Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

Predição de Desempenho de Aplicações Paralelas para Máquinas Agregadas Utilizando Modelos Estocásticos

Lucas Janssen Baldo

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação.

Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes

Porto Alegre, junho de 2007.

PUCRS

Campus Central
Av. Ipiranga, 6681 – prédio 16 – CEP 90619-900
Porto Alegre – RS – Brasil
Fone: +55 (51) 3320-3544 – Fax: +55 (51) 3320-3548
Email: bceadm@pucrs.br
www.pucrs.br/biblioteca

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Predição de Desempenho de Aplicações Paralelas para Máquinas Agregadas Utilizando Modelos Estocásticos***", apresentada por Lucas Janssen Baldo, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 12/12/2006 pela Comissão Examinadora:

Prof. Dr. Luiz Gustavo Leão Fernandes – Orientador — PPGCC/PUCRS

Prof. Dr. Fernando Luís Dotti — PPGCC/PUCRS

Prof. Dr. Paulo Henrique Lemelle Fernandes — PPGCC/PUCRS

Prof. Dr. Nicolas Maillard — UFRGS

Homologada em 04/01/07, conforme Ata No. 001 pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P. 16 – sala 106 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@inf.pucrs.br
www.pucrs.br/facin/pos

Agradecimentos

Primeiramente gostaria de agradecer a todos os meus amigos e familiares que me apoiaram e me suportaram durante os últimos dias de escrita deste trabalho. Gostaria de agradecer também a Deus que sempre me mostrou os caminhos que minha vida deveria seguir.

Ao pessoal do meu projeto (CAP), por terem me ajudado muito no desenvolvimento deste trabalho e me ensinado muito como trabalhar em grupo. Thiago, gostaria de te agradecer pelo teu apoio desde o nosso início, na pequena sala. Continue sendo um cara dedicado como tu és, que com certeza colherás frutos. Matheus, obrigado pela ajuda nas tantas figuras que eu precisava. Teu talento sempre será reconhecido, e com certeza lhe trará sucesso. Márcio, valeu por teres me ensinado muitas coisas, principalmente no latex. Tu és uma pessoa que com certeza terá um ótimo futuro profissional, não só pelos teus conhecimentos, mas pelo teu jeito de ser. Enfim, a todos vocês três eu deixo aqui o meu muito obrigado, me sentindo honrado de ter conhecido vocês.

Existem duas pessoas que com certeza faltarão palavras para explicar o quanto eles foram pra mim nestes últimos anos. Pedro, que foi meu colega e amigo desde o início da faculdade, muito obrigado por tudo que nós passamos. Foram muitos momentos em que podemos aprender um com o outro e nos divertir. Espero que essa nossa amizade dure para sempre. Te desejo o maior sucesso do mundo, que tua experiência na França seja ótima. Gustavo, meu orientador, posso te dizer que nunca tinha conhecido uma pessoa tão responsável com tu. Tua amizade e teu companherismo são cativantes. Cara, tu foi o responsável por eu estar aqui, escrevendo estes agradecimentos. Muito obrigado por tudo. Tenho certeza que sempre seremos amigos.

Minha família querida, mãe, pai, irmã. Valeu pelo apoio que vocês me deram. Sem ajuda de vocês, seria quase impossível chegar lá. Que nossa família seja unida para sempre. Que nossa alegrias sejam compartilhadas entre nós. Que o amor e a saúde estejam sempre entre nós. Fê, meu amor, muito obrigado por estar sempre do meu lado. Teu carinho foi essencial nos dias de tensão. Amo todos vocês!

Por fim, acho acho que na vida existem coisas que a gente perde, e coisas que a gente ganha. Mas ela é uma só. E é para ser vivida.

