode-se validar o modelo do protocolo (Figura 5.9) . A tabela 5.6 relaciona o conjunto de possíveis situações com o comportamento esperado pelo protocolo CMB.

Para cada uma das situações provocadas deve-se mencionar que o modelo construído comportou-se de acordo com as especificações contidas na proposta original do protocolo validando, desta forma, a implementação do modelo do protocolo CMB.

5.4.2 Validação do Modelo da Plataforma e da Aplicação

Para realizar a validação do modelo da plataforma e da aplicação foi realizada, através dos valores obtidos por aferição [Ulson 1999], uma comparação entre os relatórios estatísticos fornecidos pela simulação do protocolo CMB apresentada nesta tese e a implementação do protocolo efetuada em [Ulson 1999].

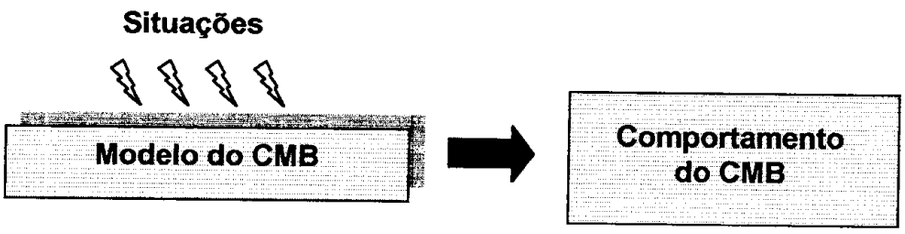


Figura 5.9 Validação do Modelo do Protocolo CMB. A validação do modelo CMB é realizada através da observação do seu comportamento em comparação ao protocolo original.

Para isso foram utilizados os mesmos modelos de aplicação (Servidor Central) e os mesmos parâmetros de desempenho para o modelo da plataforma (SP2 IBM). Desta forma através da equivalência entre os valores fornecidos (quantidade de mensagens nulas e completas transmitidas, porcentagem de utilização dos servidores SMPL, tamanho médio de fila dos servidores, porcentagem de mensagens nulas, tempo de simulação, etc.) pelos relatórios das duas abordagens (aferição e modelagem) validou-se os modelos da plataforma e da aplicação. Os relatórios utilizados na validação encontram-se no apêndice B desta tese.

Tabela 5.6 – Validação do Modelo do Protocolo CMB. Para cada situação provocada espera-se um comportamento dentro das regras propostas no modelo original do protocolo CMB.

Situação	Comportamento Esperado
Chegada de uma mensagem defasada	Bloqueio do processo
Processo sem tarefa e não recebe mensagem nula	Bloqueio do processo
Processo "bloqueado" e chega uma mensagem nula	Desbloqueio do processo
Processo não tem mensagens para processar	Envia mensagem nula
Mensagens nulas no sistema	Não devem interferir na simulação
Processo executa uma mensagem	Processo deve enviar uma mensagem nula com o valor do tempo de ocorrência da mensagem a todos os processos lógicos

5.5 Estratégia de Execução

Como foi mencionado anteriormente, o objetivo da implementação da simulação descrita neste capítulo é analisar o desempenho do protocolo CMB considerando-se as características da plataforma utilizada e o do modelo do sistema real que está efetivamente sendo simulado. Estas características compreendem a carga de mensagens nulas, o tempo de simulação utilizado, a taxa de chegada de mensagens completas, o desempenho dos processadores e do meio físico e o comportamento bloqueante do modelo.

A partir da análise destes parâmetros, em conjunto com os resultados obtidos em [Ulson 1999], pode-se definir medidas nas quais a troca de protocolos possa ser realizada no sentido CMB para Time Warp. O comportamento do protocolo Time Warp está sendo estudado em paralelo a este trabalho como parte da tese de outro programa de doutorado.

Na avaliação do comportamento do protocolo *CMB* foram executados os modelos descritos na seção 5.3.4.1 e suas variantes. Para garantir que os dados obtidos estivessem estatisticamente corretos, todos os testes foram executados várias vezes com o objetivo de aumentar o tamanho da amostra e garantir a precisão dos dados. Para cada conjunto de dados obtido foram determinados média aritmética, desvio padrão, variância e outros valores relacionados à amostra, além dos testes estatísticos necessários (o Apêndice A).

Para cada modelo e plataforma considerados neste trabalho a estratégia de execução foi baseada nas trocas de valores para as variáveis: probabilidade de bloqueio, quantidade de tarefas no sistema (para modelos fechados), taxa de chegada de tarefas (para modelos abertos), quantidade de ciclos realizados pelas tarefas (condição de parada da simulação) e troca de sementes para garantir resultados aleatórios. Desta forma pode-se representar a execução das simulações através da figura 5.10. Nela cada quadro representa uma estrutura de repetição e o rótulo do quadro indica a variável modificada.

Para cada execução foi gerado um arquivo do tipo texto contendo os relatórios estatísticos de cada simulação. A análise dos resultados é apresentada no capítulo 6.

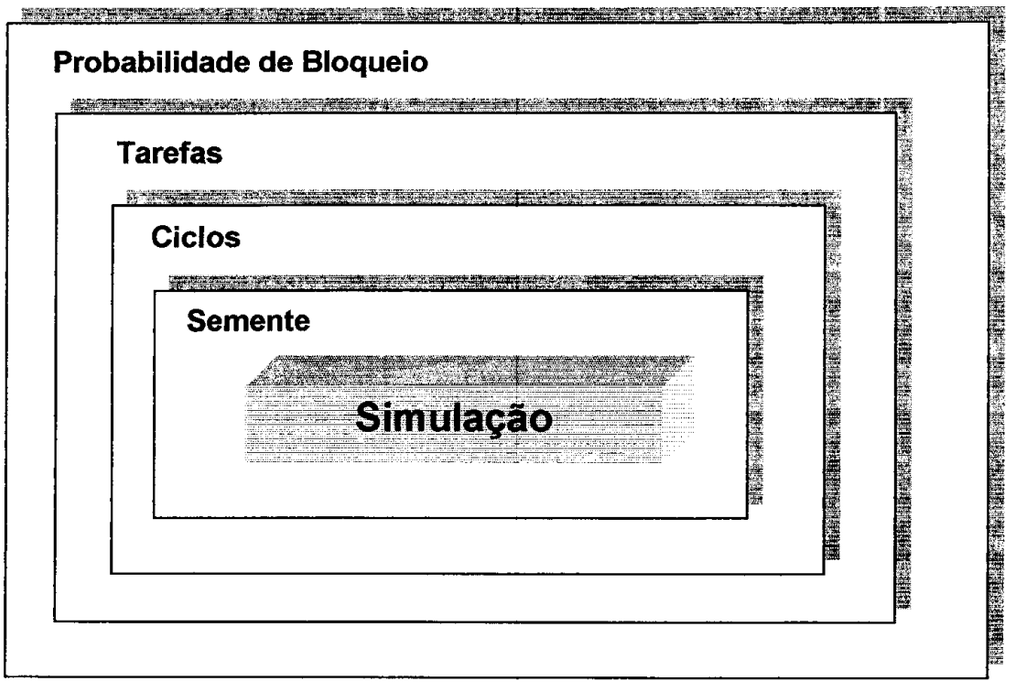


Figura 5.10 Estratégia de Execução das Simulações.

5.6 Conclusões

A proposta de um mecanismo de troca de protocolos de sincronização para uma simulação distribuída é bastante atraente uma vez que a determinação de qual o melhor protocolo de sincronização para um determinado modelo não é uma tarefa simples. Além das características iniciais do modelo deve-se considerar que o comportamento do sistema simulado pode ser modificado dinamicamente durante sua execução.

Entretanto, além da determinação do modo com que a troca de protocolos é realizada (capítulo 4) deve-se considerar qual o melhor momento em que a troca de protocolos irá contribuir para o aumento do desempenho da simulação.

Desta forma foram apresentadas neste capítulo as características do modelo e da implementação de um sistema de simulação do protocolo de sincronização CMB, utilizando a extensão funcional SMPL [MacDougall 1987], onde são consideradas as características da plataforma do hardware utilizado para a execução e da aplicação utilizada como estudo de caso para a simulação. O objetivo do sistema de simulação descrito é definir o momento em que o protocolo CMB deixa de ser interessante e a troca para o protocolo Time Warp deve ser considerada.

Através da flexibilidade fornecida pelas técnicas de simulação foi construído um conjunto de modelos de plataformas cujas características de desempenho dos elementos de processamento e dos meios de comunicação abrangessem as principais plataformas disponíveis atualmente.

Da mesma forma, um modelo do protocolo CMB foi implementado para sincronizar a simulação. Para que o modelo do protocolo CMB fosse validado foi realizado um plano de estratégia no qual o modelo construído é submetido a uma série de situações especiais cujas reações são comparadas com as reações esperadas para o protocolo CMB. Comparações entre os valores obtidos em casos particulares e resultados obtidos em um sistema real foram efetuadas.

Foram considerados, como estudos de caso, os modelos do Sistema de Servidor Geral e o Sistema Computacional Simplificado. Esses modelos, bastante utilizados e característicos para a análise de desempenho de simulações distribuídas, possuem características interessantes que

possibilitaram a avaliação do comportamento do protocolo CMB sob diferentes aspectos com relação à natureza do modelo de aplicação.

No capítulo 6, é apresentada uma análise do comportamento do protocolo CMB, realizada através dos resultados adquiridos com as execuções das simulações utilizando-se o sistema de simulação descrito neste capítulo.

Capítulo 6

Comportamento do Protocolo CMB: Análise dos Resultados

Este capítulo apresenta a análise dos resultados obtidos através da execução da simulação descrita no capítulo 5. Os gráficos de comportamento das simulações em relação às variáveis consideradas na estratégia de execução também são analisados. As características de desempenho do protocolo frente às diferentes plataformas e modelos são apresentadas como parâmetro de medida para a realização da troca de protocolo.

6.1 Introdução

Este capítulo apresenta uma análise dos principais resultados obtidos através da execução da simulação descrita no capítulo 5. O processo de execução das simulações, envolvendo os modelos de aplicação e as diferentes plataformas representativas, descritas no capítulo anterior, gerou medidas estatísticas que, através de suas análises, fornecem um panorama do comportamento do protocolo CMB frente a um conjunto de variáveis de contexto. A obtenção das características do comportamento do protocolo CMB permite formular sob quais condições o protocolo CMB não apresenta desempenhos satisfatórios sugerindo portanto a troca de protocolos de sincronização.

O processo de análise de resultados é apresentado em duas etapas. A primeira considera as arquiteturas individualmente (ou seja, envolve a análise do comportamento para cada plataforma utilizada) e a segunda etapa fornece uma análise comparativa (envolvendo o comportamento das simulações entre as diferentes plataformas utilizadas). Cada uma dessas etapas foi subdividida em várias seções, cada seção está relacionada à um conjunto de medidas estatísticas fornecidas pelos relatórios das simulações (porcentagem de mensagens nulas, tempo simulado, utilização de recursos da simulação, etc.)[1].

---

[1] Análises adicionais que consideram medidas estatísticas mais específicas podem ser encontradas no Apêndice C.

A realização dos testes de hipóteses, que tem por objetivo verificar se as diferenças observadas para as medidas utilizadas para a análise do modelo de simulação são estatisticamente significativas, é apresentada no Apêndice A.

Para que a análise dos resultados seja melhor compreendida, a seguir são definidas as variáveis de desempenho e medidas estatísticas utilizadas na realização das simulações. No final deste capítulo são apresentadas as conclusões finais.

6.2 Variáveis de Desempenho e Medidas Estatísticas Utilizadas para Análise dos Resultados

Durante a execução de uma simulação distribuída, sincronizada pelo protocolo CMB, algumas variáveis caracterizam o andamento da simulação e são utilizadas como parâmetro para a execução dos modelos definidos no capítulo 5. Analisando-se os resultados obtidos com a simulação descrita no capítulo 5 será possível caracterizar as condições em que a simulação CMB apresenta um baixo desempenho e portanto deve-se considerar a troca de protocolo. A figura 6.1 apresenta os parâmetros de entrada e de saída utilizados para a execução das simulações. Esses parâmetros são discutidos nas seções 6.2.1 e 6.2.2 .

6.2.1 Parâmetros de Entrada

Para cada execução da simulação é utilizado um conjunto de parâmetros cujos valores identificam sob quais condições a simulação será realizada.

Figura 6.1. Parâmetros de Entrada e Saída. Conjunto de valores que caracterizam o contexto da simulação.

A seguir são definidos os principais parâmetros de entrada:

- **% Bloqueio**

A forma com que as tarefas são realizadas durante a execução de uma simulação pode mudar de modelo para modelo. No CMB uma tarefa permanece bloqueada na lista de eventos futuros quando seu tempo de ocorrência ainda não foi atingido pelo seu processo lógico. Quando isto ocorre este processo lógico emite uma mensagem nula (CMB original) indicando que o processo não executará nada até o tempo de ocorrência da mensagem bloqueada. Em alguns modelos existe uma predominância das tarefas que raramente ficam "bloqueadas".

(mensagem completa) ficar bloqueada até que o LVT do processo lógico seja atualizado por uma mensagem nula.

Desta forma, do ponto de vista do desempenho da simulação, pode-se dizer que a quantidade de mensagens bloqueadas em um determinado modelo influi diretamente na quantidade de mensagens nulas emitidas pelos processos lógicos da simulação, aumentando assim a sobrecarga nos meios de comunicação e no mecanismo de sincronização do protocolo CMB.

- **Tarefas/Ciclos e Entradas/Chegadas**

A quantidade de mensagens completas (ou tarefas) que circularão pela simulação é definida através das variáveis *Tarefas* e *Entradas*. Nos modelos fechados a quantidade de tarefas é fixa durante toda a simulação e o seu valor é definido pela variável *Tarefas*. Essas tarefas deverão realizar uma quantidade predeterminada de ciclos (representada pela variável *Ciclos*) até que a simulação termine. Por outro lado, nos modelos abertos não existe uma quantidade fixa de tarefas que circulam pelo sistema. Sendo assim, a quantidade máxima de tarefas que é inserida nesta classe de modelos durante a simulação é definida pela variável *Entradas,* cujo valor indica a quantidade máxima de mensagens que o modelo deverá receber durante a simulação (condição de término da simulação). A variável *Chegadas,* nos modelos fechados, representa o tempo médio entre chegadas das mensagens que serão inseridas no modelo. Desta forma, pode-se dizer que quando a simulação receber uma quantidade *Entradas* de tarefas, com o tempo entre chegadas igual a *Chegadas*, a simulação termina.

Uma vez que as mensagens completas são responsáveis pelo andamento dos valores dos *clocks* dos processos lógicos (LVT), o número de mensagens completas que circulam por um modelo influi diretamente no desempenho da simulação.

6.2.2 Parâmetros de Saída

Existe um conjunto de valores, resultantes da execução das simulações, que servirão como parâmetros para realizar a análise do comportamento do protocolo CMB. Pode-se destacar entre eles:

- **Tempo Simulado**

Esta variável representa a quantidade de tempo gasto para que a simulação atinja a condição de parada. Esse valor é obtido através da soma dos tempos necessários para que as tarefas utilizem os recursos da simulação (meio físico e processadores) até que a condição de parada da simulação seja satisfeita.

Quando uma simulação distribuída possui como condição de parada o avanço da simulação até um valor predeterminado de tempo de execução, é esperado que quanto maior o tempo simulado, melhor o desempenho da simulação. Entretanto, torna-se importante ressaltar que, nas simulações consideradas nesta tese, as condições de parada estão representadas através do número de ciclos que as tarefas devem executar (para o modelo fechado) e pela quantidade de tarefas que chegam ao sistema (para o modelo aberto).

Sendo assim, ao contrário do que se poderia esperar em uma simulação onde a condição de parada está baseada na sua execução até um valor preestabelecido de tempo, nas simulações realizadas nesta tese, quanto menor for o tempo necessário para a execução dos ciclos ou das tarefas melhor será o desempenho da simulação. Essa situação acontece porque a simulação considerada está simulando uma simulação distribuída e o tempo simulado é, na realidade, o tempo gasto para simular um conjunto de eventos. Desta forma, através do valor do tempo simulado, pode-se obter uma visão do desempenho do protocolo CMB na simulação em relação aos parâmetros de entrada (quantidade de tarefas, plataforma utilizada e porcentagem de bloqueio).

- **% Nulas**

A variável *%nulas* é utilizada para medir a quantidade de mensagens nulas transmitidas durante a simulação. Este valor é calculado em relação à quantidade total de mensagens transmitidas pelos processos lógicos durante a simulação. Comparando o tempo simulado com as porcentagens de mensagens nulas obtidas durante a simulação pode-se determinar qual a influência dessas mensagens no desenvolvimento da simulação.

- **Utilização de Recurso**

Esta medida fornece a porcentagem de utilização de um determinado recurso (uma *facility* SMPL) durante toda a simulação. Em outras palavras, pode-se medir a sobrecarga causada pelo aumento da quantidade de mensagens (nulas e completas) através do valor da utilização do recurso.

A seguir, as seções 6.3 e 6.4 apresentam os resultados obtidos com a simulação. Para facilitar a compreensão dos gráficos que serão apresentados, ao lado de cada um deles existe uma tabela de configuração contendo os valores utilizados para compor o contexto da simulação. O exemplo da figura 6.2 mostra uma tabela de configuração para uma simulação utilizando o modelo Sistema de Computação Simplificado (SCS) com a plataforma representativa AA, variando a porcentagem de bloqueio (% Bloqueio) e a quantidade de tarefas e, finalmente, executando 10000 ciclos.

6.3 Análise Individual

Esta etapa da análise dos resultados mostra as características do comportamento do protocolo CMB considerando os aspectos individuais de cada plataforma. Sendo assim, não existe nesta etapa da análise, nenhuma comparação entre os desempenhos obtidos pelas diferentes plataformas consideradas neste trabalho.

6.3.1 Relação entre a Quantidade de Mensagens Nulas e a Probabilidade de Bloqueio

Neste seção é analisado o comportamento do número de mensagens nulas em relação à característica bloqueante do modelo (% bloqueio). O objetivo é observar a relação entre o número de mensagens nulas geradas por um modelo e a quantidade de tarefas bloqueadas durante a simulação.

A figura 6.3 mostra a relação entre a quantidade de mensagens nulas geradas e a probabilidade de bloqueio das tarefas para o modelo do sistema computacional simplificado.

Através dos resultados pode-se dizer que a menor quantidade de mensagens nulas observada incide nos modelos 0%-bloqueantes. Isto pode ser explicado através da maneira com que as mensagens nulas são geradas.

Área do Gráfico

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
SC	SP2	0	50	➢ 10000
➢ SCS	CRAYT3D	10	100	30000
	➢ AA	20	250	50000
	AB	30	500	var.
	BA	40	➢ var.	
	BB	50		
	var.	60		
		70		
		80		
		90		
		100		
		➢ var.		

Figura 6.2 Tabela de Configuração. Para cada gráfico apresentado neste capítulo existe um conjunto de valores específicos que indicam a configuração utilizada na execução da simulação.

Como foi mencionado anteriormente, quando uma tarefa não é executada imediatamente após atingir a cabeça da lista de eventos futuros o processo lógico deve enviar uma mensagem nula. Nestes modelos (0% de bloqueio) as tarefas que chegam em um processo lógico são imediatamente escalonadas para executarem seus eventos. Portanto, não existe a necessidade de alterar o valor dos relógios dos processos lógicos. Entretanto, podem ocorrer situações nas quais um processo lógico fique com

sua lista de ventos futuros *vazia*, nestas circunstâncias uma mensagem nula deve ser enviada pelo processo que contém esta lista para indicar essa ausência de processamento. Desta forma, mesmo com um modelo 0%-bloqueante os processos podem enviar mensagens nulas. Para o modelo do servidor central (Figura 6.4) a quantidade de mensagens nulas enviadas para indicar lista de eventos futuros vazia não ultrapassou a marca de 20% em relação ao número total de mensagens circulando pelo sistema. Por outro lado, quando existe um aumento da probabilidade de bloqueio, a quantidade de mensagens nulas, mesmo para modelo do sistema computacional simplificado, aumenta em relação ao número total de mensagens, sendo que em todos os casos estudados a quantidade de mensagens nulas tende a uma porcentagem de 60%.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	SP2	0	1000	➢ 1.0
➢ SCS	CRAY	10	➢ 2000	50.0
	➢ AA	20		100.0
	AB	30		var.
	BA	40		
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

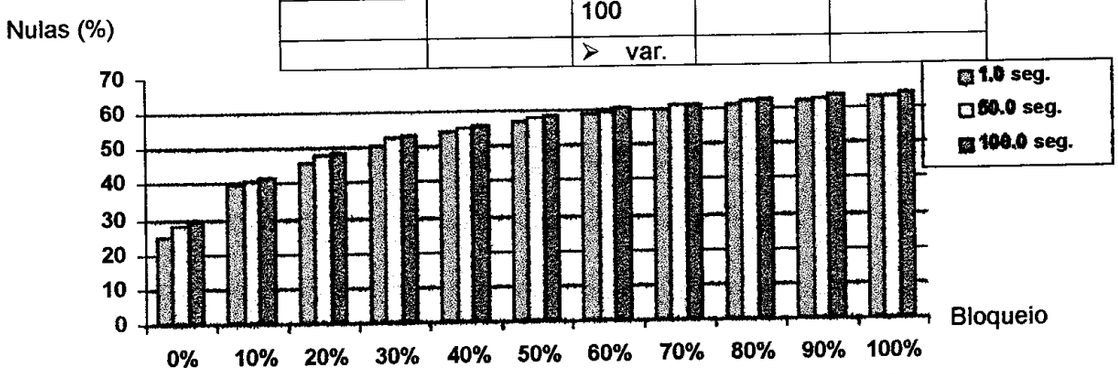


Figura 6.3 Probabilidade de Bloqueio x % Nulas: Arquitetura AA

Em [Ulson 1999] onde as mesmas configurações foram utilizadas, este comportamento também foi observado por aferição. Através da observação deste comportamento, pode-se afirmar que a quantidade de mensagens nulas transmitidas durante uma simulação é uma consequência da natureza bloqueante do modelo simulado. Modelos altamente bloqueantes necessitam de uma quantidade maior de mensagens nulas para satisfazer as mensagens bloqueadas. Este mesmo comportamento foi observado para diferentes quantidades de tarefas no sistema e para diferentes números de ciclos realizados por estas tarefas.

No modelo do sistema computacional simplificado as quantidades de mensagens nulas ultrapassaram ligeiramente os valores atingidos no modelo do servidor central. Essa pequena elevação na porcentagem de mensagens nulas foi ocasionada pela alta taxa de mensagens completas utilizadas no modelo aberto.

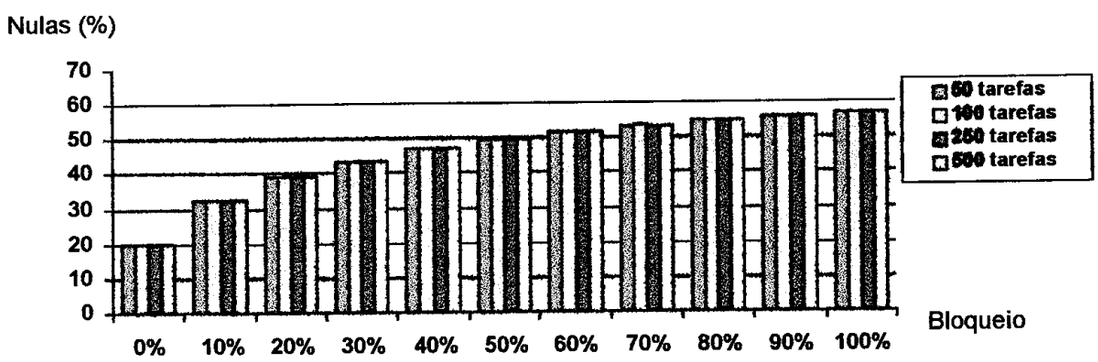


Figura 6.4 Probabilidade de Bloqueio x % Nulas: Modelo SC

6.3.2 Influência da Probabilidade de Bloqueio do Modelo e das Mensagens Nulas em Relação ao Tempo Simulado

Foi verificado na seção anterior que a relação entre a probabilidade de bloqueio das mensagens (independentemente do modelo e da quantidade de mensagens completas) e a quantidade de mensagens nulas indica que quanto maior a probabilidade de bloqueio do modelo, maior é a quantidade de mensagens nulas geradas durante a simulação. Desta forma, a análise das influências da probabilidade de bloqueio e da porcentagem de mensagens nulas no tempo simulado são equivalentes.

A influência das mensagens nulas em uma simulação distribuída utilizando o protocolo CMB tem sido foco de intensa discussão durante os últimos anos [Jha; Bagrodia 1994, Das 1996, Spolon et al. 1999]. A possibilidade de uma degradação no desempenho de simulações devido às altas taxas de mensagens nulas representa uma grande desvantagem dos métodos baseados em mensagens nulas.

Neste trabalho a influência das mensagens nulas no desempenho da simulação é medido através do valor do tempo de simulação (definido na seção 6.2.2) uma vez que as mensagens nulas concorrem com as mensagens completas utilizando os mesmos recursos físicos da simulação e não provocam um avanço na simulação dos processos.

Percebe-se, pelos gráficos das figuras 6.5 e 6.6, que ao contrário do esperado, os piores valores de tempo de simulação foram obtidos para os modelos com probabilidade de bloqueio entre 30 e 80 %. Dentro deste

intervalo os modelos com 50% de bloqueio apresentam os tempos máximos obtidos para as simulações.

Para quantificar esse comportamento, a tabela 6.1 mostra as medidas proporcionais entre os tempos de simulação, obtidos com a plataforma AA, para modelos com probabilidades de bloqueio maior que zero e modelos sem probabilidade de bloqueios (0% de Bloqueio).

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	SP2	0	50	10000
SCS	CRAYT3D	10	100	30000
	➢ AA	20	250	➢ 50000
	AB	30	500	var.
	BA	40	➢ var.	
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

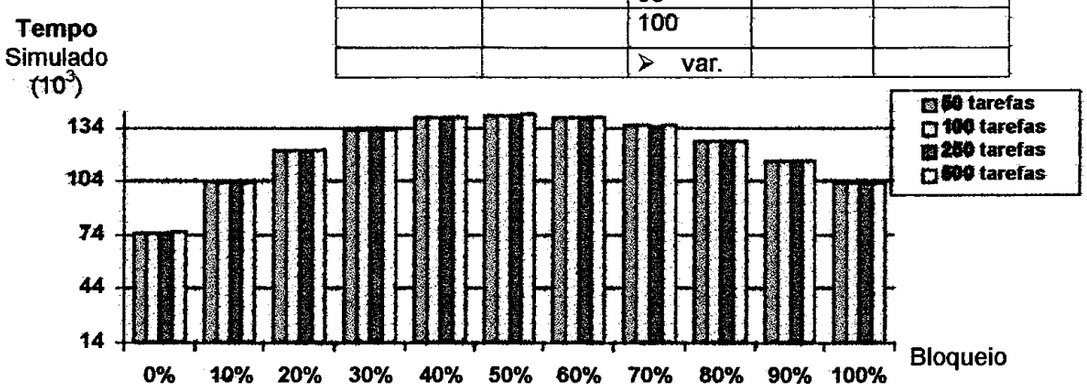


Figura 6.5 Probabilidade de Bloqueio x Tempo Simulado: Arquitetura AA

Nota-se nessa tabela que as simulações cujas tarefas possuem 20% de probabilidade de esperarem para serem executadas até que o *clock* do processo (LVT) atingisse sua marca de tempo de ocorrência (20% de bloqueio) apresentaram tempos de simulação em média 62% maior que aquelas cujas tarefas eram executadas imediatamente após atingir a cabeça da lista de eventos futuros (0% de bloqueio).

Tabela 6.1 Variação dos tempos de simulação em relação à probabilidade de bloqueio: Plataforma AA.

%Bloqueio	0	20	40	50	60	80	100
% Aumento Relativo	---	62	85	90	87	69	38

Percebe-se também, que os tempos de simulação atingem um valor máximo quando a probabilidade de bloqueio está próximo de 50%. Após esta marca os valores de tempo começam a baixar mas não chegam a atingir a marca mínima obtida com 0% de probabilidade. A explicação desse comportamento, conforme detalhado nos parágrafos a seguir, baseia-se na quantidade de mensagens nulas e no tempo que os processos lógicos ficam bloqueados.

Para facilitar o entendimento pode-se dividir o comportamento do tempo simulado em relação à probabilidade de bloqueio (observado nos gráficos 6.5 e 6.6) em duas partes. A primeira parte refere-se aos valores de tempo obtidos no intervalo de probabilidade de bloqueio 0% - 50%. Neste intervalo, os valores do tempo de simulação aumentam à medida que a probabilidade de bloqueio também aumenta. Isso ocorre porque o número de mensagens nulas transmitidas também aumenta, e desta forma, a sobrecarga causada pelo aumento das mensagens nulas é responsável pelo aumento dos tempos de simulação neste primeiro intervalo.

Na segunda parte dos gráficos 6.5 e 6.6, relativa ao intervalo de probabilidade de bloqueio 50% - 100%, existe uma diminuição nos valores de tempo de simulação. Esse comportamento é explicado ao considerar que, para estes valores de probabilidade de bloqueio, a quantidade de tarefas

aguardando um avanço no *clock* dos processos lógicos (bloqueadas) é bastante alta. Desta forma, a chegada de uma mensagem nula pode desbloquear mais de uma mensagem completa. Em outras palavras, quanto maior o número de mensagens bloqueadas, mais eficientes tornam-se as mensagens nulas. Associado a este comportamento, observa-se que com o crescimento da porcentagem de bloqueio a porcentagem de mensagens nulas cresce até um ponto, quando acontece uma saturação (Figuras 6.3 e 6.4). Desta forma, a diminuição do crescimento de mensagens nulas auxilia no comportamento observado nos gráficos das figuras 6.5 e 6.6.

Apesar do comportamento decrescente obtido no intervalo 50% - 100%, nota-se que o aumento das mensagens nulas e das mensagens bloqueadas não permitiu que o desempenho do intervalo 0% - 50% fosse superado.

Desta forma, verifica-se que os modelos com características de bloqueio próximas a 50% são aqueles onde o protocolo de sincronização CMB, nas condições apresentadas para estas simulações, apresentam um baixo desempenho. Sendo assim, pode-se dizer que a substituição do protocolo CMB pelo Time Warp deve ser realizada quando a relação mensagens nulas/tarefas bloqueadas é insuficiente para promover um avanço com maior freqüência no *clock* dos processos. Nesta situação, o tempo total de simulação aumenta diminuindo o desempenho da simulação.

Estes comportamentos foram verificados em todos os modelos de aplicação utilizados neste trabalho e levam a concluir que, uma vez que os piores tempos de simulação foram obtidos com quantidades intermediárias de mensagens nulas, uma porcentagem maior de mensagens nulas (em

relação ao número total de mensagens no sistema) *não implica* necessariamente em um tempo de simulação maior.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	SP2	0	1000	1.0
➢ SCS	➢ CRAY	10	➢ 2000	50.0
	AA	20	var.	100.0
	AB	30		➢ var.
	BA	40		
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

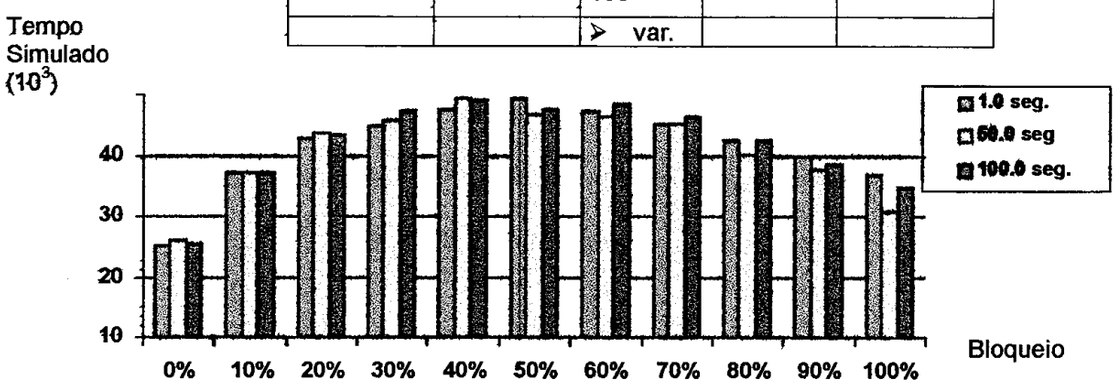


Figura 6.6 Probabilidade de Bloqueio x Tempo Simulado: Arquitetura CRAY

Uma forma alternativa de comprovar a afirmação do parágrafo anterior pode ser obtida através da observação dos gráficos das figuras 6.7 e 6.8. Nestes gráficos estão representados os valores dos tempos de simulação em relação à porcentagem de mensagens nulas geradas durante a simulação. Nota-se que, apesar da quantidade de mensagens nulas crescer desde o início do gráfico, os tempos de simulação apresentam uma melhora a partir dos valores maiores que 50% - 60% de mensagens nulas.

Configuração				
Modelo	*Plataf.*	*% bloq.*	*tarefas*	*ciclos*
➢ SC	SP2	0	50	10000
SCS	➢ CRAY	10	100	30000
	LAN	20	250	➢ 50000
		30	➢ 500	var.
		40	var.	
		50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

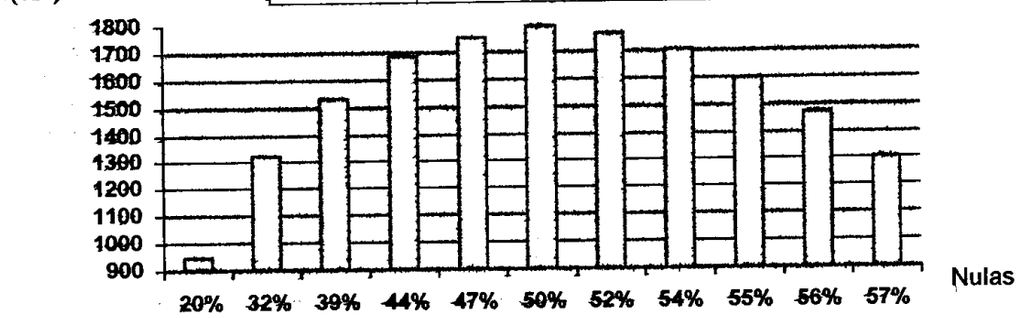


Figura 6.7 Tempo Simulado x % Nulas: Arquitetura CRAY Modelo SC

6.3.3 Influência das Mensagens Nulas no Desempenho dos Recursos da Simulação.

O objetivo deste item é mostrar o comportamento dos recursos (*facilities*) empregados na simulação em relação às alterações da quantidade de mensagens nulas.

Seguindo um princípio básico do protocolo CMB, as mensagens nulas não devem influenciar nos resultados estatísticos da simulação. As mensagens nulas devem apenas atualizar o *clock* do processo lógico que irá recebê-las.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	SP2	0	1000	1.0
➢ SCS	➢ CRAY	10	➢ 2000	50.0
	AA	20	var.	➢ 100.0
	AB	30		var.
	BA	40		
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

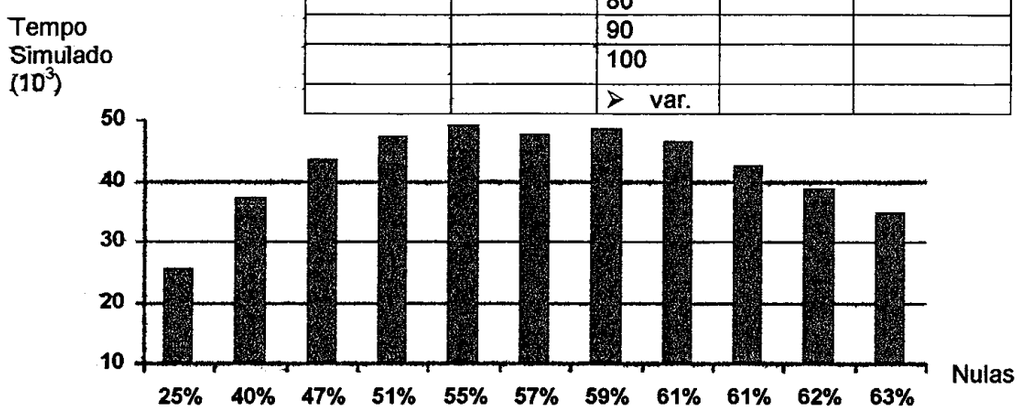


Figura 6.8 Tempo Simulado x % Nulas: Arquitetura CRAY Modelo SCS

A medida de utilização do processo lógico *cpu* (do modelo da aplicação), identificada nos gráficos 6.9 e 6.11, considera apenas o processamento das tarefas (mensagens completas). O parâmetro *Utilização* para um recurso da simulação, fornecido pelo relatório original da extensão SMPL, indica a quantidade de tempo (medida em porcentagem) em que o recurso manteve-se ocupado.

Nos modelos utilizados, as tarefas iniciam seu ciclo sendo processadas pelo recurso *cpu* (processo lógico). Logo após, através de uma medida de probabilidade, essas tarefas são direcionadas para um dos *discos* que compõem os modelos. Essa característica do modelo torna este recurso o mais utilizado na simulação e por este motivo a *cpu* foi escolhida para

descrever o comportamento dos recursos da simulação do modelo de aplicação em relação às mensagens nulas.

Para o modelo do servidor central, o comportamento dos valores de período de utilização mostra-se inversamente proporcional ao comportamento do tempo de simulação (compare os gráficos das figuras 6.9 e 6.10). Ou seja, os menores valores de utilização do recurso *cpu* são para o intervalo 43 – 53 % de mensagens nulas e os maiores tempos de simulação se encontram neste mesmo intervalo. Este comportamento é explicado ao considerar que a *cpu* é o recurso que, devido a sua alta utilização em comparação aos *discos*, tem uma influência muito grande no desempenho da simulação. Desta forma, pode-se dizer que, no período em que existia uma grande quantidade de mensagens completas bloqueadas ou utilizando os *discos*, não houve uma freqüência de avanços significativos nos *clocks* da simulação por causa da baixa quantidade de tarefas utilizando a *cpu*.

O gráfico de utilização de recursos para o modelo do sistema computacional simplificado (figura 6.11) mostra o mesmo comportamento descrito anteriormente.

Estas constatações levam a considerar a possibilidade da troca de protocolos de sincronização quando o modelo submetido à simulação sincronizada pelo protocolo CMB possuir períodos de escassez (provocados pelas características do modelo da aplicação considerada) de mensagens completas circulando pelo sistema, uma vez que, para uma simulação conservativa, onde o avanço dos *clocks* dos processos lógicos é essencial para obter desempenhos satisfatórios, é importante existir uma frequente atualização dos *clocks*.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	SP2	0	50	10000
SCS	CRAY	10	100	30000
	➢ AA	20	250	➢ 50000
	AB	30	➢ 500	var.
	BA	40	var.	
	BB	50		
	var.	60		
		70		
		80		
		90		
		100		
		➢ var.		

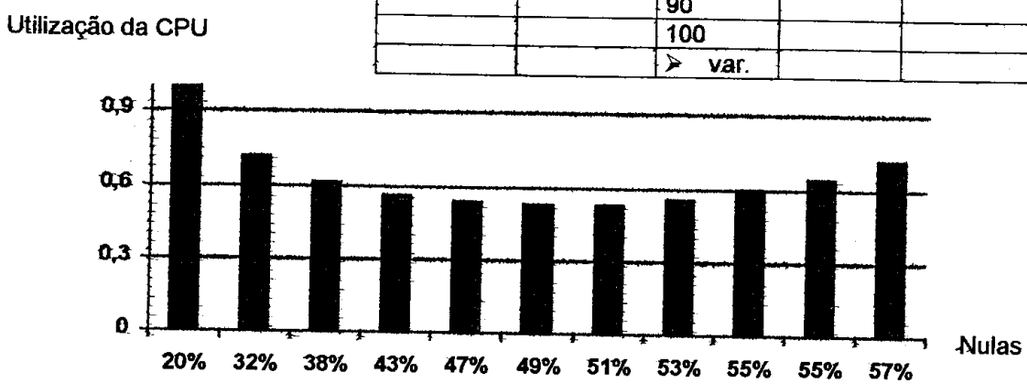


Figura 6.9 Utilização da CPU x % Nulas: Plataforma AA

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	SP2	0	50	10000
SCS	CRAY	10	100	30000
	➢ AA	20	250	➢ 50000
	AB	30	➢ 500	var.
	BA	40	var.	
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

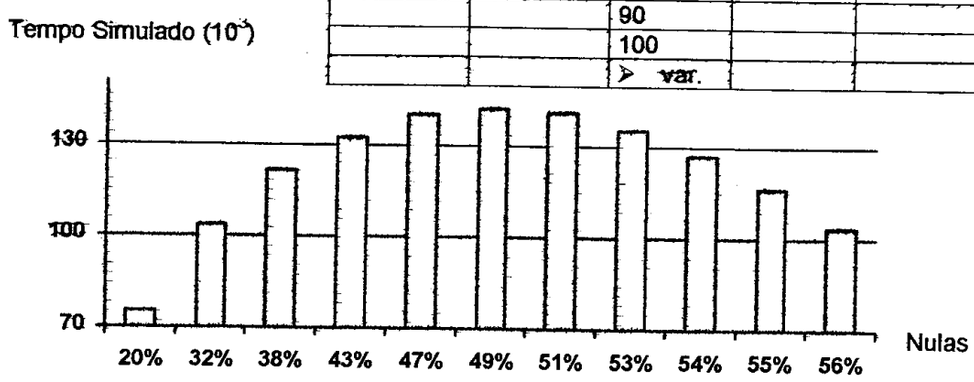


Figura 6.10 Tempo Simulado x % Nulas: Plataforma AA

Por outro lado, em um protocolo otimista como o Time Warp, as execuções das tarefas não dependem exclusivamente dos valores dos LVT's.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	SP2	0	1000	➢ 1.0
➢ SCS	CRAYT3D	10	➢ 2000	50.0
	➢ AA	20	var.	100.0
	AB	30		var.
	BA	40		
	BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

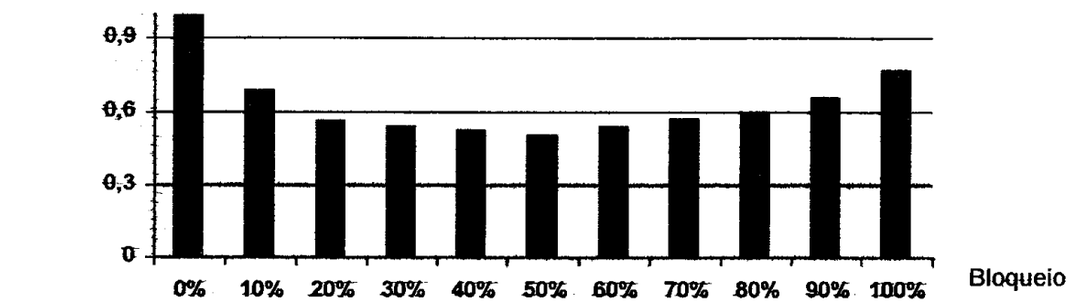


Figura 6.11 Utilização da CPU x % Bloqueio: Plataforma AA

6.4 Análise Comparativa

Nesta fase da análise do comportamento do protocolo CMB são descritas as características obtidas através das comparações entre as simulações realizadas entre as diferentes arquiteturas. O objetivo é fornecer parâmetros para a realização da troca de protocolos de sincronização

através do comportamento das simulações considerando cada classe das arquiteturas representativas descritas no capítulo 4.

6.4.1 Influência do Desempenho dos Processadores e do Meio Físico em Relação ao Tempo de Simulação

Nesta seção são comparados os tempos de simulação considerando as características de cada plataforma utilizada. Para isso são comparados os tempos de simulação obtidos em relação à quantidade de ciclos que as tarefas devem realizar (para o modelo do servidor central) e em relação ao tempo entre chegadas (modelo sistema computacional simplificado).

As análises nesta etapa comparativa são realizadas separadamente para cada modelo de aplicação.