Resumo

Um dos maiores problemas na área de computação de alto desempenho é a dificuldade de definir qual a melhor estratégia de paralelização de uma aplicação. Neste contexto, a utilização de métodos analíticos para a avaliação de desempenho de aplicações paralelas aparece como uma alternativa interessante para auxiliar no processo de escolha das melhores estratégias de paralelização. Neste trabalho, propõe-se a adoção do formalismo de Redes de Autômatos Estocásticos para modelar e avaliar o desempenho de aplicações paralelas especialmente desenvolvidas para máquinas agregadas (i.e., *clusters*). A metodologia utilizada é baseada na construção de modelos genéricos para descrever esquemas clássicos de implementação paralela, tais como Mestre/Escravo, Fases Paralelas, *Pipeline* e Divisão e Conquista. Estes modelos são adaptados em casos de aplicações reais através da definição de valores para parâmetros de entrada dos modelos. Finalmente, com intuito de verificar a precisão da técnica de modelagem adotada, comparações com resultados de implementações reais são apresentadas.

Abstract

One of the main problems in the high performance computing area is the difficulty to define which is the best strategy to paralelize an application. In this context, the use of analytical methods to evaluate the performance behavior of such applications seems to be an interesting alternative and can help to identify the best implementation strategies. In this work, the Stochastic Automata Network formalism is adopted to model and evaluate the performance os parallel applications, specially developed for clusters of workstations platforms. The methodology used is based on the construction of generic models to describe classical parallel implementation schemes, like Master/Slave, Parallel Phases, Pipeline and Divide and Conquer. Those models are adapted to represent cases of real applications through the definition of input parameters values. Finally, aiming to verify the accuracy of the adopted technique, some comparisons with real applications implementation results are presented.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

IRM	Imagens de Ressonância Magnética	15
SAN	*Stochastic Automata Networks*	16
COW	*Cluster of Workstations*	16
NOW	*Network of Workstations*	19
TPN	*Temporized Petri Nets*	20
PEPS	*Performance Evaluation of Parallel Systems*	23
CAP	Centro de Pesquisa e Desenvolvimento de Aplicações Paralelas	38
FED	*Field Emission Display*	38
FOP	*Format Object Processing*	38
CRT	*Cathode Ray Tube*	48
PDF	*Portable Document Format*	52
PS	*Post Script*	52
SVG	*Scalable Vector Graphics*	52
XSL-FO	*Extensible Style Sheet Language - Formatting Objects*	52

Lista de Algoritmos

Sumário

Capítulo 1

Introdução

Nas últimas três décadas, a computação de alto desempenho tem se tornado uma área de pesquisa crescente da Ciência da Computação. A razão para isto está relacionada ao grande crescimento das necessidades de poder computacional de outras áreas de pesquisa, como por exemplo Biologia, Física, Medicina, Geologia, Química, Astronomia, Engenharia, *etc.* Todas essas disciplinas apresentam pesquisas de estado da arte baseadas em simulação ou técnicas de visualização as quais requerem muito mais poder computacional do que um computador monoprocessado pode prover. Três exemplos representativos correntes deste cenário são a análise de genomas [28], predição de terremoto [26] e visualização de dados médicos obtidos através de Imagens de Ressonância Magnética (IRM) [19].

Soluções em computação de alto desempenho são fortemente dependentes da plataforma alvo. Existem diferentes plataformas para executar programas de alto desempenho e diversas formas de desenvolver programas eficientes para estas plataformas. Podemos citar, por exemplo, máquinas multiprocessadas muito velozes, porém com um custo muito alto, até arquiteturas do tipo grade completamente distribuídas, passando por supercomputadores vetoriais ou máquinas agregadas. Além disso, muitas dessas máquinas são extremamente complexas de controlar, adicionando um custo de manutenção estrutural àquele de sua aquisição.

Neste cenário, máquinas agregadas começaram a ganhar popularidade desde 1995 com o projeto Beowulf [11]. A idéia inovadora trazida por este projeto foi a abordagem de utilização de recursos de "prateleira" para criar ambientes de alto desempenho. Agregados do projeto Beowulf podem ser definidos como uma coleção de processadores dedicados ao processamento paralelo (diferente de estações de rede) comunicando-se através da mais veloz e eficiente interconexão acessível. Este projeto foi o ponto de começo de uma massiva difusão de arquiteturas de máquinas agregadas no mundo científico. Devido a este baixo custo, ambientes agregados têm se tornado uma alternativa interessante para alcançar alto desempenho com um agrupamento de centenas ou até milhares de nós.