- **Modelo Servidor Central**

Para o modelo do servidor central, quando comparadas as plataformas constituídas por processadores com desempenhos distintos, o desempenho do meio físico não teve uma influência determinante no desempenho final das simulações (figura 6.12). Esse comportamento pode ser explicado quando a característica da granulosidade do modelo (grossa), aliada ao tamanho das mensagens utilizadas pela simulação são consideradas.

Foi verificado em [Ulson 1999] que as plataformas com meio de comunicação de baixo desempenho são adequadas para aplicações com granulosidade grossa. Desta forma a utilização de um modelo com granulosidade grossa, aliada a utilização de mensagens pequenas e o alto

desempenho atribuído aos processadores das plataformas AA e AB fizeram com que essas plataformas apresentassem tempos de simulação satisfatórios.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	➢ SP2	0	50	10000
SCS	➢ CRAY	10	100	30000
	➢ AA	20	250	50000
	➢ AB	30	➢ 500	➢ var.
	➢ BA	40	var.	
	➢ BB	➢ 50		
		60		
		70		
		80		
		90		
		100		
		var.		

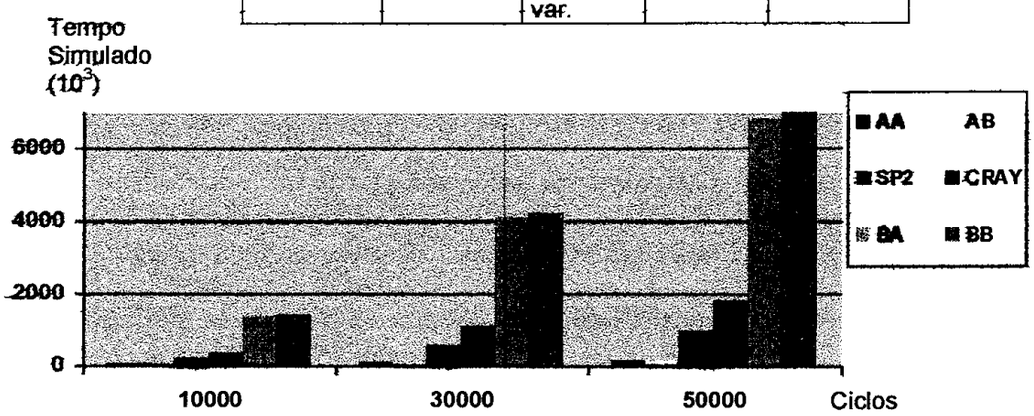


Figura 6.12 Gráfico Comparativo: Tempo Simulado x Ciclos

Entretanto, é evidente que mesmo nesse contexto, a utilização de um meio físico eficiente causa uma melhora bastante significativa no desempenho da simulação. Esse comportamento pode ser verificado quando plataformas com o mesmo processador mas com meios de comunicação com diferentes desempenhos são comparadas (Figura 6.13).

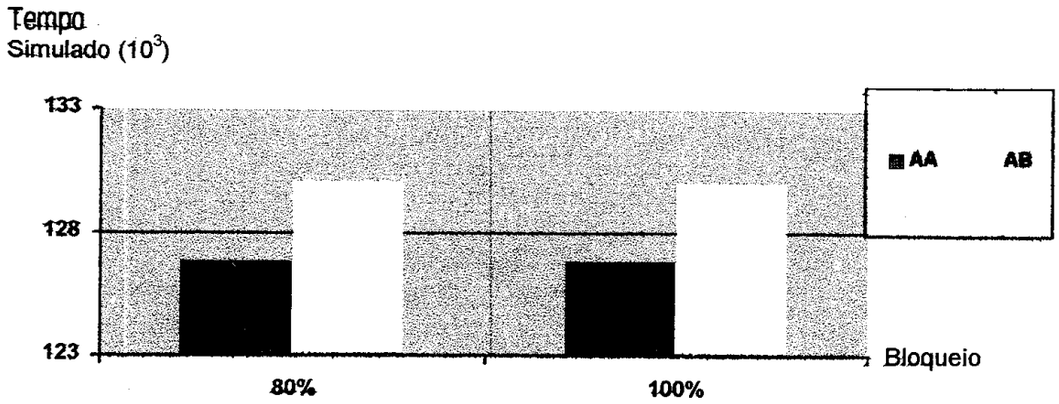


Figura 6.13 Arquiteturas com o mesmo processador e com meios de comunicação distintos.

A diferença entre os tempos do gráfico 6.13 é em média de 4.000 segundos e tende a aumentar quando a quantidade de mensagens nulas aumenta (a influência das mensagens nulas em conjunto das características das plataformas são discutidas na seção 6.4.2).

A diferença apresentada no gráfico 6.12 entre os tempos obtidos pelas plataformas AA e AB, em relação aos tempos obtidos pelas plataformas BA e BB, é uma consequência da grande diferença no desempenho dos processadores utilizados. Os processadores que compõem as plataformas AB e AA são, em média, cerca de 60 vezes mais velozes que os processadores das plataformas BA e BB.

A utilização de processadores com características de desempenho bastante distintas vem da necessidade de representação das arquiteturas disponíveis atualmente.

- **Modelo Sistema Computacional Simplificado**

O tempo de simulação, considerando plataformas com processadores e meios de comunicação diferentes, também foi determinado pela característica do desempenho dos processadores nos modelos abertos. Percebe-se novamente que as arquiteturas que possuem processadores com alto desempenho atingiram o final das simulações em um tempo menor (figura 6.14) e que o meio de comunicação determinou a diferença entre os tempos quando as plataformas utilizam processadores iguais.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	➢ SP2	0	➢ 1000	1.0
➢ SCS	➢ CRAY	10	2000	50.0
	➢ AA	20	var.	100.0
	➢ AB	30		➢ var.
	➢ BA	40		
	➢ BB	➢ 50		
		60		
		70		
		80		
		90		
		100		
		var.		

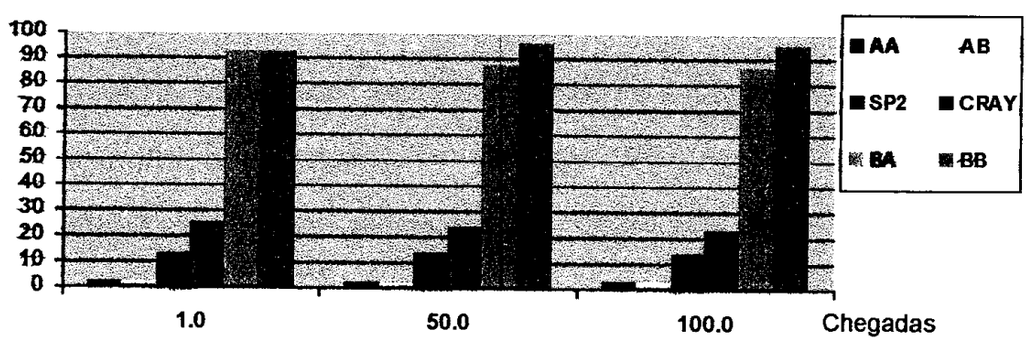


Figura 6.14 Gráfico Comparativo: Tempo Simulado x Ciclos

Percebe-se que a arquitetura BB, independente da probabilidade de bloqueio, apresenta tempos razoavelmente maiores aos apresentados pela arquitetura BA. Como a única características que diferencia essas simulações é o desempenho do meio físico, pode-se dizer que a influência do desempenho do meio físico para essas simulações é maior para os modelos abertos, uma vez que estes modelos, como foi discutido anteriormente, apresentam quantidades de mensagens nulas maiores.

6.4.2 Probabilidade de Bloqueio e Característica da Plataforma

- **Modelo Servidor Central**

Os gráficos das figuras 6.15 e 6.16 mostram o comportamento do tempo simulado em relação à probabilidade de bloqueio para 10000 e 50000 tarefas respectivamente. Nota-se inicialmente que os valores dos tempos de simulação entre os dois gráficos aumentam consideravelmente. Este aumento é esperado devido ao acréscimo na quantidade de ciclos que as tarefas devem realizar na simulação.

O comportamento das curvas observadas nos gráficos 6.15 e 6.16 já foi discutido na seção 6.3.2 quando foi considerada a influência das mensagens nulas no tempo de simulação para cada plataforma individualmente. A diferença dos tempos de simulação entre as diferentes plataformas também foi discutida na seção anterior (seção 6.4.1).

O aumento dos tempos de simulação quando a probabilidade de bloqueio tende a 50% foi observada em todas as plataformas consideradas

neste trabalho. A tabela 6.2 mostra o aumento, medido em porcentagem, em relação aos modelos 0% bloqueantes, dos valores do tempo de simulação obtidos para cada plataforma.

Tabela 6.2 Variações dos tempos de simulação em relação à probabilidade de bloqueio considerando todas as plataformas.

		% Bloqueio						
	Plataforma	**0**	**20**	**40**	**50**	**60**	**80**	**100**
% Aumento no tempo em relação ao modelo com 0% de bloqueio	AA	---	62	85	90	87	69	38
	AB	---	62	89	93	92	72	40
	Cray	---	61	84	89	86	68	37
	SP2	---	61	85	89	86	68	37
	BA	---	62	85	89	85	69	37
	BB	---	62	89	94	92	71	40

O que torna a análise desses gráficos (6.15 e 6.16) relevante é o aumento da concavidade das curvas quando comparados os valores de tempo de simulação entre 10000 e 50000 ciclos. Isto significa que quanto maior o número de ciclos que as tarefas devem realizar (período da simulação) maior é a diferença entre os valores satisfatórios (aqueles que foram obtidos com os intervalos 0-20 e 80 –100 % de bloqueio) e os valores insatisfatórios (intervalo 40 a 70 % de bloqueio).

Esse aumento da concavidade, observado em todas as plataformas analisadas, quando o período de simulação foi incrementado (de 1000 para 5000 ciclos) indica que o intervalo crítico (30 a 70% de bloqueio) apresentou um aumento considerável no total de mensagens bloqueadas, prejudicando o desempenho da simulação. Esse comportamento deve ser considerado normal, uma vez que aumentando o número de ciclos da simulação, aumenta-se a quantidade de tarefas transitando pelo modelo.

Configuração				
Modelo	**Plataf.**	**% bloq.**	**tarefas**	**ciclos**
➢ SC	➢ SP2	0	50	➢ 10000
SCS	➢ CRAY	10	100	30000
	➢ AA	20	250	50000
	➢ AB	30	➢ 500	var.
	➢ BA	40	Var.	
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

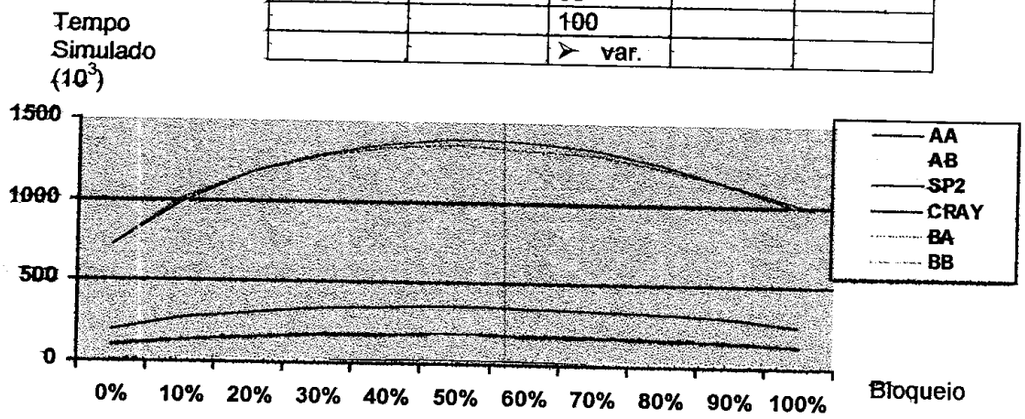


Figura 6.15 Tempo Simulado x % Bloqueio: 10000 Ciclos

O gráfico 6.17 mostra, considerando a mesma configuração do gráfico 6.15, o comportamento do tempo simulado para as arquiteturas AB e AA. Neste gráfico percebe-se que a influência do meio físico com baixo desempenho da plataforma AB fez com que os tempos para esta plataforma fossem maiores independente da probabilidade de bloqueio. Entretanto a diferença entre os tempos das duas plataformas torna-se mais acentuada no mesmo intervalo intermediário de probabilidade de bloqueio mencionado no parágrafo anterior (em torno de 4.000,00 segundos).

Esse aumento é justificado pelo aumento da quantidade de mensagens nulas fazendo com que a plataforma AB, que possui o meio de comunicação mais deficiente, fosse influenciada com a sobrecarga de comunicação.

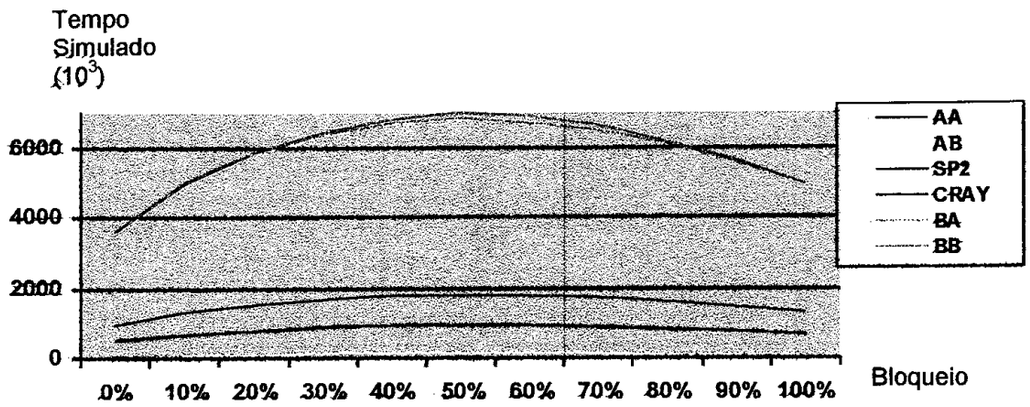


Figura 6.16 Tempo Simulado x % Bloqueio: 50000 Ciclos

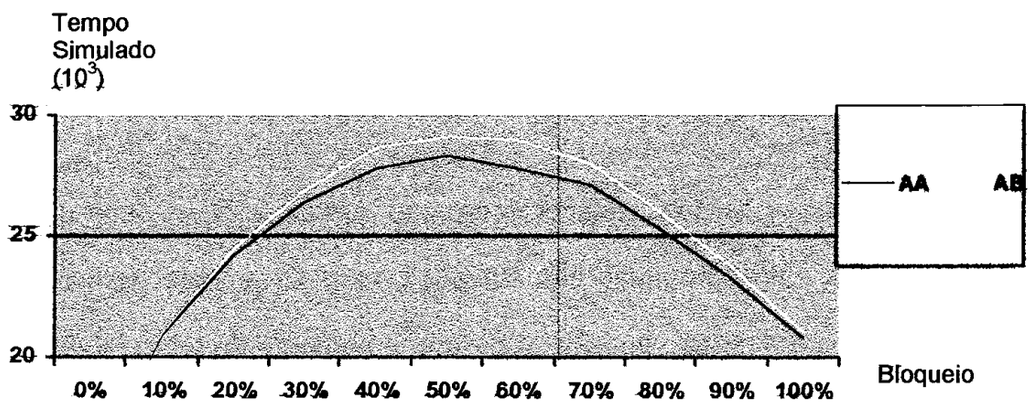


Figura 6.17 Tempo Simulado x % Bloqueio: Plataformas AA e AB

- **Modelo Sistema Computacional Simplificado**

O aumento da concavidade nas curvas para os valores dos tempos simulados também foi verificado no modelo do sistema computacional. Entretanto o fator que provocou este aumento foi a diferença de valores entre os tempo de chegadas (representada pelos gráficos 6.18 e 6.19).

Configuração				
Modelo	*Plataf.*	*% bloq.*	*entradas*	*chegadas*
SC	➢ SP2	0	1000	➢ 1.0
➢ SCS	➢ CRAY	10	➢ 2000	50.0
	➢ AA	20	var.	100.0
	➢ AB	30		var.
	➢ BA	40		
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

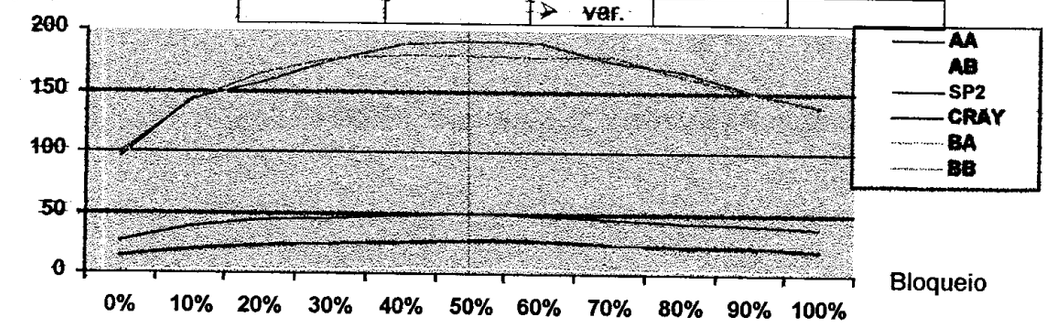


Figura 6.18 Tempo Simulado x % Bloqueio: 1.0 Tempo entre Chegadas

Com o aumento da concavidade, ocorrendo quando a quantidade de ciclos e o tempo entre chegadas são elevados, respectivamente para os modelos fechados e abertos, pode-se dizer que o aumento do tempo a ser simulado agrava as conseqüências observadas nos modelos com probabilidades de bloqueio próximas a 50%.

6.4.3 Relação entre a Probabilidade de Bloqueio e a Plataforma na Quantidade de Transmissões de Mensagens Nulas

Nesta seção é realizada uma análise a respeito da quantidade de mensagens nulas geradas por uma simulação a partir da característica

bloqueante do modelo simulado em diversas plataformas. Como foi mencionado anteriormente, a quantidade de mensagens nulas transmitidas entre os processos lógicos de uma simulação distribuída utilizando o protocolo CMB depende diretamente da probabilidade de uma mensagem completa ser ou não executada após atingir a primeira posição da lista de eventos futuros.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	➢ SP2	0	1000	1.0
➢ SCS	➢ CRAY	10	➢ 2000	➢ 50.0
	➢ AA	20	var.	100.0
	➢ AB	30		var.
	➢ BA	40		
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

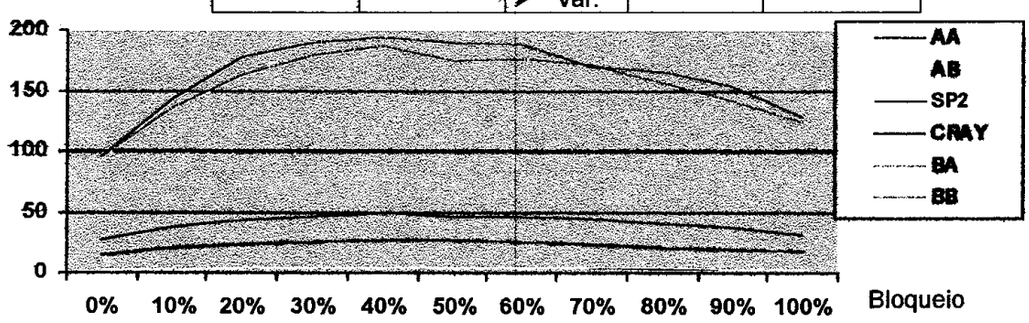


Figura 6.19 Tempo Simulado x % Bloqueio: 50.0 Tempo entre Chegadas

- **Modelo Servidor Central**

Os gráficos das figuras 6.20 e 6.21 fornecem as porcentagens de mensagens nulas transmitidas em relação ao número total de mensagens enviadas (completas e nulas) utilizando o modelo do servidor central variando a probabilidade de bloqueio.

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

Percebe-se através destes gráficos que o comportamento da porcentagem de mensagens nulas em função da probabilidade de bloqueio, independente da quantidade de ciclos (3000 ou 5000 ciclos) utilizada, é crescente tendendo a uma saturação, conforme foi discutido na seção 6.3.1 quando as plataformas foram analisadas separadamente.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	➢ SP2	0	50	10000
SCS	➢ CRAY	10	100	➢ 30000
	➢ AA	20	250	50000
	➢ AB	30	➢ 500	var.
	➢ BA	40	var.	
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

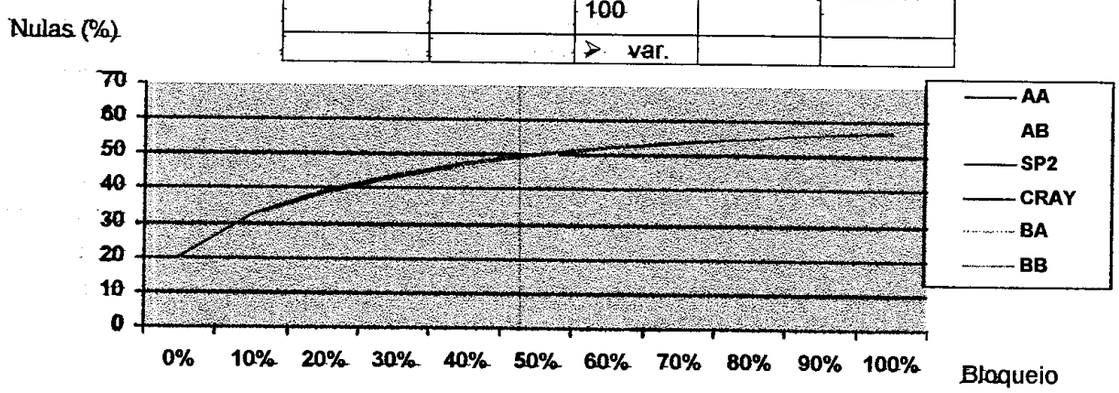


Figura 6.20 Mensagens Nulas em Relação à Probabilidade de Bloqueio: 3000 Ciclos

Observando os gráficos, nota-se primeiramente que, pela grande proximidade das curvas, a quantidade de mensagens nulas gerada independe da plataforma utilizada. Além disso, em nenhuma situação a quantidade de mensagens nulas ultrapassou 60% das mensagens que

circulavam pelo sistema independentemente da plataforma utilizada e da quantidade de ciclos necessários para o término da simulação. Isto significa que, mesmo nos modelos onde todas as tarefas eram bloqueadas quando atingissem a cabeça da lista de eventos futuros (100% bloqueio), não houve uma porcentagem maior que 60% de mensagens nulas para este modelo. A partir deste fato algumas questões podem surgir: Como pode existir uma quantidade maior de mensagens nulas transmitidas em relação às mensagens completas? Por que não houve em nenhum caso uma quantidade de mensagens nulas superior a 60%?