1.1 Motivação

As aplicações que se pretende executar em ambientes do tipo máquinas agregadas devem ser programadas utilizando-se o paradigma de troca de mensagens, uma vez que a memória utilizada não é compartilhada entre os processadores. Assim, saber se uma aplicação obterá um desempenho desejado em máquinas agregadas é uma questão que hoje somente pode ser respondida quando esta aplicação for executada, pois existem diversos fatores influentes no desempenho da mesma tais como o modelo de programação escolhido, a granularidade das tarefas ou até mesmo a rede utilizada. Além disso, diversos modelos de programação surgiram no momento em que ambientes do tipo *cluster* começaram a ser utilizados. Com isso, aplicações podem ser desenvolvidas de diversas formas. Porém, saber de antemão qual modelo é o que melhor enquadra uma aplicação é outra pergunta que somente pode ser respondida através da implementação.

Neste cenário, a predição de desempenho de programas de alto desempenho surge como uma interessante alternativa para otimizar os processos de desenvolvimento de aplicações. A análise de desempenho é baseada na construção de modelos genéricos, usando algum formalismo, para descrever cada método de implementação. Como resultados de tal análise, deve ser possível prever o desempenho de uma dada implementação em diversos níveis, variando desde probabilidades de transmissão, porcentagem de uso dos nós, e até estimativas de tempo de execução.

1.2 Objetivos

Nos últimos anos, um novo formalismo baseado em Redes de Autômatos Estocásticos (*Stochastic Automata Network* - SAN) tem sido utilizado para descrever estruturas interdependentes complexas [9, 15, 20, 22]. Este trabalho tem como objetivo realizar uma investigação aprofundada sobre a utilização do formalismo SAN para a modelagem e análise de desempenho de aplicações paralelas que se adaptem a ambientes do tipo *Cluster of Workstations* (COW). A escolha do formalismo SAN baseia-se na dificuldade de encontrar outros formalismos que sejam capazes de representar comportamentos independentes de diferentes módulos que se relacionem entre si.

1.3 Estrutura do Trabalho

Este trabalho está dividido como segue. No próximo capítulo, alguns trabalhos relacionados são discutidos, apresentando a idéia básica de cada um e mostrando a diferença de cada formalismo estudado. O capítulo 3 apresenta uma breve descrição do formalismo escolhido para ser utilizado ao longo deste trabalho. O capítulo 4 introduz os modelos de programação para aplicações paralelas mais utilizados e apresenta os seus respectivos mo-

delos SAN genéricos (*i.e.*, autômatos e estados sem taxas). As formas de parametrização desses modelos genéricos são descritas no capítulo 5. Quatro aplicações escolhidas como estudo de caso são detalhadas no capítulo 6. A análise e comparação dos resultados obtidos com a execução dos modelos e das aplicações paralelas é realizada na seção 7. Por fim, a conclusão deste trabalho e possíveis trabalhos futuros são apresentados no capítulo 8.

Capítulo 2

Trabalhos Relacionados

Nesta seção, alguns trabalhos relacionados são apresentados. Primeiramente, três trabalhos que utilizam SAN para modelar e avaliar desempenho de suas aplicações são discutidos. Mais adiante, outros formalismos que poderiam ser utilizados para modelagem de aplicações paralelas são apresentados juntamente com aspectos sobre a escolha de SAN para este trabalho.

2.1 Trabalhos que utilizam SAN

Nos últimos anos, formalismos estocásticos para predição de desempenho de aplicações têm sido utilizados com mais freqüência. Em [10], utiliza-se SAN para analisar interações entre agentes com sistemas baseado em componentes. Estes modelos criados permitiram aos autores descobrir detalhes significantes no desempenho de agentes. Mais precisamente, com modelos SAN os autores conseguiram atribuir corretamente diferenças de desempenho entre agentes ou entre um agente e o ambiente em que ele reside. É da opinião dos autores que o formalismo utilizado é flexível, fácil de usar e que pôde prover uma variedade de percepções no comportamento de agentes modelados.