Configuração				
Modelo	***Plataf.***	***% bloq.***	***tarefas***	***ciclos***
➢ SC	➢ SP2	0	50	10000
SCS	➢ CRAY	10	100	30000
	➢ AA	20	250	➢ 50000
	➢ AB	30	➢ 500	var.
	➢ BA	40	var.	
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

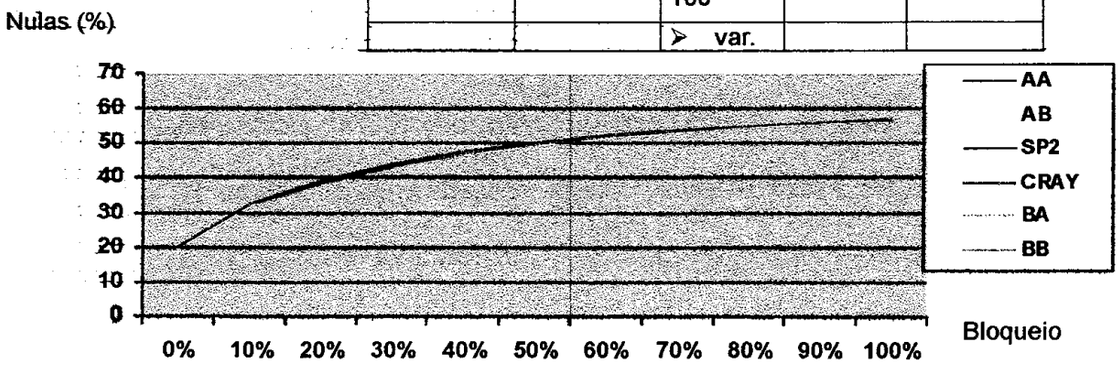


Figura 6.21 Mensagens Nulas em Relação à Probabilidade de Bloqueio: 5000 Ciclos

A resposta para a primeira questão pode ser facilmente formulada através da forma com que o modelo é implementado.

Foi observado, por exemplo, em uma simulação com número de tarefas igual a 500, com quantidade de ciclos igual a 10000 e probabilidade de bloqueio 100%, os valores de emissões de mensagens que estão ilustradas na figura 6.22.

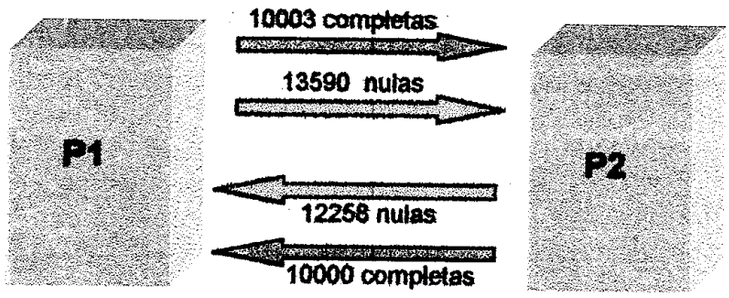


Figura 6.22 Emissões de mensagens nulas e completas. A quantidade de mensagens nulas é maior que a quantidade de mensagens completas devido às características do modelo.

Para que as tarefas executassem os 10000 ciclos exigidos foi necessário que essas tarefas fossem enviadas do processo P1 para o processo P2 e retransmitidas para P1. Considerando que a probabilidade de bloqueio é 100%, cada tarefa recebida pelos processos emitiu pelo menos uma mensagem nula. Isso gera um total de 10000 (P1 para P2) + 10000 (P2 para P1) mensagens completas transmitidas e 10000 + 10000 mensagens nulas transmitidas (sendo assim, 50% do total são mensagens nulas). Mas a quantidade de mensagens nulas excede esses valores (13590 e 12258 respectivamente). Essas mensagens nulas adicionais foram necessárias em

situações onde cada um dos processos ficaram com suas listas de eventos futuros vazias e, segundo a proposta original do CMB, quando um processo não possui eventos a serem processados estes processos devem enviar mensagens nulas indicando esta ausência de processamento.

Com relação à segunda questão, a quantidade de mensagens nulas não ultrapassa em nenhuma situação a razão de 60% por causa da quantidade de tarefas que transitaram pelo modelo. Nada impede, entretanto, que em um modelo fechado com um número de tarefas menor, ou um modelo aberto com baixa taxa de chegada de tarefas esse limite possa ser extrapolado.

Foi exatamente isso que ocorreu com o modelo do sistema computacional simplificado que será discutido a seguir.

- **Modelo Sistema Computacional Simplificado**

A quantidade de mensagens nulas observada neste modelo superou ligeiramente a quantidade de mensagens nulas geradas no modelo do servidor central. Esse aumento, em torno de 4% em média, é explicado pela diminuição das mensagens completas em consequência do alto tempo entre chegadas utilizado nas simulações. Com um número menor de mensagens completas circulando pelo sistema a probabilidade de um processo lógico ficar com sua lista de eventos futuros vazia é maior e essa situação é suficiente para a transmissão de novas mensagens nulas.

Outra característica importante observada nos gráficos analisados para o modelo do sistema computacional é a presença de um número maior de mensagens nulas para as arquiteturas AA e AB (figura 6.23). Nessas

arquiteturas com processadores de alto desempenho as tarefas tendem a se concentrar (em filas de espera) nos recursos que simulam o meio de comunicação, cujo desempenho em relação aos processadores são inferiores. Essa alta concentração de mensagens completas utilizando os meios de comunicação aumenta a probabilidade das listas de eventos futuros ficarem vazias.

Configuração				
Modelo	*Plataf.*	*% bloq.*	*entradas*	*chegadas*
SC	➢ SP2	0	➢ 1000	1.0
➢ SCS	➢ CRAY	10	2000	50.0
	➢ AA	20	var.	➢ 100.0
	➢ AB	30		var.
	➢ BA	40		
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		➢ var.		

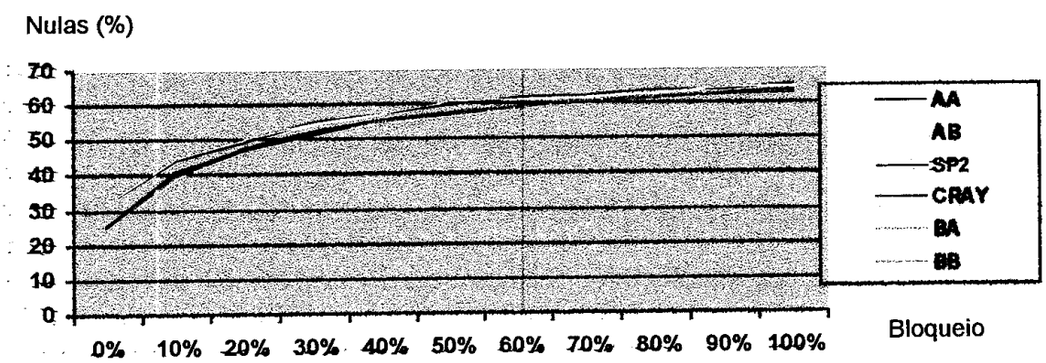


Figura 6.23 Emissões de mensagens nulas.

6.4.4 Análises Complementares sobre as Características do Comportamento do Protocolo CMB [2]

Análises complementares, além das realizadas nas seções anteriores, foram consideradas através da observação do comportamento do protocolo CMB sob diferentes variáveis e situações.

O tempo real de simulação, neste trabalho, não é um indicativo do comportamento do protocolo CMB em uma plataforma específica uma vez que os processos lógicos das simulações foram executados concorrentemente. Uma análise detalhada do comportamento do protocolo CMB em função dessa variável pode ser obtida em [Ulson 1999].

Foi verificado, como era esperado, um crescimento dos valores de tempo da simulação proporcional ao aumento do número de ciclos (servidor central) e tempo entre chegadas das tarefas (sistema computacional simplificado). Isto é explicado pela condição do término da simulação que é determinada por essas variáveis.

Além disto, os valores de tempo de simulação para as diferentes configurações utilizadas (plataforma, % bloqueio, etc.) , ao contrário dos valores de tempo real, não sofrem alterações consideráveis quando o número de tarefas é modificado. Isto ocorre por que os ciclos que devem ser realizados não estão diretamente relacionados à cada tarefa especificamente. De certa forma uma tarefa *i* pode realizar mais ciclos que uma tarefa *j*. A quantidade de ciclos para uma simulação não diz quantos

---

[2] As informações contidas nesta seção estão apresentadas detalhadamente no Apêndice C.

ciclos *cada* tarefa deva realizar e sim quantos ciclos *todas* as tarefas devem realizar.

A quantidade de código (eventos) a ser executado na simulação é o principal fator que determina o valor do tempo de execução da simulação. Quanto maior o número de mensagens nulas, maior será o número de eventos a serem executados e , portanto, maior o tempo de execução.

Os valores de utilização do recurso SMPL *cpu* obtidos, independente do modelo utilizado, foram muito próximos indicando que a modelagem das arquiteturas representativas e do protocolo CMB não alterou a simulação do modelo proposto. Ou seja, as mensagens nulas não tiveram participação nos resultados obtidos na simulação dos modelos do servidor central e sistema computacional, conforme as especificações da proposta original do protocolo de sincronização CMB.

6.5 Conclusões

A troca dinâmica de protocolos de sincronização envolve duas questões principais: *como* realizar a troca considerando suas características distintas e *quando* realizar a troca, isto é, determinar o melhor momento para a troca. O capítulo 4 mostrou *como* pode ser realizada a troca de protocolos considerando as diferentes características dos protocolos CMB e Time Warp através do mecanismo de troca.

A questão de *quando* converter de CMB para Time Warp é discutida nos capítulos 5 e 6 desta tese. O capítulo 5 mostrou como é implementado o sistema de simulação utilizado para realizar o estudo do comportamento do

protocolo CMB, frente a diferentes características do modelo de aplicação e das plataformas.

Neste capítulo, os resultados obtidos na realização das simulações foram analisados com o intuito de fornecer um panorama das situações em que o protocolo CMB apresenta medidas de desempenho desfavoráveis, sugerindo desta forma a troca de protocolo de sincronização (CMB para Time Warp).

Concluiu-se, através das análises, que a quantidade de mensagens nulas geradas durante a execução da simulação utilizando o protocolo CMB, depende principalmente da característica bloqueante do modelo de aplicação simulado.

Tabela 6.3 Situações Favoráveis à Troca de Protocolos.

Utilização de plataformas com processadores de baixo desempenho e modelos com alto tempo de serviço das mensagens completas;
Simulação de modelos abertos com alta taxa de chegadas de mensagens e plataforma com meio físico de baixo desempenho;
Modelos abertos com caraterísticas de bloqueio entre 30 e 60% independentemente da plataforma;
Modelos fechados com caraterísticas de bloqueio entre 30 e 80% independentemente da plataforma;
Simulações de modelos com mensagens maiores;

Para cada modelo simulado existe uma relação entre as mensagens nulas geradas e as mensagens completas bloqueadas cujo valor ideal

possibilita a atualização frequente dos *clocks* dos processos lógicos provocando o bom desenvolvimento da simulação.

Nessa situação, apesar das mensagens nulas significarem uma sobrecarga para a simulação, elas colaboram para garantir o avanço dos relógios dos processos lógicos. Assim, conforme o número de processos bloqueados aumenta, o aumento das mensagens nulas possibilita a liberação dos processos.

Assim, verificou-se que os piores desempenhos foram obtidos quando o modelo considerado apresentava probabilidades de bloqueio situados em uma faixa intermediária (próximas a 50% de bloqueio). A situação tornava-se ainda pior quando os modelos abertos eram considerados sugerindo fortemente a realização da troca de protocolos.

Através das características das mensagens utilizadas pelo protocolo CMB, o pequeno tamanho das mensagens nulas e completas minimizou a sobrecarga de comunicação tornando-a secundária no contexto geral relativo ao desempenho do protocolo CMB. Provavelmente, se as mensagens utilizadas pelo protocolo CMB fossem maiores, as arquiteturas com meio de comunicação de baixo desempenho apresentassem desempenho menos satisfatório.

Outra situação que mostrou-se favorável à troca de protocolo de sincronização refere-se à utilização de modelos de aplicação abertos com alta taxa de chegadas de mensagens. Nestas condições a grande quantidade de mensagens completas causou um desequilíbrio na relação mensagens nulas/mensagens bloqueadas promovendo uma queda no desempenho da simulações.

A tabela 6.3 mostra as principais condições, obtidas através da análise dos resultados, nas quais a simulação sincronizada pelo protocolo CMB apresentou baixo desempenho.

Essas condições, em adição aos resultados obtidos pela análise do comportamento do protocolo CMB através das técnicas de aferição em [Ulson 1999] são suficientes para determinar quando existirá uma possível troca de protocolos devido ao baixo desempenho oferecido pelo protocolo CMB nas condições discutidas anteriormente.

Capítulo 7

Conclusões Finais

Este capítulo apresenta as conclusões obtidas através da proposta do mecanismo de troca de protocolos em tempo de execução e do estudo do protocolo CMB através das técnicas de modelagem.

Através das conclusões obtidas, pela análise do comportamento do protocolo CMB, pôde-se definir parâmetros necessários para a identificação do momento da troca de protocolos.

As contribuições fornecidas por este trabalho e as propostas de trabalhos futuros também são apresentadas.

7.1 Introdução

Diferentes ferramentas podem ser utilizadas na avaliação de desempenho. Uma delas é a simulação, que constitui uma ferramenta poderosa utilizada em diversas áreas com finalidades diversificadas, devido principalmente à sua grande flexibilidade e ao baixo custo apresentado. Todavia uma simulação realista requer esforço computacional e desta forma o tempo de processamento necessário muitas vezes inviabiliza a simulação.

A utilização da simulação seqüencial, para modelos detalhados pode requerer horas ou mesmo dias de processamento computacional. Este problema é crítico quando estão sendo simulados sistemas computacionais.

Porém as técnicas de simulação têm evoluído. A natureza do mecanismo da lista de acontecimentos futuros imposta pela simulação seqüencial orientada a evento, onde a cada ciclo da simulação apenas um item é removido da lista de acontecimentos, dificulta a execução paralela do algoritmo em máquinas com mais de um elemento processador.

Entretanto a maneira tradicional e seqüencial tem dado lugar às técnicas de simulação distribuída que tentam explorar convenientemente o paralelismo . Em uma simulação distribuída ocorre o particionamento da lista de eventos futuros e desta forma torna-se necessário um protocolo de sincronização.

A questão relativa aos méritos dos protocolos conservativos e otimistas tem sido freqüentemente abordada. Entretanto, existe um consenso geral de que a formulação de regras para determinar a superioridade de um protocolo em relação a outro é uma tarefa bastante difícil.

Alguns protocolos que assumem uma política mista entre os protocolos otimistas e conservativos têm sido propostos. Entretanto, é difícil para o modelador da simulação, que não é intimamente familiarizado com os protocolos de sincronização e com as arquiteturas paralelas, ajustar os parâmetros apropriados para obter um bom desempenho.

O objetivo deste trabalho é fornecer condições ao modelador da simulação distribuída para alterar, durante a execução da simulação e através de um mecanismo que reuna informações a respeito do desempenho da simulação, o protocolo utilizado para a sincronização da simulação distribuída.

Com o auxílio desse mecanismo de troca de protocolos pretende-se auxiliar o usuário a desenvolver aplicações eficientes utilizando o paradigma da simulação distribuída, sem exigir desse usuário o conhecimento das características e particularidades do protocolo de sincronização utilizado e com conhecimento apenas de conceitos básicos de programação paralela.

A revisão apresentada nos capítulos 2 e 3 baseia-se em uma vasta bibliografia enfatizando os conceitos fundamentais dentro do paradigma de simulação distribuída e de computação paralela. Neles abordam-se conceitos que vão desde as limitações da simulação seqüencial e definições de processo no contexto da simulação até a descrição dos principais protocolos pertencentes às classes Otimistas e Conservativos desenvolvidos a partir da década de 80 até os dias de hoje.

O esquema de troca de mensagens para sincronizar uma simulação distribuída utilizado pelos protocolos CMB e Time Warp foram abordados no

capítulo 4. Neste capítulo foi respondida a pergunta relacionada à troca de protocolos de sincronização: "Como realizar a troca?".

Para cada protocolo considerado neste trabalho (CMB e Time Warp) existe um conjunto de recursos e características (estruturas de dados, variáveis de estado, variáveis estatísticas e estruturas de interface de comunicação) que devem ser consideradas no momento da troca de protocolos. O capítulo 4, a partir da seção 4.2, discute a proposta do mecanismo de troca de protocolos. Para isso foi proposto um mecanismo que possibilite a troca considerando as diferentes estruturas que compõem os protocolos CMB e Time Warp.

Foi considerado que uma das características principais com relação a implementação dos protocolos de sincronização refere-se às estruturas de dados que auxiliam no gerenciamento das mensagens utilizadas na simulação. Para cada protocolo de sincronização existe um conjunto de variáveis e estruturas de dados diferentes, isto é, que fazem parte de um determinado protocolo mas que não possuem similares em outro protocolo.

A troca de protocolos durante a execução da simulação exige que estruturas que não existiam em um determinado instante em um processo lógico CMB, por exemplo, passem a fazer parte do mesmo processo lógico Time Warp e vice e versa. De um forma geral foi utilizado um mecanismo complementar de utilização de estruturas de dados. Isto é, a maior parte das estruturas de dados foram reunidas em apenas um conjunto de estruturas. Desta forma um processo Time Warp, por exemplo, abrangeria além de suas estruturas de dados originais, as estruturas de dados de um processo lógico

CMB e essas estruturas eram utilizadas conforme o protocolo que estaria regendo a simulação.

A forma com que é feita a troca de protocolos de sincronização não foi o único enfoque deste trabalho. Além disto foi realizada uma análise do comportamento do protocolo CMB para determinar qual o melhor momento em que a troca de protocolos contribui para o aumento do desempenho da simulação. Além do protocolo de sincronização existem vários fatores que influenciam no comportamento de uma simulação distribuída.

O capítulo 5 descreve o modelo de simulação construído, utilizando a extensão funcional SMPL, para analisar o protocolo CMB. Neste modelo são consideradas as características da plataforma do hardware utilizado para a execução, do protocolo de sincronização CMB e finalmente da aplicação utilizada como estudo de caso para a simulação.

Características como quantidade de tarefas, probabilidade de bloqueio e fluxo de mensagens do modelo foram introduzidas para que, através dos valores obtidos nos relatórios de cada simulação, o desempenho da simulação fosse analisado através de medidas significativas como quantidade de mensagens nulas geradas e tempo de simulação.

No capítulo 6 foi realizada uma análise completa do desempenho da simulação dos modelos propostos considerando os diferentes itens introduzidos através da estratégia de execuções.

7.2 Conclusões Finais

Uma vez que existe um duplo enfoque do trabalho desenvolvido, optou-se por dividir em duas seções a relação das principais conclusões obtidas.

7.2.1 Mecanismo de Troca de Protocolos

Com relação ao mecanismo de troca de protocolos de sincronização proposto neste trabalho, pode-se obter algumas conclusões que são descritas a seguir.

- Uma vez que a viabilidade lógica da troca de protocolos foi demonstrada por JHA e BAGRODIA em [Jha 1994] pôde-se, através da descrição do mecanismo de troca proposto, verificar a viabilidade prática da troca considerando cada uma das estruturas e variáveis envolvidas nos dois protocolos de sincronização considerados neste trabalho. Desta forma, pode-se afirmar que é possível construir um sistema de simulação no qual o protocolo de sincronização seja ajustado, automaticamente, de acordo com o conjunto de características que envolve o contexto de uma simulação distribuída.

- A união das estruturas de dados dos protocolos CMB e Time Warp inserida em um mesmo processo lógico da simulação não interfere no desempenho da simulação uma vez que, após a troca, as estruturas adicionais permanecerão inoperantes;

- Para que a troca de protocolos ocorra efetivamente, deve ser constatado um baixo desempenho da simulação que justifique o tempo gasto na realização da troca causado pela sobrecarga adicional no processamento das operações entre as estruturas dos protocolos. Entretanto, com o surgimento de sistemas computacionais cada vez mais rápidos essa sobrecarga terá uma menor influência nas tomadas de decisões do mecanismo de troca de protocolos;

7.2.2 Estudo do Comportamento do Protocolo CMB para Realização da Troca de Protocolos

Com a implementação do modelo de simulação realizada neste trabalho pôde-se analisar o comportamento do protocolo CMB sob diferentes contextos. Características como fluxo de tarefas do modelo de aplicação, probabilidade de bloqueio e quantidade de tarefas e desempenho das plataformas foram introduzidas em um único modelo. Da análise dos resultados podemos concluir:

- **Influência das Mensagens Nulas**

A influência das mensagens nulas em uma simulação distribuída tem sido foco de intensa discussão durante os últimos anos. A possibilidade de uma degradação no desempenho de simulações devido a altas taxas de

mensagens nulas representa uma grande desvantagem dos métodos baseados em mensagens nulas.

Neste trabalho a influência das mensagens nulas no desempenho da simulação foi avaliada através do valor do tempo de simulação (variável tempo simulado) uma vez que as mensagens nulas concorrem com as mensagens completas utilizando os mesmos recursos da simulação e não provocam diretamente um avanço no *clock* dos processos lógicos.

Pelas análises, pode-se concluir que, a quantidade de mensagens nulas nem sempre é proporcional ao desempenho da simulação. Essa relação é influenciada por outros fatores e as simulações com piores desempenhos são aquelas cujos modelos apresentavam probabilidades de bloqueio com valores intermediários (entre 30 e 70 %) independentemente da plataforma e do modelo utilizado. Dentro deste intervalo os piores desempenhos foram observados nos modelos com probabilidade de bloqueio de 50%.

Desta forma, conclui-se que os modelos com características intermediárias de bloqueio são aqueles onde o método de sincronização CMB, nas condições apresentadas para esta simulação, apresentam um baixo desempenho. Esta constatação contraria a idéia de que a simulação de um modelo com grande quantidade de emissões de mensagens nulas necessariamente apresenta um baixo desempenho.

Este comportamento foi explicado (capítulo 6) através da maneira com que o método CMB é implementado. Segundo as diretrizes originais do CMB toda mensagem que não for executada imediatamente após atingir a cabeça da lista de eventos futuros deverá provocar a emissão de uma

mensagem nula aos outros processos indicando esta "ausência momentânea" de processamento de eventos. Logo o número de mensagens nulas em trânsito no sistema é proporcional à característica bloqueante do modelo.

Consequentemente, quando a relação mensagens nulas/tarefas bloqueadas é insuficiente para promover um avanço com maior freqüência nos tempos dos processos o desempenho da simulação diminui elevando o tempo total de simulação.

Isto não significa que as mensagens nulas nunca influenciam negativamente no desempenho da simulação, pois o comportamento decrescente após a marca de 50% indica que a quantidade de mensagens nulas trocadas entre os processos está sendo suficiente para avançar os *clocks* dos processos mas mesmo assim a sobrecarga causada pelo aumento destas mensagens não permite que o tempo de simulação seja tão satisfatório quanto aquele obtido na condição ideal (0% de bloqueio).

Apesar deste comportamento ter sido verificado em todas as simulações independentemente da plataforma, a influência negativa das mensagens nulas é maior nas simulações realizadas em plataformas com meios de comunicação de baixa velocidade.

Esta conclusão está baseada na observação dos tempos de simulação expressos através das concavidades representadas nos gráficos. Estas concavidades expressam valores insatisfatórios para a relação mensagens completas/ mensagens nulas. Com o aumento da concavidade ocorrendo quando a quantidade de ciclos e o tempo entre chegadas são elevados, para os modelos fechado e aberto respectivamente, pode-se concluir que esses

dois fatores contribuem diretamente com o valor da relação mensagens completas/mensagens nulas.

- **Influência da Característica Bloqueante da Aplicação**

Através da modelagem da característica de bloqueio de um processo, que indica a probabilidade de suas mensagens ficarem esperando para serem processadas, pode-se concluir que até mesmo para os modelos 0% bloqueantes podem ocorrer situações na qual um processo lógico fique com sua lista de ventos futuros *vazia*. Nessas circunstâncias uma mensagem nula deve ser enviada pelo processo que contém esta lista para indicar essa ausência de processamento. Desta forma, mesmo com um modelo 0%-bloqueante os processos devem enviar mensagens nulas.

Por outro lado, conclui-se pelos resultados que, quando existe um aumento da probabilidade de bloqueio do modelo, a quantidade de mensagens nulas em relação ao número total de mensagens aumenta proporcionalmente. Esta característica pode ser explicada considerando-se que sempre que uma tarefa (mensagem completa) é bloqueada, seguindo os critérios do método CMB em sua versão original, esta mensagem deve emitir para os outros processos uma mensagem nula indicando que seu processo lógico não processará nenhum evento. Portanto quanto maior a quantidade de mensagens bloqueadas, maior a quantidade de mensagens nulas enviadas.

Este comportamento reforça a conclusão do item anterior que confirma a não interferência direta e exclusiva das mensagens nulas no desempenho da simulação.

Através dos gráficos de probabilidade de bloqueio e quantidade de mensagens nulas conclui-se que a quantidade de mensagens nulas transmitidas durante uma simulação é uma consequência da natureza bloqueante do modelo simulado. Os modelos altamente bloqueantes necessitam de uma quantidade maior de mensagens nulas para satisfazer as mensagens bloqueadas. Logo, não se pode considerar o desempenho do método CMB apenas pela quantidade de mensagens nulas que está sendo gerada.

Este mesmo comportamento foi observado para diferentes quantidades de tarefas no modelo do servidor central.

Além disso pode-se concluir, através das simulações realizadas, que para as classes de modelos estudados o valor percentual de mensagens nulas nunca ultrapassa a quantidade de mensagens completas circulando pelo sistema. Isso ocorre porque as mensagens completas são as responsáveis pelas transmissões das mensagens nulas.

Para o modelo do servidor central a quantidade de mensagens nulas não ultrapassou a marca de 60% mesmo com o modelo 100% bloqueante. Em [Ulson 1999], onde a mesma configuração foi utilizada em uma plataforma SP2, este comportamento também foi observado por aferição.

Outra questão refere-se a indiferença na porcentagem de mensagens nulas quando a quantidade de ciclos que as mensagens devem realizar na simulação é alterada. A porcentagem de mensagens nulas geradas em um

simulação com um número alto de ciclos é praticamente a mesma observada em uma simulação com poucos ciclos.

A quantidade de mensagens nulas observada nos modelos abertos superou a quantidade de mensagens nulas geradas no modelo do servidor central. Esse aumento, em torno de 4% em média (que equivale a aproximadamente 300 mensagens) ocorreu por causa da escassez de mensagens completas em consequência do alto tempo entre chegadas utilizado nas simulações dos modelos abertos. Com um número menor de mensagens completas circulando pelo sistema a probabilidade de um processo lógico ficar com sua lista de eventos futuros vazia ou suas mensagens bloqueadas é maior e essas situações são suficientes para a transmissão de novas mensagens nulas.

- **Influência da Plataforma**

As implementações da simulação distribuída para os modelos desenvolvidos neste trabalho apresentaram melhor desempenho quando executadas nas plataforma AA e AB, seguidas pelas plataformas SP2 IBM e Cray T3D. As características de desempenho dos elementos de processamento e do meio de comunicação influenciaram diretamente no desempenho do modelo de simulação distribuída .

Pode-se concluir que, para os modelos estudados, o desempenho da simulação depende de uma combinação (entre processador e meio físico) onde a velocidade do meio de transmissão exerce uma maior influência.

Para os modelos com baixa porcentagem de mensagens nulas a diferença entre os tempos de simulação é menor, entretanto, quando essa porcentagem aumenta torna-se evidente que as plataformas com meio de comunicação de baixa velocidade são prejudicadas pelo alta quantidade de mensagens em trânsito.

Verificou-se, através da observação dos tempos da plataforma AB, que um bom processador compensa, para os modelos estudados, essa perda no desempenho da simulação causado pelo baixo rendimento do meio de comunicação.

Além da degradação do desempenho da simulação causado pelo meio de comunicação de baixo desempenho foi observado nas plataformas com meio de comunicação de baixa velocidade uma quantidade de mensagens nulas sensivelmente superior em relação às plataformas com meio de comunicação de alto desempenho (aproximadamente 0.5%) nos modelos com valores de probabilidade de bloqueio até 80%.

Isto ocorreu por que nos modelos com valores de probabilidade de bloqueio até 80% o tempo gasto pelas mensagens completas utilizarem o meio físico proporcionou uma quantidade de bloqueios maior. Toda vez que uma mensagem bloqueia o processo, e consequentemente uma nula é transmitida, quase imediatamente após chega uma mensagem nula e desbloqueava o processo. Essa atividade bloquear/desbloquear é maior, para as plataformas AB e BB, devido ao tempo existente entre as chegadas das mensagens completas causado pelo baixo desempenho dos meios físicos utilizados para estas plataformas.

- **Influência do Fluxo da Tarefas (Modelos Abertos x Modelos Fechados)**

Outra conclusão obtida refere-se à influência da instabilidade dos modelos abertos provocando o aumento da quantidade média de mensagens nulas no sistema. A possibilidade de não existir nenhuma mensagem completa (tarefa) circulando pelo sistema durante um determinado ciclo da simulação, nesses modelos, determinou as transmissões de mensagens nulas adicionais.

Por outro lado, o valor máximo de mensagens nulas alcançado nos modelos abertos também ultrapassou ligeiramente o valor máximo atingido nos modelos do servidor central. Essa pequena elevação na porcentagem de mensagens nulas foi ocasionada pela alta taxa de chegada de mensagens completas atribuída nesses modelos.

Essa ocorrência leva a concluir que, para os modelos abertos, o valor ideal para a relação mensagens completas/mensagens nulas é necessariamente maior que o valor ideal apresentado pelo modelo fechado.

Entretanto, mesmo para modelos abertos onde o tempo necessário para a simulação atingir estabilidade é maior, as mensagens nulas não influenciam diretamente no desempenho da simulação.

Os números indicativos da utilização de recursos para os modelos abertos mostram que em simulações de modelos desta classe a sub utilização dos recursos da simulação pode prejudicar o desempenho final.

Com relação a influência do desempenho do meio físico para diferentes classes de modelos, discutiu-se no capítulo 6 que a arquitetura BB,

independente da probabilidade de bloqueio, apresenta tempos consideravelmente superiores aos apresentados pela arquitetura BA nos modelos abertos. Desta forma, pode-se concluir que a influência do desempenho do meio físico nas simulações realizadas neste trabalho é maior nos modelos abertos.

Com relação a utilização dos recursos da simulação, apesar das quantidades de mensagens nulas apresentarem diferentes valores para cada arquitetura, os valores de utilização do recursos SMPL, independente do modelo utilizado, foram muito próximos. Este comportamento era esperado e indica que a quantidade de mensagens nulas não influencia na utilização dos recursos envolvidos na simulação.

- **Influência do Período de Simulação e da Quantidade de Tarefas**

O período no qual uma simulação é realizada, determinado pela condição de parada de uma estrutura de repetição que controla a simulação, deve ser determinado com certo cuidado.

Pôde-se notar que, mesmo nos modelos fechados onde a estabilidade da simulação é atingida mais rapidamente, existe um período inicial onde o comportamento do protocolo CMB ainda se mostra distante do observado quando uma certa quantidade de tempo já tenha transcorrido.

Após esse período inicial crítico, percebe-se claramente que o modelo atinge uma certa estabilidade. Essa estabilidade pode ser percebida através das variáveis estatísticas do modelo de aplicação e de desempenho da simulação. A porcentagem de mensagens nulas, por exemplo, tende a se

manter fixa independente da quantidade de ciclos que a simulação execute posteriormente ao período inicial. Por maior que seja a quantidade de tarefas (mensagens completas) que cheguem ao sistema o número de mensagens nulas em proporção a essa quantidade de tarefas é o mesmo.

Um fator determinante no período de simulação refere-se à quantidade de tarefas inseridas no modelo de simulação. Independentemente das características do modelo de aplicação sabe-se que, em uma simulação distribuída, as variáveis de tempo que controlam cada um dos processos lógicos são atualizadas, na grande maioria das vezes, quando uma tarefa é processada. Desta forma, a quantidade de tarefas inseridas da simulação constitui um fator determinante do tempo de realização da simulação.

- **Viabilidade das Técnicas de Modelagem no Estudo do Paradigma da Simulação Distribuída Conservativa**

O paradigma da simulação distribuída possibilita o desenvolvimento de aplicações eficientes, explorando os benefícios oferecidos pela computação paralela.

A análise minuciosa dos resultados obtidos demonstra que o desempenho da simulação distribuída é afetado diretamente pelas características da plataforma utilizada e pode ser agravado pelas características do modelo desenvolvido.

A utilização das técnicas de modelagem através da implementação do protocolo *CMB* com a extensão funcional *SMPL* demonstra a viabilidade da utilização de extensões funcionais e ambientes de passagem de mensagens para o desenvolvimento da simulação distribuída conservativa.

Através da validação, feita por comparação dos valores estatísticos obtidos através dos resultados da análise do comportamento utilizando técnicas de aferição em [Ulson 1999] , pode-se concluir que a modelagem apresenta resultados confiáveis para a análise de desempenho de simulações distribuídas.

7.3 Contribuições

As principais contribuições deste trabalho são apresentadas a seguir.

1. Caracterização do Mecanismo de Troca de Protocolos

O processo de construção de um mecanismo de troca de protocolos em tempo de execução, que até então possuía apenas definições teóricas, passa a ter com este trabalho os parâmetros para sua implementação. Definiu-se neste trabalho como deve ser a organização do mecanismo de troca, através da criação de processos que cuidarão da constante monitorização da simulação (processo observador), da detecção de baixo desempenho e portanto da necessidade de troca de protocolo (processo gerenciador) e do processo que possibilita a compatibilização das estruturas que compõem os mecanismo de sincronização CMB e Time Warp (processo conversor).

2. Metodologia para Avaliação de Desempenho de Protocolos de Sincronização

A possibilidade de analisar o desenvolvimento de uma simulação distribuída utilizando-se um computador seqüencial constitui uma das várias opções que as técnicas de modelagem oferecem para este tipo trabalho.

Este trabalho apresenta uma técnica para a avaliação de protocolos de sincronização, oferecendo a modelagem da arquitetura e definindo como o protocolo de sincronização e a aplicação devem ser representados através dos eventos da simulação seqüencial.

A construção de modelos que simulem desde a plataforma até o próprio modelo de aplicação pode ser estendida a uma grande quantidade de estudos de casos. Apesar deste trabalho considerar alguns modelos clássicos e algumas plataformas representativas, a técnica desenvolvida e apresentada neste trabalho pode ser facilmente utilizada na construção de modelos específicos que venham de encontro a necessidade do modelador.

3. Aplicação da Metodologia Desenvolvida para Outros Protocolos

A análise de desempenho do protocolo conservativo CMB, apresentado neste trabalho, constitui um estudo de caso que pode ser utilizado para a

análise do comportamento de outros protocolos de sincronização (conservativos, otimistas ou mistos).

Esse atividade é possível devido à modularidade da simulação, implementada neste trabalho, que permite a substituição dos eventos necessários para a modelagem do protocolo em questão.

4. Estudo do Comportamento da Simulação Distribuída

O trabalho possibilita a previsão da tendência do comportamento da simulação distribuída conservativa em função das características arquiteturais da plataforma utilizada, das características de bloqueio dos modelos, entre outros fatores. Considerando essas características foi possível estabelecer um método para analisar o comportamento da simulação distribuída. Utilizando esse método e tendo conhecimento das características da plataforma utilizada, pode-se verificar a tendência do comportamento da simulação distribuída conservativa independente do protocolo de sincronização utilizado.

Desta forma, é possível para o usuário verificar a relação custo/benefício da utilização do paradigma distribuído conservativo e determinar se a utilização desse paradigma é vantajosa.

7.4 Propostas para Trabalhos Futuros

Esta seção traz uma relação das propostas de trabalhos futuros.

1. Implementação do Mecanismo de Troca de Protocolo

Este trabalho serve como base para a implementação do mecanismo de troca de protocolos oferecendo toda a organização que deve ser seguida para a troca. A implementação deste mecanismo permitirá uma avaliação mais detalhada de suas vantagens e desvantagens. Considerando a troca entre os protocolos CMB e Time Warp, a questão de quando deve ser realizada é também analisada neste trabalho. Assim, já é possível iniciar a construção do mecanismo de troca considerando a troca do protocolo CMB pelo protocolo Time Warp. Através da análise do comportamento do protocolo Time Warp, em desenvolvimento como tese de doutorado pela aluna Renata Spolon [Spolon 2000], o mecanismo de troca de protocolos poderá ser implementado na íntegra.

2. Construção da Interface Gráfica do Mecanismo de Troca

A implementação do mecanismo de troca de protocolos envolve, como trabalho a parte, a construção de uma interface gráfica que possibilite a interatividade do modelador com o processo de troca de protocolos. A implementação da interface gráfica deverá prover o sistema de troca de protocolos de gráficos de monitoramento e variáveis de desempenho que serão fornecidas durante o desenvolvimento da simulação.

3. Estudo do Comportamento de Protocolos Variantes do Time Warp

A possibilidade de expandir a número de opções de escolha do protocolo que sincronizará a simulação, a partir de um determinado momento, torna o estudo do comportamento de outros protocolos bastante atrativo.

A maioria dos protocolos descendentes do Time Warp mantém uma lista de estruturas de manipulação de mensagens e variáveis de tempo similar ao protocolo original. A detecção de determinadas situações onde esses protocolos acrescentariam um maior desempenho no andamento da simulação, possibilitaria a inclusão destes protocolos no mecanismo de troca automática.

4. Estudo da Obtenção da Consistência para a Troca de Protocolos na Simulação Time Warp

A realização intencional dos *rollbacks* para todos os processos lógicos até o valor do GVT para garantir o estado consistente e permitir a realização da troca de Time Warp para CMB foi considerada a mais adequada para a realização da troca de protocolos Time Warp para CMB.

Outra alternativa, consistia em avançar os LVT's dos processos lógicos até a marca do maior LVT.

Este trabalho apresenta uma análise dos prós e contras de cada uma dessas alternativas. Quantificar esta comparação através de simulações ou protótipos é uma tarefa que deverá ser abordada em um trabalho futuro.

APÊNDICE A

ANÁLISES ESTATÍSTICAS

A.1 Introdução

A inferência estatística tem como objetivo tirar conclusões sobre parâmetros populacionais a partir de análise dos dados amostrais. Alguns pontos importantes são o tamanho amostral e o modo de selecionar a amostra, a natureza da inferência desejada e a precisão das conclusões. A seção A.2 mostra a realização dos testes de hipóteses utilizados no trabalho.

A.2 Teste de Hipóteses para Amostras Pequenas

O teste de hipóteses realizado tem por objetivo verificar se as diferenças observadas para os tempos de execução (tempo simulado) dos modelos de simulação analisados são estatisticamente significativas.

Foram definidas hipóteses para a comparação dos dados obtidos, e a partir da aceitação ou rejeição dessas hipóteses, pode-se concluir sobre a validade de uma determinada afirmação.

H_0 é chamada hipótese de nulidade, e H_1 é a hipótese alternativa. Para provar qual hipótese é correta, tenta-se provar inicialmente a hipótese nulidade. Se não é possível provar a hipótese de nulidade, a hipótese alternativa é considerada verdadeira (para manter uniformidade, o resultado final é dado em termos da hipótese H_0, ou seja, aceitar ou rejeitar H_0).

Neste trabalho foram considerados os valores para o tempo de simulação e porcentagem de mensagens nulas obtidas nas seis plataformas representativas para o modelo do servidor central.

As seções A.3 e A.4 mostram as análises estatísticas realizadas para cada um desses parâmetros.

A.3 Tempo Simulado

H_0: Não existe efeito de tratamento entre os tempos obtidos para cada plataforma, ou $\tau_1 = \tau_2 = ... = \tau_6 = 0$

H_1: Existe $\tau_i \neq 0$; $i = 1, ..., 6$

Tabela A.1 Análise de Variância para o tempo simulado.

Causas de Variação	GL	SQ	QM	F
Tratamentos	5	$1{,}79387 * 10^{15}$	$3{,}58775 * 10^{14}$	248,54 ($p < 0{,}0001$)
Resíduo	786	$1{,}13463 * 10^{15}$	$1{,}44355 * 10^{12}$	
Total	791	$2{,}92850 * 10^{15}$		

$F_{(0{,}05;\ 5;\ 786)} = 2{,}23$

No nível de 5% de significância pode ser concluído que *há diferença* entre os tempos médios simulados das arquiteturas em estudo.

Desta forma, quando se tem mais de dois níveis do fator em estudo, a técnica paramétrica mais usual para avaliar as diferentes respostas médias dos níveis é a da análise da variância, complementada com os testes de comparações múltiplas.

Se os níveis do fator são quantitativos deve-se utilizar a análise de regressão (se possível) para inferir em todo o intervalo considerado. Quando os níveis são qualitativos deve-se optar por testes de comparações múltiplas. No caso de pares de médias o mais usual é o Teste de Tukey, e no caso de mais de duas médias, o de Scheffé.

Para identificar quais plataformas apresentam diferença entre as médias de tempo, prossegue-se a análise estatística com o teste de comparações de médias de Tukey.

A.3.1 Teste de Tukey

$\mathbf{H_0}$: As médias de tempo de simulação são iguais, ou $\mu_1 = \mu_2 = \ldots = \mu_6$

$\mathbf{H_1}$: Existe $\mu_i \neq \mu_{i'}$, $i = 1, \ldots, 6$ $\quad i' = 1, \ldots, 6$ $\quad i \neq i'$

$DHS = q * (QMRes/r)^{1/2}$

Onde:

q é o valor tabelado da amplitude total *estudentizada*, $(q_{\alpha, k, \varphi})$ α = nível de significância, k = número de tratamentos e φ = graus de liberdade do Resíduo.

QMRes = Quadrado Médio Residual;

r = repetições;

$(q_{0,05, 6, 786}) = 4,03$; QMRes = $1,44355*10^{12}$; r = 132;

$DHS = 4,03 * (1,44355*10^{12}/132) = 421437,9127$;

A tabela A.2 mostra os valores absolutos das diferenças entre as médias de tempo das máquinas AA, AB, BA, BB, CRAY e SP2.

Tabela A.2 Pares de Médias e Conclusões

Pares de médias	Valor absoluto da diferença	Conclusão
AA – AB	1 329,27	Não se diferem
AA – BA	3 425 848,79	AA ≠ BA
AA - BB	3 472 966,47	AA ≠ BB
AA – CRAY	846 563,90	AA ≠ CRAY
AA – SP2	412 163,77	Não se diferem
AB – BA	3 424 519,51	AB ≠ BA
AB - BB	3 471 637,20	AB ≠ BB
AB - CRAY	845 234,52	AB ≠ CRAY
AB – SP2	410 834,49	Não se diferem
BA - BB	47 117,69	Não se diferem
BA - CRAY	2 579 285,00	BA ≠ CRAY
BA – SP2	3 013 685,02	BA ≠ SP2
BB - CRAY	2 626 402,68	BB ≠ CRAY
BB – SP2	3 060 802,71	BB ≠ SP2
CRAY – SP2	434 400,02	CRAY ≠ SP2

Tabela A.3 Média, desvio-padrão e resultados do teste estatístico do tempo simulado.

AA	AB	BA	BB	CRAY	SP2
72827,81 ± 42015,99	74157,08 ± 42916,48	3498676,59 ± 2019336,95	3545794,28 ± 2053998,52	919391,60 ± 531674,89	484991,58 ± 279941,59
A (1)	A	B	B	C	A

DHS(5%) = 421437,91

(1) Médias seguidas de mesma letra não diferem entre si.

A.4 Mensagens Nulas

Para a análise das porcentagens de mensagens nulas formulou-se as seguintes hipóteses:

H_0: Não existe efeito de tratamento entre as plataformas para a porcentagem de mensagens nulas, ou $\tau_1 = \tau_2 = ... = \tau_6 = 0$

H_1: Existe $\tau_i \neq 0$ $\quad i = 1, ..., 6$

Tabela A.4 Análise de Variância para a porcentagem de mensagens nulas.

Causas de variação	GL	SQ	QM	F
Tratamentos	5	0,000417	0,0000834	0,006247 ($p > 0,05$)
Resíduo	786	10,49761	0,013356	
Total	791	10,49803		

$F_{(0,05;\ 5;\ 786)} = 2,23$

Para a análise dos dados utilizou-se a transformação *arco seno* $(x_i/100)^{1/2}$, sendo que o resultado de F, não significativo, conclui que não houve uma diferença significativa entre as médias de porcentagens das mensagens nulas em estudo, no nível de 5% de significância.

Entretanto as pequenas variações obtidas nos valores médios de porcentagem de mensagens nulas, em determinadas situações relativas ao contexto do trabalho, representam quantidades significativas na análise do desempenho das simulações em relação à quantidade de mensagens nulas observadas.

Tabela A.5 Média, desvio-padrão e resultado do teste estatístico da porcentagem de mensagens nulas.

AA	AB	BA	BB	CRAY	SP2
45,88 ± 11,02 A (1)	45,98 ± 11,04 A	45,89 ± 11,00 A	46,02 ± 11,03 A	45,88 ± 10,99 A	45,80 ± 11,09 A

(1) Médias seguidas de mesma letra não diferem entre si.

A.5 Conclusões

Considerando-se o valor da quantidade de simulações tomadas em cada uma das plataformas (n_i = 132, i = 1, ...,6), e a distribuição aproximadamente normal, para a comparação das médias de tempo e da porcentagem utilizou-se a técnica de Análise de Variância , F = 248,54 ($p < 0,0001$), complementada com o Teste de Tukey para a comparação múltipla de médias de tempo de simulado.

Na análise da porcentagem das mensagens nulas, os dados sofreram a transformação *arc sem* $(x_i/100)^{1/2}$, sendo que o resultado de F = 0,006 foi não significativo ($p > 0,05$).

Bibliografia Utilizada

Achcar, J. A.; Rodrigues, J. Introdução à Estatística para Ciências e Tecnologia. Notas do ICMC-USP, 1995

Costa Neto, P. L. O. Estatística. Edgard Blûcher Ltda, 1997.

Mendenhall, W.; Sincich, T. Statistic for Engineering and the Sciences, 1992.

Kendal, M. G. *Factor analysis as statistical techinique.* J. R. Statist. Soc. B., 12, 60-75, 1950.

APÊNDICE B

Código das Simulações e Relatórios Estatísticos

Neste apêndice estão disponíveis os códigos construídos para realizar o estudo do comportamento do protocolo CMB. Por uma questão de simplificação optou-se por apresentar, neste apêndice, apenas o código das simulações cujas plataformas possuem características extremas para o desempenho dos elementos de processamento e do meio de comunicação, para cada um dos modelos de aplicação. No final deste apêndice encontra-se um exemplo dos relatórios utilizados para realizar a validação do modelo construído.

B.1 Arquitetura AA/Modelo Sistema do Servidor Central

```
#include "smpl.h"
#include "smpl.c"
#include "stat.c"
#include "rand.c"
#include "conio.h"
#include <stdio.h>
#include <dos.h>


#define tasks 500 // total de tarefas
#define qd 1

// ***************************************************************************
// ************************** Medidas de Desempenho *****************
// ***************************************************************************


real // meio físico  , NORMAL()

        mf1=0.00000166,
        mf2=0.00000001,  // desvio padrão

        tservicocpu =1.503,

        tdisco = 4.509,
        media = 1.12;
// ***************************************************************************
// ***************************************************************************
int
        prob_bloq = 100; // probabilidade de p1 e p2 bloquearem

real   nts,
        ciclos = 50000.0; // numero de ciclos

int     semente; // valor da semente do gerador

// ***************************************************************************
// ********************** Variaveis do modelo ******************************
int  i,
 cpu,
 disk1,
 disk2,
 disk3,
 disk4,
 meiofisicovai, // facilities dos meio fisicos
 meiofisicovem;
```

```
// ******* Variaveis do Modulo de Distribuicao ******************************

 char x;
 int
  probproc, // var. aleatoria
  prob, // var. aleatoria
  prob1, // var. aleat.
  prob2, // var. aleat. de probabilidade de bloqueio
  entroup1, // indica quando p1 esta bloqueado
  entroup2, // indica quando p2 esta bloqueado
  req;// numero do evento associado a geracao de nulas ou completas real
        nvai,   // variáveis contadoras de mensagens
        nvem,
        tvai, // tarefas que saem da cpu
        tvem, // tarefas que voltam a cpu
        nula,// nvai + nvem
        nulas_no_sistema;

 int segundos1, segundos2; // var. de tempo real de simulação
 struct dostime_t t1,t2;   // tempo sim. real,  para uso do gettime

 real
    acaboubloq, // variaveis de tempo de bloqueio
    comecoubloq,
    comecoubloq2,
    acaboubloq2,

    clock_cpu, // LOCAL VIRTUAL TIME DO CPU()
    clock_disco, // LOCAL VIRTUAL TIME DO DISCOS()
    gvt, // GLOBAL VIRTUAL TIME
    aux1,
    acumulo_cpu, // acumulam tempo de bloqueio
    acumulo_disco,
    porcentagem;


 struct token
  {
  int un;
  } task[tasks+1];

 struct token *p;
 int aux_token[500],aux_token2[500];
 int final_cpu=-1, ponta=0, final_disco=-1; // para manipular a fila


// ************************* Variaveis de auxilio a estat¡stica *******

   real
   soma_tempos=0.0,soma_clocks=0.0,soma_t_p1_p2 = 0.0,
   soma_t_p2_p1=0.0,soma_n_p1_p2 = 0.0, soma_n_p2_p1 = 0.0,
```

```
   soma_acumulo_cpu = 0.0, soma_acumulo_disco = 0.0,
   soma_porcent = 0.0;


// ************************************************************************
// ************************* Manipulação do CMB ******************
// ************************************************************************


void inserir_fila(int *vet_token, int tarefa, int *final_fila)
 {
    if (*final_fila == 499)
            {printf("Fila cheia \n");
                x=getchar();}
                  else
                    {
                        *final_fila = *final_fila +1;
                        vet_token[*final_fila]=tarefa;
                    }
 }

// ***************************************************************************

int retirar_fila(int *vet_token,int *final_fila)
 { int i,saida;
    if (*final_fila ==-1) then
                  {printf("Fila vazia \n");
                   x=getchar();}
                  else
                    {
                        saida=vet_token[ponta];
                        for (i=0;i<=*final_fila - 1;i++)
                            {
                                vet_token[i]=vet_token[i+1];

                              }
                        *final_fila = *final_fila-1;
                    }

  return(saida);
 }

// ************************* Bloqueio CPU() *************************

  void bloqueia_ou_nao_p1()
   {
   int tkn_bloq=0; // flag que identifica se foi esta msg que bloqueou

   prob2=random(1,100);

   // deu prob2, nao esta bloq, , completa
   // insere na fila e envia nula
   if ((prob2<=prob_bloq) && (entroup1==0) && (i!=-1))
          {comecoubloq = time();// tomar tempo
```

```
            entroup1=1;  // deixar bloqueado
            inserir_fila(aux_token,i,&final_cpu);
            tkn_bloq=1;
            schedule(4,0.0,-1);
            nulas_no_sistema = nulas_no_sistema + 1.0;
           }

     // nao esta bloq, , completa
    // utiliza cpu e escalona proximo evento
     if ((entroup1==0) && (i!=-1))
           { if (request(cpu,i,0)!=qd) then { aux1= expntl(tservicocpu);
                        schedule(3,aux1,i);}
           }

     // nao esta bloq, , nula
    // vai para o evento 3, onde sai do sistema
     if ((entroup1==0) && (i==-1))
           { aux1= expntl(tservicocpu);
             schedule(3,aux1,-1);}


     // esta bloq, , completa
    //  vai para fila e manda nula para evento 4
     if ((tkn_bloq!=1) && (entroup1==1) && (i!=-1))
           {  inserir_fila(aux_token,i,&final_cpu);
            }

     // esta bloq, , nula
    //  esvazia fila, todas requisitam cpu e depois vao para evento 3
    if ((i==-1) && (entroup1==1))
           {acaboubloq = time(); // tempo final do bloqueio
            entroup1 =0; // desbloquear
            acumulo_cpu = acumulo_cpu +(acaboubloq -comecoubloq);

             if (final_cpu>=0) then
                                {
                                        do {
                                                i=retirar_fila(aux_token,&final_cpu);
                                                if (request(cpu,i,0)!=qd) then {
                                                        aux1= expntl(tservicocpu);
                                                        schedule(3,aux1,i);}
                                          } while (final_cpu>=0);
                                          nulas_no_sistema=nulas_no_sistema-1.0;
                                }

           }
     }

// ************************** Bloqueio Discos() **********************

   void bloqueia_ou_nao_p2()
{
    int fuieu2=0, recurso,disc;
```

```
prob2=random(1,100);

// deu prob2, nao esta bloq, , completa
// insere na fila e manda nula
if ((prob2<=prob_bloq) && (entroup2==0) && (i!=-1))
        {comecoubloq2 = time();// tomar tempo
         entroup2=1;  // deixar bloqueado
         inserir_fila(aux_token2,i,&final_disco);
         fuieu2=1;
         schedule(15,0.0,-1);
         nulas_no_sistema = nulas_no_sistema + 1.0;
        }

 // nao esta bloq, , completa
// manda escolher disco
 if ((entroup2==0) && (i!=-1))
        { schedule(30,0.0,i); }

// nao esta bloq, , nula

if ((entroup2==0) && (i==-1))
        { schedule(8,erlang(tdisco,media),-1);} // manda nula p/ 8 (sai do sistema)
                                // gasta o tempo do processo logico disco

// esta bloq, , completa
//  vai para fila e manda nula para evento escalo. mfvem
 if ((fuieu2!=1) && (entroup2==1) && (i!=-1))
        {  inserir_fila(aux_token2,i,&final_disco);
        }

 // esta bloq, , nula
//  esvazia fila, todas requisitam cpu e depois vao para evento 3
if ((i==-1) && (entroup2==1))
        {acaboubloq2 = time(); // tempo final do bloqueio
         entroup2 =0; // desbloquear
         acumulo_disco = acumulo_disco +(acaboubloq2 -comecoubloq2);

         if (final_disco>=0) then
                                {
                                        do {
                                                i=retirar_fila(aux_token2,&final_disco);
                                                schedule(30,0.0,i);
                                          } while (final_disco>=0);
                                          nulas_no_sistema=nulas_no_sistema-1.0;
                                }         }    }


// ********************************* Atualizar GVT ****************************

 void atualizar_gvt()
   {
       if (clock_cpu < clock_disco) then
                                        {gvt = clock_cpu;}
                                else
```

```
                                        {gvt = clock_disco;}
    }

//********************************** Processo CPU ****************************
  void Proc_cpu()

        { // clock_cpu = l5[i]+clock + 1;
          // A linha acima Valida a situação 1. O processo fica bloqueado
          // com a chegada de uma mensagem com timestamp maior que o clock
          // do processo.

          // testando mensagem defasada!
          if (clock_cpu > (l5[i]+clock)) then
                                  { printf("\n Mensagem Defasada! Contrariando
 Principio Conservativo. Cpu() Bloqueado.");
                                   x=getchar();
                                   }

          bloqueia_ou_nao_p1();


          clock_cpu=clock; // atualizando o clock da cpu
          atualizar_gvt(); // atualizando o Global Virtual Time

  }
// ******************************** Processo Discos **************************

  void Proc_discos()
  {     // testando mensagem defasada!
          if (clock_disco > (l5[i]+clock)) then
                                  { printf("\n Mensagem Defasada! Contrariando
Principio Conservativo. Discos() Bloqueado.");
                                   x=getchar();
                                   }

           clock_disco=clock;
           atualizar_gvt();

           bloqueia_ou_nao_p2();

  }


// %%%%%%%%%%%%%  MAIN %%%%%%%%%%%%%%%%%%%%%%

 void main()
 {

 int
 event;

 //semente = 1; // valor da semente do gerador

 for (semente=1;semente<=3;semente++)
```

```
{
// Iniciando vari veis

nts = ciclos;
entroup1=0; // indica quando p1 esta bloqueado
entroup2=0; // indica quando p2 esta bloqueado
nvai=0.0;    // vari veis contadoras de mensagens
nvem=0.0;
tvai=0.0; // tarefas que saem da cpu
tvem=0.0; // tarefas que voltam a cpu
nula=0.0;// nvai + nvem
nulas_no_sistema=0.0;
clock_cpu=0.0; // LOCAL VIRTUAL TIME DO CPU()
clock_disco=0.0; // LOCAL VIRTUAL TIME DO DISCOS()
gvt=0.0; // GLOBAL VIRTUAL TIME
acumulo_cpu=0.0; // acumulam tempo de bloqueio
acumulo_disco=0.0;
final_cpu=-1; ponta=0; final_disco=-1; // para manipular a fila

// Iniciando Sistema SMPL

smpl(0,"SIMULAÇÃO: CMB/Plataforma AA");

// valor da semente
strm =semente;   // este valor sera atribuido diretamente a strm

cpu=facility("CPU",1);
disk1=facility("disk1",1);
disk2=facility("disk2",1);
disk3=facility("disk3",1);
disk4=facility("disk4",1);

meiofisicovai = facility ("MF Vai",300);
meiofisicovem = facility ("MF Vem",300);


for(i=1;i<=tasks;i++) schedule(1,0.0,i);
clrscr();

 // iniciando tempo do dos
 _dos_gettime(&t1);


while (nts)

{

cause(&event,&i);
p=&task[i];
switch(event)
{
        case 1: // comeco de tour
          schedule(2,0.0,i);
         break;
```

```
// REQUISITAR CPU EM P1
case 2:
 Proc_cpu();
 break;
// LIBERANDO CPU EM P1
case 3:
  if (i!=-1)  then  // se for nula nao escalona mais
            { release(cpu,i);
              schedule(4,0.0,i);
            }
   else
   {
    nulas_no_sistema = nulas_no_sistema - 1.0;
   }

   if (nulas_no_sistema == 0.0) then { schedule(4,0.0,-1);
                                       nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                       }
// aqui morrem as nulas vindas do processo disco()
break;
// REQUISITANDO MF_VAI
case 4:
if (request(meiofisicovai,i,0)!=qd)  then
      {schedule(5,normal(mf1,mf2),i);}  // tempo de comunicacao
                                        // gasto p/ ser enviada
   break;
// LIBERANDO MF_VAI
case 5:
   release(meiofisicovai,i);
   // contabilizando envio de tarefa (completa)
   if (i!=-1) then { tvai = tvai +1.0;}
   // contabilizando envio de nula
   else {nvai = nvai+1;}// envio nula para evento 6
   schedule(6,0.0,i);
   break;

// REQUISITANDO DISCO
case 6:
 Proc_discos();
  break;
case 30: // ESCOLHA DO DISCO
  if (i!=-1)
      {
            p->un=random(1,4);
            switch (p->un)
            {

            case 1:
                  schedule(7,0.0,i);
                  break;
            case 2:
                  schedule(9,0.0,i);
```

```
                        break;
                case 3:
                        schedule(11,0.0,i);
                        break;
                case 4:
                        schedule(13,0.0,i);
                        break;
                }
        }
        break;
 case 7:  // requisitando disco1
 if (i!=-1) then // nulas nao utilizam facility disk()
 {
  if (request(disk1,i,0)!=qd)
        {schedule(8,erlang(tdisco,media),i);}
 }
 else
 { schedule(8,erlang(tdisco,media),i); // aqui vai o tempo para nula
 }                        // ser processada
 break;
// liberando disco1
case 8:
 if (i!=-1) // só as tarefas deixam o discos()
 then
 { release(disk1,i);
   schedule(15,0.0,i);
 }
 else
 {
 nulas_no_sistema = nulas_no_sistema - 1.0;
 }
  if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                      nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                      }
// aqui morrem as nulas vindas do processo cpu()
break;
 case 9:  // requisitando disco2
 if (i!=-1) then // nulas nao utilizam facility disk()
 {
  if (request(disk2,i,0)!=qd)
        {schedule(10,erlang(tdisco,media),i);}
 }
 else
 { schedule(10,erlang(tdisco,media),i); // aqui vai o tempo para nula
 }                        // ser processada

  break;
// liberando disco2
case 10:
 if (i!=-1) // só as tarefas vao deixar o discos()
 then
 { release(disk2,i);
   schedule(15,0.0,i);
```

```
}        else
{
nulas_no_sistema = nulas_no_sistema - 1.0;
}

 if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                     nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                     }
// aqui morrem as nulas vindas do processo cpu()
break;
 case 11:  // requisitando disco3
 if (i!=-1) then // nulas nao utilizam facility disk()
 {
  if (request(disk3,i,0)!=qd)
       {schedule(12,erlang(tdisco,media),i);}
 }
 else
 { schedule(12,erlang(tdisco,media),i); // aqui vai o tempo para nula
 }                          // ser processada
  break;
// liberando disco3
case 12:
 if (i!=-1) // s¢ as tarefas vao deixar o discos()
 then
 { release(disk3,i);
   schedule(15,0.0,i);
 }
 else
 {
 nulas_no_sistema = nulas_no_sistema - 1.0;
 }

 if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                     nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                     }
// aqui morrem as nulas vindas do processo cpu()
 break;

case 13:  // requisitando disco4

 if (i!=-1) then // nulas nao utilizam facility disk()
 {
  if (request(disk4,i,0)!=qd)
       {schedule(14,erlang(tdisco,media),i);}
 }
 else
 { schedule(14,erlang(tdisco,media),i); // aqui vai o tempo para nula
 }                          // ser processada
   break;
// liberando disco4
case 14:
 if (i!=-1) // s¢ as tarefas vao deixar o discos()
```

```
          then
          { release(disk4,i);
            schedule(15,0.0,i);
          }
          else
          {
          nulas_no_sistema = nulas_no_sistema - 1.0;
          }
           if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                                nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                                }
        // aqui morrem as nulas vindas do processo cpu()
        break;


        // REQUISITANDO MF_VEM
        case 15:
                if (request(meiofisicovem,i,0)!=qd)
                   {schedule(16,normal(mf1,mf2),i);}
                break;
        // LIBERANDO MF_VEM
        case 16:
                release(meiofisicovem,i);

                if (i!=-1)  then { tvem = tvem +1.0;}// envio de tarefa
                            // contabilizando envio de nula
                            else {nvem = nvem+1.0;}
                if (i!=-1) // Toma o tempo apenas se for tarefa
        {nts=nts-1;}
                schedule(1,0.0,i);  // ESCALONADO PARA RECOMECAR CICLO

          break;
      }  }

// ***************************************************************************
// ************************   RELATORIOS   ***************************
// ***************************************************************************

  _dos_gettime(&t2); // fim do tempo dos
  segundos1 = t1.hour*60*60 + t1.minute*60 + t1.second;
  segundos2 = t2.hour*60*60 + t2.minute*60 + t2.second;

  soma_tempos = soma_tempos + (segundos2-segundos1);
  soma_clocks = soma_clocks + clock;
  soma_t_p1_p2 = soma_t_p1_p2 + tvai;
  soma_t_p2_p1 = soma_t_p2_p1 + tvem;
  soma_n_p1_p2 = soma_n_p1_p2 + nvai;
  soma_n_p2_p1 = soma_n_p2_p1 + nvem;
  soma_acumulo_cpu = soma_acumulo_cpu + acumulo_cpu;
  soma_acumulo_disco = soma_acumulo_disco + acumulo_disco;
  nula = nvai+nvem;
  porcentagem=(100.0 * nula)/(nula+tvai+tvem);
  soma_porcent = soma_porcent + porcentagem;
```

```
} // fim da estrutura "for" para troca de semente

    reportf();
    printf("*********************************************************************************** \n");
    printf("Tempo M,dio (segundos): %f \n",soma_tempos/3);
    printf("\nClock M,dio: %f                    ",soma_clocks/3);
    printf("%1.f M,dia Tarefas P1 -> P2 \n",soma_t_p1_p2/3);
    printf("Acumulo m,dio da cpu = %f            ",soma_acumulo_cpu/3);
    printf("%1.f M,dia Tarefas P2 -> P1 \n",soma_t_p2_p1/3);
    printf("Acumulo M,dio do disco = %f          ",soma_acumulo_disco/3);
    printf("%1.f M,dia Nulas  P1 -> P2 \n",soma_n_p1_p2/3);

    printf("Clock com bloqueio: %f
 ",(soma_clocks+soma_acumulo_cpu+soma_acumulo_disco)/3);
    printf("%1.f M,dia Nulas  P2 -> P1 \n",soma_n_p2_p1/3);

    printf("\n %1.f M,dia Tarefas Transmitidas
 ",(soma_t_p1_p2+soma_t_p2_p1)/3);
    printf("%1.f M,dia Nulas Transmitidas ",(soma_n_p1_p2+soma_n_p2_p1)/3);

    printf("                    %.2f % Mens. Nulas/Tarefas ",soma_porcent/3);
    x=getchar();
    clrscr();
} // fim do programa
```

B.2 Arquitetura BB/Modelo Sistema Computacional Simplificado

```
#include "smpl.h"
#include "smpl.c"
#include "stat.c"
#include "rand.c"
#include "conio.h"
#include <stdio.h>
#include <dos.h>
#define qd 1
// ***************************************************************************
// ************************* Medidas de Desempenho *****************
// ***************************************************************************

real // meio fisico  , NORMAL()
        // B -> Baixo
        mf1=0.003788,
        mf2=0.000001,  // desvio padrao
        tservicocpu =72.0;
        tdisco = 216.0,      // SP2 = erlang(30.0, 7.5)
        media = 54.0;        // CRAY= erlang(56.84,14.21)
```

```
// ****************************************************************************
int
        prob_bloq = 100,
        entradas = 2000,  // 1000 e 2000
        semente; //    valor da semente do gerador
real
        tempo_chegada = 100.0;

// ****************************************************************************
// ********************** Variaveis do modelo ****************************
int  i,
outro=1,
saiu=1,
dsc,
  cpu,
  disk1,
  disk2,
  meiofisicovai, // facilities dos meio fisicos
  meiofisicovem;

// ******* Variaveis do Modulo de Distribuicao ******************************

char x;
int
  probproc, // var. aleatória
  prob, // var. aleatória
  prob1, // var. aleat.
  prob2, // var. aleat. de probabilidade de bloqueio
  prob3, // var. aleat. para probabilidade de gerar tarefa nova ou nao
  entroup1, // indica quando p1 esta bloqueado
  entroup2, // indica quando p2 esta bloqueado
  req;// numero do evento associado a geração de nulas ou completas
real
        nvai,    // variáveis contadoras de mensagens
        nvem,
        tvai, // tarefas que saem da cpu
        tvem, // tarefas que voltam a cpu
        nula,// nvai + nvem
        nulas_no_sistema;

int segundos1, segundos2; // var. de tempo real de simulação
struct dostime_t t1,t2;   // tempo sim. real,  para usar _dos_gettime
real
    acaboubloq, // variáveis de tempo de bloqueio
    comecoubloq,
    comecoubloq2,
    acaboubloq2,

    clock_cpu, // LOCAL VIRTUAL TIME DO CPU()
    clock_disco, // LOCAL VIRTUAL TIME DO DISCOS()
    gvt, // GLOBAL VIRTUAL TIME
    aux1,
    acumulo_cpu, // acumulam tempo de bloqueio
```

```
   acumulo_disco,
   m_tarefas, // acumula a quantidade de tarefas no sistema em cada ciclo
   total_ciclos,// soma o número de ciclos
    mediatotal_ciclos=0.0, // media de ciclos na simulação
   me_tarefas, // média de tarefas por semente utilizada
   media_tarefas=0.0, // media total durante as execuções
   porcentagem;

int aux_token[500],aux_token2[500];
int final_cpu=-1, ponta=0, final_disco=-1; // para manipular a fila

// ************************ Variaveis de auxilio a estatística *******
   real
   soma_tempos=0.0,soma_clocks=0.0,soma_t_p1_p2 = 0.0,
   soma_t_p2_p1=0.0,soma_n_p1_p2 = 0.0, soma_n_p2_p1 = 0.0,
   soma_acumulo_cpu = 0.0, soma_acumulo_disco = 0.0,
   soma_porcent = 0.0;

void inserir_fila(int *vet_token, int tarefa, int *final_fila)
 {
    if (*final_fila == 499)
            {printf("Fila cheia \n");
               x=getchar();}
                 else
                   {
                       *final_fila = *final_fila +1;
                       vet_token[*final_fila]=tarefa;
                   } }

// ****************************************************************************
int retirar_fila(int *vet_token,int *final_fila)
 { int i,saida;
    if (*final_fila ==-1) then
                 {printf("Fila vazia \n");
                  x=getchar();}
                 else
                   {
                       saida=vet_token[ponta];
                       for (i=0;i<=*final_fila - 1;i++)
                           {
                               vet_token[i]=vet_token[i+1];                    }
                       *final_fila = *final_fila-1;                 }
   return(saida);
  }

// ************************* Bloquear ou nao CPU() **************************
   void bloqueia_ou_nao_p1()
    {
    int tkn_bloq=0; // flag que identifica se foi esta msg que bloqueou
    prob2=random(1,100);
    // deu prob2, nao esta bloq, , completa
    // insere na fila e manda nula
    if ((prob2<=prob_bloq) && (entroup1==0) && (i!=-1))
          {comecoubloq = time();// tomar tempo
```

```
            entroup1=1;  // deixar bloqueado
            inserir_fila(aux_token,i,&final_cpu);
            tkn_bloq=1;
            schedule(4,0.0,-1);
            nulas_no_sistema = nulas_no_sistema + 1.0;
           }
     // nao esta bloq, , completa
    // utiliza cpu e escalona proximo evento
     if ((entroup1==0) && (i!=-1))
           { if (request(cpu,i,0)!=qd) then { aux1= expntl(tservicocpu);
                        schedule(3,aux1,i);}
           }     // nao esta bloq, , nula
    // vai para o evento 3, onde morre
     if ((entroup1==0) && (i==-1))
           { aux1= expntl(tservicocpu);
             schedule(3,aux1,-1);}
     // esta bloq, , completa
    //  vai para fila e manda nula para evento 4
     if ((tkn_bloq!=1) && (entroup1==1) && (i!=-1))
           {  inserir_fila(aux_token,i,&final_cpu);          }
     // esta bloq, , nula
    //  esvazia fila, todas requisitam cpu e depois vao para evento 3
    if ((i==-1) && (entroup1==1))
           {acaboubloq = time(); // tempo final do bloqueio
            entroup1 =0; // desbloquear
            acumulo_cpu = acumulo_cpu +(acaboubloq -comecoubloq);

            if (final_cpu>=0) then
                                {
                                        do {
                                                i=retirar_fila(aux_token,&final_cpu);
                                                if (request(cpu,i,0)!=qd) then {
                                                        aux1= expntl(tservicocpu);
                                                        schedule(3,aux1,i);}
                                          } while (final_cpu>=0);
                                          nulas_no_sistema=nulas_no_sistema-1.0;
                                }          }    }

// ************************** Bloqueio Discos() **********************

   void bloqueia_ou_nao_p2()
{
    int fuieu2=0, recurso,disc;
    prob2=random(1,100);
    // deu prob2, nao esta bloq, , completa
    // insere na fila e manda nula
    if ((prob2<=prob_bloq) && (entroup2==0) && (i!=-1))
          {comecoubloq2 = time();// tomar tempo
           entroup2=1;  // deixar bloqueado
           inserir_fila(aux_token2,i,&final_disco);
           fuieu2=1;
           schedule(15,0.0,-1);
           nulas_no_sistema = nulas_no_sistema + 1.0;
          }
```

IFSC-USP SERVIÇO DE BIBLIOTECA INFORMAÇÃO

```
 // nao esta bloq, , completa
// manda escolher disco
 if ((entroup2==0) && (i!=-1))
        { schedule(30,0.0,i); }
 // nao esta bloq, , nula
 if ((entroup2==0) && (i==-1))
        { schedule(8,erlang(tdisco,media),-1);}
                                // gasta o tempo do processo logico disco
 // esta bloq, , completa
// vai para fila e manda nula para evento escalo. mfvem
 if ((fuieu2!=1) && (entroup2==1) && (i!=-1))
        { inserir_fila(aux_token2,i,&final_disco);
        }

 // esta bloq, , nula
// esvazia fila, todas requisitam cpu e depois vão para evento 3
if ((i==-1) && (entroup2==1))
        {acaboubloq2 = time(); // tempo final do bloqueio
         entroup2 =0; // desbloquear
         acumulo_disco = acumulo_disco +(acaboubloq2 -comecoubloq2);

         if (final_disco>=0) then
                                {
                                        do {
                                                i=retirar_fila(aux_token2,&final_disco);
                                                schedule(30,0.0,i);
                                          } while (final_disco>=0);
                                          nulas_no_sistema=nulas_no_sistema-1.0;
                                }         }    }

// ******************************** Atualizar GVT ***************************

 void atualizar_gvt()
   {
       if (clock_cpu < clock_disco) then
                                        {gvt = clock_cpu;}
                                else
                                        {gvt = clock_disco;}
   }

//******************************** Processo CPU ****************************
 void Proc_cpu()

       { // clock_cpu = l5[i]+clock + 1;
         // A linha acima Valida a Situa‡Æo 1. O processo fica bloqueado
         // com a chegada de uma mensagem com timestamp maior que o clock
         // do processo.

         // testando mensagem defasada!
         if (clock_cpu > (l5[i]+clock)) then
                                { printf("\n Mensagem Defasada! Contrariando
Principio Conservativo. Cpu() Bloqueado.");
                                 x=getchar();
```

```
                                    }

            bloqueia_ou_nao_p1();


            clock_cpu=clock; // atualizando o clock da cpu
            atualizar_gvt(); // atualizando o Global Virtual Time

  }
// ****************************** Processo Discos **************************

  void Proc_discos()
  {     // testando mensagem defasada!
            if (clock_disco > (l5[i]+clock)) then
                                    { printf("\n Mensagem Defasada! Contrariando
Principio Conservativo. Discos() Bloqueado.");
                                    x=getchar();
                                    }

              clock_disco=clock;
              atualizar_gvt();

              bloqueia_ou_nao_p2();

   }

// Especifica para este modelo: calcula a média de tarefas no sistema para
// cada ciclo realizado por uma tarefa completa

  void tarefas()
  {
    m_tarefas = m_tarefas + (outro-saiu);
    total_ciclos = total_ciclos +1;
  }


// %%%%%%%%%%%%%%%%%%%%  MAIN  %%%%%%%%%%%%%%%%%%%%

  void main()
  {

  int
  event;

  for (semente=4;semente<=9;semente++)
  {
  // Iniciando vari veis

  outro=1, saiu =0;
  entroup1=0; // indica quando p1 esta bloqueado
  entroup2=0; // indica quando p2 esta bloqueado
  nvai=0.0;    // vari veis contadoras de mensagens
  nvem=0.0;
  tvai=0.0; // tarefas que saem da cpu
```

```
tvem=0.0; // tarefas que voltam a cpu
nula=0.0;// nvai + nvem
nulas_no_sistema=0.0;
clock_cpu=0.0; // LOCAL VIRTUAL TIME DO CPU()
clock_disco=0.0; // LOCAL VIRTUAL TIME DO DISCOS()
gvt=0.0; // GLOBAL VIRTUAL TIME
acumulo_cpu=0.0; // acumulam tempo de bloqueio
acumulo_disco=0.0;

m_tarefas =0.0;
total_ciclos=0.0;
me_tarefas=0.0;
final_cpu=-1; ponta=0; final_disco=-1; // para manipular a fila
// Iniciando Sistema SMPL
smpl(0," CMB/Plataforma BB/Sistema Computacional Simplificado");

// valor da semente
strm =semente;
cpu=facility("CPU",1);

disk1=facility("disk1",1);
disk2=facility("disk2",1);
meiofisicovai = facility ("MF Vai",1);
meiofisicovem = facility ("MF Vem",1);

// Escalonando primeira tarefa para o sistema
schedule(1,0.0,1);

clrscr();

 // iniciando tempo do dos
 _dos_gettime(&t1);

while (outro<entradas)   // numero m ximo suportado pelo smpl
{
cause(&event,&i);

switch(event)
{
        case 1: // comeco de tour

          schedule(2,0.0,i);
          prob3 = random(1,100);

         // calculo da media de tarefas por ciclo
         if  (i!= -1) then tarefas();

         if (((i!= -1) && (prob3<=75)) || (outro-saiu == 0))
         {
          outro = outro +1;   // de 0.0005 a 100.0 ou mais
          schedule(1,expntl(tempo_chegada),outro);
          }

          break;
```

```
// REQUISITAR CPU EM P1
case 2:
 Proc_cpu();
 break;

// LIBERANDO CPU EM P1
case 3:
  if (i!=-1) then // se for nula nao escalona mais
            { release(cpu,i);
              schedule(4,0.0,i);
            }
  else
  {
   nulas_no_sistema = nulas_no_sistema - 1.0;
  }

  if (nulas_no_sistema == 0.0) then { schedule(4,0.0,-1);
                                      nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                    }
// aqui morrem as nulas vindas do processo disco()
break;

// REQUISITANDO MF_VAI
case 4:

if (request(meiofisicovai,i,0)!=qd) then
      {schedule(5,normal(mf1,mf2),i);} // tempo de comunicacao
                                       // gasto p/ ser enviada
  break;

// LIBERANDO MF_VAI
case 5:
  release(meiofisicovai,i);
  // contabilizando envio de tarefa (completa)
  if (i!=-1) then { tvai = tvai +1.0;}
  // contabilizando envio de nula
  else {nvai = nvai+1.0;}// envio nula para evento 6
  schedule(6,0.0,i);
  break;

// REQUISITANDO DISCO
case 6:
 Proc_discos();
 break;

case 30: // ESCOLHA DO DISCO OU SAI DO SISTEMA

 if (i!=-1)
    {
          dsc=random(1,100);
          if (dsc <=25) then dsc = 1;
          if ((dsc>25) && (dsc<=50)) then dsc = 2;
```

```
                    if (dsc >50) then dsc = 3;
                    switch (dsc)
                    {

                    case 1:
                            schedule(7,0.0,i);
                            break;

                    case 2:
                            schedule(9,0.0,i);
                            break;

                    case 3:
                            // completa sai do sistema
                            saiu = saiu +1;
                            if (nulas_no_sistema == 0.0) then
                                        { schedule(1,0.0,-1);
                                         nulas_no_sistema = nulas_no_sistema
+ 1.0;
                                        }
                            break;          }       }

            break;

        case 7:  // requisitando disco1
        if (i!=-1) then // nulas nao utilizam facility disk()
        {
         if (request(disk1,i,0)!=qd)
              {schedule(8,erlang(tdisco,media),i);}
        }
        else
        { schedule(8,erlang(tdisco,media),i); // aqui vai o tempo para nula
        }                       // ser processada

        break;

       // liberando disco1
       case 8:

        if (i!=-1) // s¢ as tarefas vao deixar o discos()
        then
        { release(disk1,i);
          schedule(15,0.0,i);   }
        else
        {
        nulas_no_sistema = nulas_no_sistema - 1.0;          }

         if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                             nulas_no_sistema = nulas_no_sistema
+ 1.0;                                       }
       // aqui morrem as nulas vindas do processo cpu()
       break;

       case 9:  // requisitando disco2
```

```
        if (i!=-1) then // nulas nao utilizam facility disk()
        {
         if (request(disk2,i,0)!=qd)
              {schedule(10,erlang(tdisco,media),i);}
        }
        else
        { schedule(10,erlang(tdisco,media),i); // aqui vai o tempo para nula
        }                          // ser processada

        break;

       // liberando disco2
       case 10:

        if (i!=-1) // só as tarefas deixam o discos()
        then
        { release(disk2,i);
          schedule(15,0.0,i);   }
        else    {
        nulas_no_sistema = nulas_no_sistema - 1.0;          }

         if (nulas_no_sistema == 0.0) then { schedule(15,0.0,-1);
                                             nulas_no_sistema = nulas_no_sistema
+ 1.0;}
       // aqui morrem as nulas vindas do processo cpu()
       break;

       // REQUISITANDO MF_VEM
       case 15:
              if (request(meiofisicovem,i,0)!=qd)
                 {schedule(16,normal(mf1,mf2),i);}

             break;

       // LIBERANDO MF_VEM
       case 16:
             release(meiofisicovem,i);

             if (i!=-1)  then { tvem = tvem +1.0;}// envio de tarefa
                         // contabilizando envio de nula
                         else {nvem = nvem+1.0;}

             schedule(1,0.0,i);  // ESCALONADO PARA RECOMECAR CICLO

         break;      }  }

// ***************************************************************************
// ************************      RELATORIOS      ***************************
// ***************************************************************************
   _dos_gettime(&t2); // fim do tempo dos
   segundos1 = t1.hour*60*60 + t1.minute*60 + t1.second;
   segundos2 = t2.hour*60*60 + t2.minute*60 + t2.second;
   soma_tempos = soma_tempos + (segundos2-segundos1);
```

```
    soma_clocks = soma_clocks + clock;
    soma_t_p1_p2 = soma_t_p1_p2 + tvai;
    soma_t_p2_p1 = soma_t_p2_p1 + tvem;
    soma_n_p1_p2 = soma_n_p1_p2 + nvai;
    soma_n_p2_p1 = soma_n_p2_p1 + nvem;
    soma_acumulo_cpu = soma_acumulo_cpu + acumulo_cpu;
    soma_acumulo_disco = soma_acumulo_disco + acumulo_disco;
    nula = nvai+nvem;
    porcentagem=(100.0 * nula)/(nula+tvai+tvem);
    soma_porcent = soma_porcent + porcentagem;
          // calculo da media de tarefas por ciclo
    me_tarefas = m_tarefas/total_ciclos;
    media_tarefas = media_tarefas + me_tarefas;

    // calculo do numero medio de ciclos executados por semente
    mediatotal_ciclos = mediatotal_ciclos + total_ciclos;

} // fim da estrutura "for" para troca de semente

    reportf();
    printf("********************************************************************************** \n");
    printf("Tempo M,dio (segundos): %f ",soma_tempos/6);
    printf("        Clock M,dio: %f    \n ",soma_clocks/6);
    printf(" Clock com bloqueio: %f     \n
",(soma_clocks+soma_acumulo_cpu+soma_acumulo_disco)/6);
    printf("%1.f M,dia Tarefas P1 -> P2        ",soma_t_p1_p2/6);
printf("%1.f M,dia Tarefas P2 -> P1        ",soma_t_p2_p1/6);
printf("          %1.f M,dia Nulas  P1 -> P2        ",soma_n_p1_p2/6);
    printf("%1.f M,dia Nulas  P2 -> P1 \n",soma_n_p2_p1/6);
    printf("     %1.f M,dia Tarefas Transmitidas
",(soma_t_p1_p2+soma_t_p2_p1)/6);
    printf("%1.f M,dia Nulas Transmitidas ",(soma_n_p1_p2+soma_n_p2_p1)/6);
    printf(" \n    %.2f % Mens. Nulas/Tarefas        ",soma_porcent/6);
     printf(" %.2f  Nro. medio de Tarefas/Ciclo  \n",media_tarefas/6);
    printf("        N£mero de ciclos realizados por semente = %f
",mediatotal_ciclos/6);
    x=getchar();
    clrscr();
} // fim do programa
```

B.3 Relatórios dos Resultados da Simulação Seqüencial e da Simulação Implementada Através das Técnicas de Modelagem

Report: Simulação Seqüencial

smpl SIMULATION REPORT

MODEL: Central Server Model

TIME: 96527.553
INTERVAL: 96527.553

FACILITY	UTIL.	MEAN BUSY PERIOD	MEAN QUEUE LENGTH	OPERATION COUNTS RELEASE	PREEMPT	QUEUE
CPU	0.8151	6.202	1.286	12687	2682	7335
disk	0.7712	29.970	0.933	2484	0	1838
disk	0.7561	29.704	0.847	2457	0	1787
disk	0.7804	29.844	0.971	2524	0	1896
disk	0.7886	30.029	1.051	2535	0	1931

class 0 tour time = 99.34
class 1 tour time = 69.37

Report : Modelo Simulação Distribuída

smpl SIMULATION REPORT

MODEL: Servidor Central

TIME: 96647.306
INTERVAL: 96647.306

FACILITY	UTIL.	MEAN BUSY PERIOD	MEAN QUEUE LENGTH	OPERATION COUNTS RELEASE	PREEMPT	QUEUE
CPU	0.8146	6.281	1.212	12664	2680	7399
disk	0.7641	29.916	0.920	2494	0	1869
disk	0.7548	30.002	0.831	2447	0	1729
disk	0.7847	29.762	0.949	2476	0	1854
disk	0.7815	30.022	1.037	2583	0	1947
Meio Físico Va[60]	0.0000	0.000	0.000	10004	0	0
Meio Físico Ve[60]	0.0000	0.000	0.000	9999	0	0

class 0 tour time = 99.31
class 1 tour time = 71.33

APÊNDICE C

Estudo do Comportamento do Protocolo CMB: Resultados Adicionais

Neste apêndice são apresentados alguns resultados e gráficos adicionais obtidos durante a etapa de estudo do comportamento do protocolo CMB através da simulação descrita no capítulo 5.

C.1 Tempo de Processamento em Relação à Quantidade de Ciclos/Entradas

Os gráficos relacionados a esta seção mostram o comportamento do valor do tempo de processamento (tempo de máquina necessário para a

execução da simulação) em relação ao número de ciclos e tarefas que as simulações devem executar.

- **Modelo Servidor Central**

Considerando o modelo do servidor central, em todas as arquiteturas utilizadas, percebe-se que o crescimento dos valores de tempo da simulação é proporcional ao aumento do número de ciclos. Este comportamento é natural e pode ser explicado pela condição do término da simulação (Figura C.1).

Segundo a implementação do modelo do servidor central, a simulação deve ser processada até o momento em que um número pré-determinado de ciclos seja realizado. Portanto, quanto maior o número de ciclos, maior o tempo de processamento da simulação. Este comportamento foi confirmado por aferição em [Ulson 1999] onde foi realizada a simulação sob as mesmas condições.

A diferença entre os tempos de execução considerando quantidades distintas de tarefas (50, 100, 250 e 500 tarefas) mostra que, quanto maior o número de tarefas, maior será o tamanho das filas para a utilização dos recursos do modelo e, consequentemente, maior o tempo de processamento da simulação.

- **Modelo Sistema Computacional Simplificado**

Para o modelo do sistema computacional simplificado a condição de parada para cada simulação consiste na quantidade de tarefas que são inseridas no modelo durante a execução da simulação (sistema aberto).

Devido às restrições da extensão SMPL, mencionadas no capítulo 4, a quantidade máxima de tarefas permitida pelo SMPL é igual a 2000. Desta forma as simulações deste modelo eram concluídas após a chegada das 2000 tarefas.

Configuração				
Modelo	*Plataf.*	*% bloq.*	*tarefas*	*ciclos*
➢ SC	SP2	➢ 0	50	10000
SCS	CRAYT3D	10	100	30000
	➢ AA	20	250	50000
	AB	30	500	➢ var.
	BA	40	➢ var.	
	BB	50		
	var.	60		
		70		
		80		
		90		
		100		
		var.		

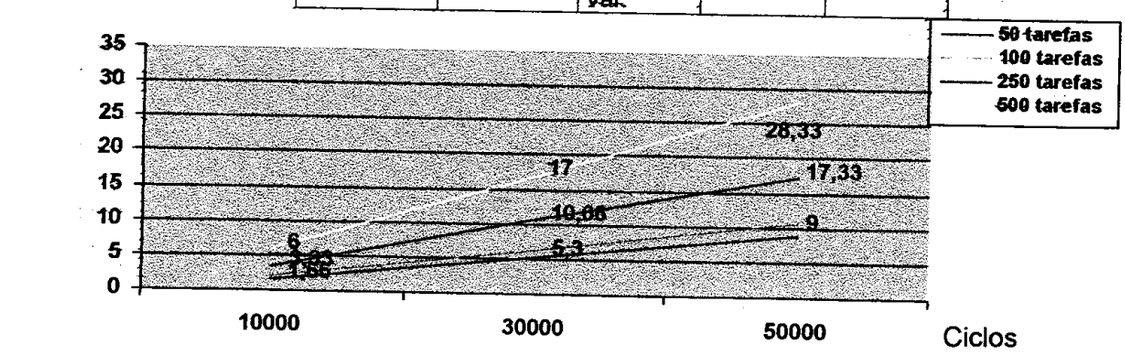


Figura C.1 Tempo Processamento x Ciclos : Modelo Servidor Central

No gráfico da figura C.2 estão representados os tempos obtidos para cada execução considerando valores distintos (1.0, 50.0 e 100.0) para o tempo médio entre chegadas das tarefas. O aumento nos tempos para cada uma destas execuções também é explicado pela condição de parada da simulação. Quanto maior a quantidade de tarefas que devem entrar no modelo, maior o tempo de execução da simulação. Além disso, pode-se observar que o valor dos tempos entre chegadas também influi diretamente

no tempo necessário para realizar as simulações, uma vez que, quanto maior o tempo entre as chegadas das tarefas, menor o número de tarefas no sistema e, finalmente, maior o tempo de processamento da simulação.

Configuração				
Modelo	***Plataf.***	***% bloq.***	***entradas***	***chegadas***
SC	SP2	➢ 0	1000	1.0
➢ SCS	CRAY	10	2000	50.0
	➢ AA	20	➢ var.	100.0
	AB	30		➢ var.
	BA	40		
	BB	50		
		60		
		70		
		80		
		90		
		100		
		var.		

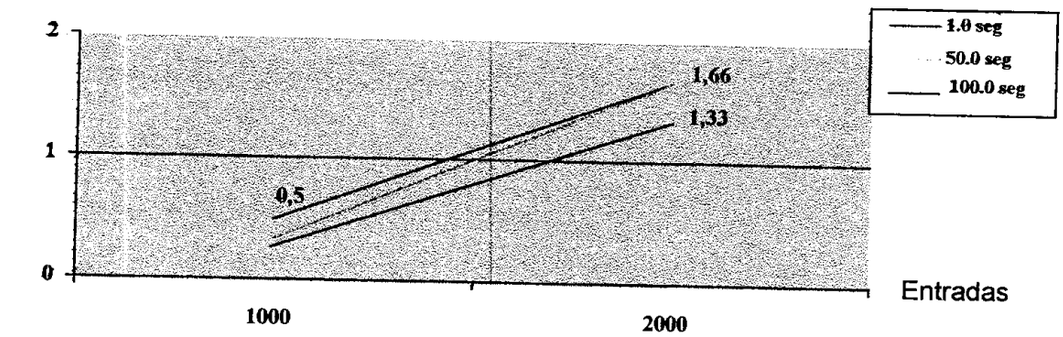


Figura C.2 Modelo Sistema Computacional Simplificado

C.2 Influência da Quantidade de Ciclos no Tempo de Processamento Considerando Diferentes Plataformas

- **Modelo Servidor Central**

Foi mencionado no capítulo 4 que as arquiteturas com baixo desempenho no recurso meio físico (mf) foram modeladas através da utilização de apenas um recurso smpl (*facility*). Esta implementação permitiu

a modelagem de um meio de comunicação compartilhado. Desta forma qualquer mensagem, nula ou completa, que necessite ser escalonada em um evento remoto deverá concorrer pelo meio físico. Esta abordagem assume características semelhantes a de um barramento em uma rede local. Assim sendo, na arquitetura AB, o alto desempenho atribuído ao processador associado à característica do recurso meio físico, descrita anteriormente, fez com que as mensagens permanecessem a maior parte do tempo utilizando ou requisitando o meio físico diminuindo o desempenho da simulação para esta arquitetura (Figura C.3). Isto pode ser comprovado através da variável estatística *tamanho médio da fila* fornecida pelo relatório do SMPL (Apêndice B). O valor desta variável é sensivelmente superior para a arquitetura AB em relação às outras arquiteturas. Com uma quantidade maior de mensagens sendo manipuladas na fila de espera do recurso meio físico a quantidade de operações de inserção e remoção de mensagens nesta fila também é maior.

Outro fator que confirma a presença de um número maior de mensagens no meio físico para a arquitetura AB é o valor da variável *período de utilização*, que na maioria das simulações foi superior para esta plataforma em relação às outras arquiteturas.

- **Modelo Sistema Computacional Simplificado**

As características do comportamento do tempo de processamento para as plataformas consideradas nesta tese não sofreram modificações significativas quando o modelo simulado é alterado.

C.3 Influência das Mensagens Nulas no Tempo de Processamento

O objetivo deste item é analisar o comportamento do tempo de processamento em relação à quantidade de mensagens nulas para uma execução concorrente do protocolo CMB.

Configuração				
Modelo	*Plataf.*	*% bloq.*	*tarefas*	*ciclos*
➢ SC	➢ SP2	0	50	10000
SCS	➢ CRAY	10	100	30000
	➢ AA	20	250	➢ 50000
	➢ AB	➢ 30	500	var.
	➢ BA	40	➢ var.	
	➢ BB	50		
		60		
		70		
		80		
		90		
		100		
		var.		

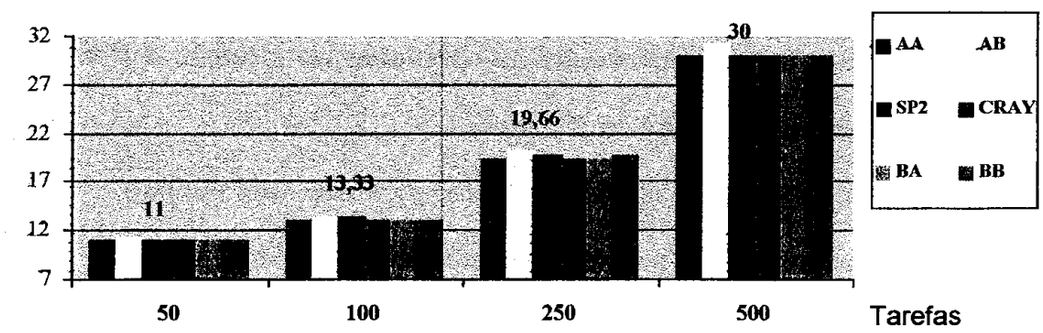


Figura C.3 Gráfico Comparativo para Tempo de Processamento da Simulação

A quantidade de código (eventos) a ser executado na simulação é o principal fator que determina o valor do tempo de processamento da simulação. Quanto maior o número de mensagens nulas, maior será o número de eventos a serem executados e , portanto, maior o tempo de processamento da simulação.

Deve-se considerar que existem eventos em uma simulação que não alteram o relógio dos processos envolvidos na simulação.

Por exemplo, uma tarefa pode ser escalonada para ser processada no tempo 0.0. Isto significa que, no modelo simulado, aquela ação que será realizada pela tarefa é imediata e por isso não acumulará nenhuma fração de tempo ao tempo total de simulação. Porém, para que a tarefa realize esta ação algumas linhas de código deverão ser executadas e portanto algum tempo (de processamento) será empregado nesta execução.

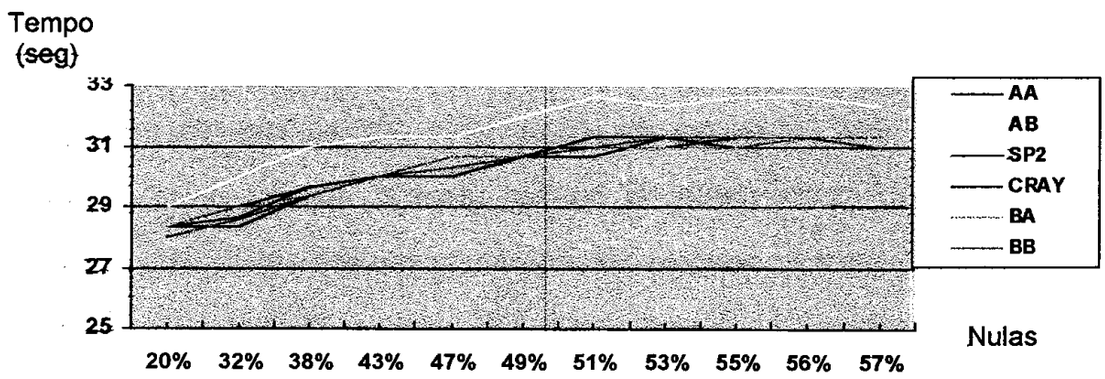


Figura C.4 Gráfico Comparativo para Tempo de Processamento da Simulação em Relação às Mensagens Nulas: Modelo Sistema Computacional Simplificado.

O aumento da quantidade de mensagens nulas, provocado pelo aumento da probabilidade de bloqueio, causou o aumento do tempo de processamento das simulações (Figura C.4). Este comportamento foi verificado em todos os modelos utilizados.

C.4 Utilização da CPU e as Mensagens Nulas

Os gráficos que fornecem o período de utilização de um recurso SMPL nas simulações auxiliam no processo de validação das simulações realizadas.

Uma das características principais de qualquer protocolo de sincronização para simulações distribuídas é a não participação nos acontecimentos dos eventos do modelo a ser simulado. No caso do protocolo CMB as mensagens nulas não podem afetar o andamento da simulação do modelo considerado. Os gráficos das figuras C.5 e C.6 mostram que os valores de utilização do recurso cpu (do modelo da aplicação), independente do modelo utilizado, foram muito próximos indicando que a modelagem das arquiteturas representativas e do protocolo CMB não alterou a simulação do modelo proposto. Ou seja, as mensagens nulas não tiveram participação nos resultados estatísticos obtidos na simulação dos modelos do servidor central e sistema computacional, conforme as especificações da proposta original do protocolo de sincronização CMB.

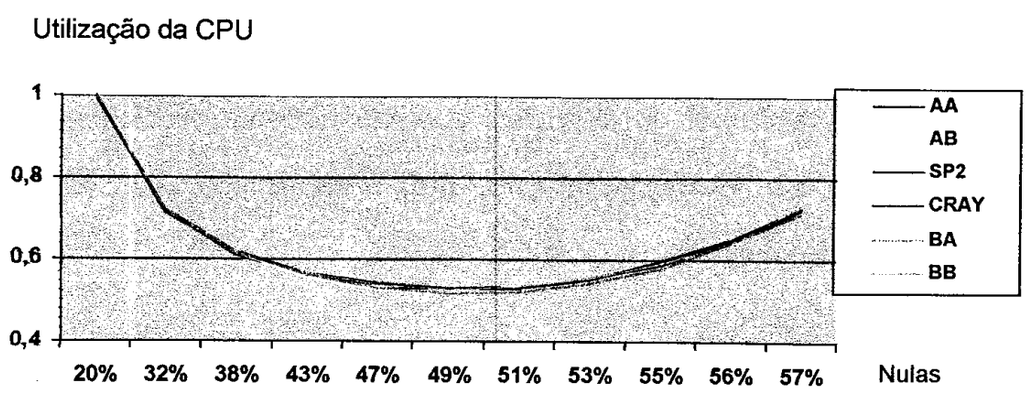


Figura C.5 Utilização da *cpu* : Modelo do servidor central

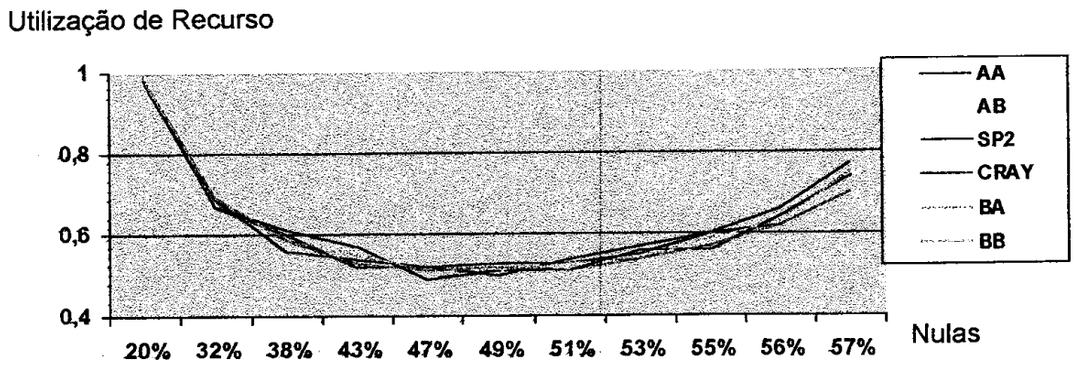


Figura C.6 Utilização da *cpu* : Modelo do sistema computacional

Referências Bibliográficas

ALMASI, G.S.; GOTTLIEB, A. *Highly Parallel Computing*, The Benjamin/Cummings Publishing Company, 1989.

ALMASI, G. S.; GOTTLIEB, A. *Highly Parallel Computing*. 2a. ed., The Benjamin Cummings Publishing Company, Inc. 1994.

AMORIM, C. L.; BARBOSA, V. C.; FERNANDES, E. S. T. Uma Introdução à Computação Paralela e Distribuída. VI Escola de Computação, 1988.

ANDREWS, G. R.; SCHNEIDER, F. B. Conceps and Notations for Concurrent Programming. Computing surveys, v.15, n.1, março 1983.

AYANI, R.; BERKMAN, B. Parallel Discrete Event Simulation on SIMD Computers. *Journal of Parallel and Distributed Computing*, v.18, p.501-508, 1993.

AYANI, R. Parallel Simulation. *Lecture Notes in Computer Science*, 1993.

BAGRODIA, R. I. Perils and Ptfalls of Parallel Discret-Event Simulation. In: PROCEEDINGS OF THE 1996 WINTER SIMULATION CONFERENCE, 1996.

BAGRODIA, R.; MEYER, R.; TAKAI, M.; CHEN, Y.; ZENG, X.; MARTIN, J.; SONG, H. Y. Parsec: a Parallel Simulation Environment for Complex Systems. *IEEE Computer*, v.31, n.10, p.77-85, 1998.

BAL, H. E.; STEINER, J. G.; TANENBAUM, A.S. Programming Languages for Distributed Systems. *ACM Computing Surveys*, v.21, n.3, p. 261-322, set., 1989.

BIRREL, A.; NELSON, B. J. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, v.2, n.1, p.39-59, fev. 1984.

BLECH, R. A. *An Overview of Parallel Processing*. Slides. Parallel Computing with PVM Workshop, Nasa Lewis Research Center, 1994. (http://www/lerc.nasa.gov/Other_Groups/IFMD/2620/tutorialPP.html).

CHANDY, K. M.; MISRA, J. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* SE-5, n.5, p. 440-452, set., 1979.

CHANDY, K. M.; MISRA, J. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Communication ACM*, vol.24, n.4, p. 198-205, abr. 1981

DAMIAN, A. *An Object Oriented Framework for the Simulation of Network Models*. Thesis of Master Science, Department of Computer Science, Calgary, Alberta, Canada, 1987.

DAS, S. R. Adaptive Protocols for Parallel Discrete Event Simulation. In: PROCEEDINGS OF THE 1996 WINTER SIMULATION CONFERENCE, dec. 8-11,1996.

DONGARRA, J. J. *Performance of Various Computers Using Standard Linear Equations Software* . US Department of Energy, may. 24, 2000 .

DUNCAN, R. A Survey of Parallel Computer Architectures. *IEEE Computer*, p.5-15, 1990.

FERSCHA, A.; LUTHI, J.; Estimating Rollback Overhead for Optimism Control in Time Warp. In: PROCEEDINGS OF THE 28TH ANNUAL SIMULATION SYMPOSIUM, 1995.

FLYNN, M. J. Some Computer Organization and Their Efetiveness. *IEEE Transactions on Computer*; c-21, p.948-60, 1972.

FOSTER, Y. *Designing and Bulding Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Publishing Company Inc., 1995.

FUJIMOTO, R. M. Lookahead in Parallel Event Simulation. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, St Charles I 11, p. 34-41, 1988.

FUJIMOTO, R. M. Performance of Time Warp Under Synthetic Workloads. In: PROCEEDINGS OF THE WESTERN MULTICONFERENCE - DISTRIBUTED SIMULATION, Society fo Computer Simulation (SCS) , San Diego, California, p. 23-28, 1990.

FUJIMOTO, R. M. Parallel Discrete Event Simulation. *Comunications of the ACM*, vol. 33, n.10, out., 1990.

FUJIMOTO, R. M. Parallel and Distributed Simulation Systems. *Willey Series on Parallel and Distributed Computing.* Albert Y. Zomaya, Series Editor Wiley - Interscience,2000.

GARCIA, N.; *Distributed Simulation With Relaxed Timing.* Thesis of Master Science, University of Manchester , Manchester, 1991.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAN, V. *PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAN, V. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory,1994.

GRILLO, R. C. M. Simulação na Construção de Protótipos de Sistemas Distribuídos, *XXII Congresso Nacional de Informática,* 1989.

HWANG, K.; BRIGGS, F. A. *Computer Architecture and Parallel Processing.* McGraw-Hill International Editions, 1984.

HWANG, K. *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill International Editions, 1993.

HWANG, K.; XU, Z. *Scalable Parallel Computing – Technology, Architecture, Programming.* McGraw-Hill Companies, 1998.

JEFFERSON, D. R. Virtual Time. *ACM Transactions Prog. Lang. Systems,* vol. 7, n.3, p. 404-425, jul. 1985.

JHA, V.; BAGRODIA, R.; A Unified Framework for Conservative and Otimistic Distributed Simulation. In: PROCEEDINGS 8TH WORKSHOP ON

PARALLEL AND DISTRIBUTED SIMULATION, IEEE Computer Society Press, p. 12-19, 1994.

LOW, Y.; LIM, C.; CAI, W. ; HUANG, S.; HSU W.; JAIN, S. TURNER, S.; Survey of Languages and Runtime Libraries for Parallel Discret-Event simulation. *Simulation*, vol. 72:3, p. 170-186, 1999.

MAcDOUGALL, M.H. *Simulating Computing Systems Techniques and Tools.* The MIT Press, 1987.

MARENZONI, P. Benchmark kernels as a tool for performance evaluation of MPP's. *Concurrency: Practice and Experience*, Università di Parma, 1998.

MCBRYAN, O. A. An Overview of Message Passing Environments. *Parallel Computing*, v.20, n.4, p.416-444, 1994.

MISRA, J., Distributed Discrete-Event Simulation. *ACM Computing Surveys*, 18(1), mar. 1986.

MORSELLI, J. C. M. Jr.; ULSON, R. S. ; SANTANA, R. C.; SANTANA, M. J.; An Approach for Dynamic Swapping of Distributed Simulation Synchronisation Protocols. IN: PROCEEDINGS OF THE ESS 2000 , 28-30 aug. 2000, Hamburgo, Alemanha..

NAVAUX, P. O. A. Introdução ao Processamento Paralelo. *Revista Brasileira de Computação*, v.5, n.2, p.31-43, 1989.

NICOL, D.; FUJIMOTO, R. *Parallel Simulation Today*,1995. Technical Report (http://www.cs.wm.edu/~subhas/parsim.html)

NICOL, D. M. Principles of Conservative Parallel Simulation. In: PROCEEDINGS OF THE 1996 WINTER SIMULATION CONFERENCE, p.128-135, 1996.

PENA, A. L. A. Desempenho do Sistema de Comunicação de uma Arquitetura Paralela Baseada em Transputers, em Ambiente Distribuído, minidissertação apresentada ao ICMSC - USP para o exame de Qualificação , ago. 1993.

QUEALY, A. *An introduction to Message Passing.* Slides, NASA LewisResearchCenter,1994. (http://www.lerc.nasa.gov/Other_Groups/IFMD/2620/tutorialPP.html)

QUINN, M. J. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill series in Supercomputing and Artificial Intelligence. McGraw-Hill Inc., 1987.

SANTANA, R. H. C.; SANTANA, M. J.; SOUZA, P. S. L.; SOUZA, M. A.; PIERKARSKI, A. E. T. *Computação Paralela.* Notas de Aula. ICMC-USP, 1997.

SOUZA, M. A. *Avaliação das Rotinas de Comunicação Ponto-a-Ponto do MPI.* Dissertação (Mestrado), ICMC-Universidade de São Paulo, 1996.

SPOLON, R.; SANTANA, M. J.; SANTANA, R. H. C.. Distributed Simulation, Time Warp and is Variants: Taxonomy and Performance Evaluation Issues. In: PROCEEDINGS 13TH EUROPEAN SIMULATION MULTICONFERENCE, p. 220 – 227. The Society for Computer Simulation International, Polonia, 1999.

STEINMAN, Jeff. Scalable Parallel and Distributed Military Simulations Using the Speedes Framework. *In: ELECSIM 96 INTERNET CONFERENCE*, 1995.

STTOTS Jr., P. D. A Comparative Survey of Concurrent Programming Languages. *ACM Sigplan Notes*, v.17, n.9, p. 76-87, set., 1982.

TANENBAUM, A. S. *Modern Operating Systems.* Prentice-Hall International Editions, 1992.

TANENBAUM, A. S. *Distributed Operating Systems*. Prentice-Hall, Inc. 1995.

TRINDADE, O. Jr. *Servidor de Processamento Paralelo Baseado em Banco de Transputers.* Tese de Doutorado, ICMC-USP, 1991.

ULSON, R. S. *Simulação Distribuída em Plataformas de Portabilidade: Viabilidade de Uso e Comportamento do Protocolo CMB*. Tese de doutorado, IFSC - Universidade de São Paulo, 1999.

ULSON, R. S. ; MORSELLI, J. C. M. ; SANTANA, R. C.; SANTANA, M. J.; Conservative Distributed Simulation on Portability Plataforms: The CMB Protocol Behavior. PROCEEDINGS OF THE IASTED INTERNATIONAL CONFERENCE APPLIED MODELING AND SIMULATION, sep. 1-3, 1999, Cairns, Australia.