Maraculescu e Nandi [20] usaram SAN para modelar aplicações de sistemas de níveis. Este trabalho teve como objetivo apresentar uma nova metodologia para modelagem de aplicações para análise de desempenho de sistemas de níveis, a qual pode auxiliar o desenvolvedor a escolher a plataforma correta para implementação de um conjunto de aplicações multimídia. O decodificador de video MPEG-2 foi a aplicação utilizada para ilustrar os benefícios desta metodologia. Com o modelo SAN, os autores conseguiram mostrar que a aplicação alvo possui um comportamento estacionário através de diferentes combinações de probabilidades, o que permitiu mapeamentos eficientes da aplicação na plataforma escolhida. Ainda, os autores consideraram SAN uma ferramenta eficiente para modelagem de comunicação de processos, e ressaltaram que este formalismo possui vantagens em relação a outros quanto à explosão no número de estados, pois SAN não gera a cadeia Markoviana.

Nos últimos anos, serviços como vídeo ou multimídia em redes de baixa latência têm se tornando importantes e muito usados. Em [22], Mokdad et. al. apresentam um novo algoritmo de roteamento para prover uma melhor qualidade de serviço de entrega de pacotes em redes de baixa latência. Neste trabalho, uma avaliação de desempenho de modelos SAN é realizada para mostrar os benefícios deste novo algoritmo. Os autores lembram que o formalismo SAN é muito utilizado para sistemas paralelos complexos, situação em que o emprego de cadeias de Markov é inviável.

2.2 Outros Formalismos

Muitos autores têm apresentado estudos genéricos oferecendo opções de predição de desempenho de aplicações paralelas [16, 17]. A comunidade de pesquisa classifica as diferentes abordagens em três grupos bastante distintos:

- As abordagens de monitoramento, as quais são na maioria das vezes baseadas na comparação de custos de tempo de execuções de implementações; Estas implementações podem ser tão superficiais quanto programas artificiais [16], ou bastante genéricos como os reconhecidos *benchmarks*, *e.g.*, *whetstones*, *dhrystones*, e *LINPACK*;

- As abordagens de simulação, as quais são baseadas no uso de uma ferramenta computacional para descrever e simular o comportamento de uma dada implementação; As ferramentas de simulação são geralmente baseadas em gerações aleatórias de escolhas do programa, mas também técnicas mais sofisticadas como *Perfect Simulation* [18] podem ser utilizadas;

- As abordagens de modelagem analítica, as quais são empregadas mais raramente em predições de programas paralelos; Os mais conhecidos formalismos da modelagem analítica, *e.g.*, Cadeias de Markov [27] e Redes de filas de espera [13], não são muito apropriados para representar paralelismo e sincronização por não possuírem primitivas que possam corresponder a estes fatores.

Como dito anteriormente, a predição de desempenho de aplicações paralelas tem sido uma boa alternativa para escolha de melhores parâmetros ideais como plataforma, número de nós e distribuição de carga. Nos últimos anos, muitos estudos revelam diferentes metodologias para predição de desempenho desta classe de aplicações. Diferente de trabalhos anteriores que são muito específicos [2, 29] ou requerem uma descrição detalhada da aplicação alvo (algumas vezes até o código fonte) [17, 31], neste trabalho foi utilizado o formalismo SAN para gerar modelos genéricos para aplicações paralelas que sejam executáveis em máquinas agregadas (*clusters*).

Em 1996, Yan, Zhang e Song [31] propuseram um modelo de predição de desempenho para redes de computadores *Network of Workstations* (NOW) heterogêneos não-dedicados.

Neste trabalho, os autores conseguiram prever tempo de execução, aceleração e eficiência de diferentes aplicações paralelas. Para esta predição de desempenho, foi utilizado uma abordagem baseada em um modelo de dois níveis. O primeiro nível utilizou um grafo de tarefas semi-determinístico para capturar comportamentos das execuções paralelas, incluindo comunicação e sincronização. O segundo nível usou um modelo de tempo discreto para quantificar os efeitos do agregado tipo NOW. Um processo iterativo foi usado para determinar os efeitos interativos entre estes dois níveis. Apesar de Yan et. al. conseguirem apresentar resultados interessantes, viu-se que é necessário o conhecimento de boa parte do código da implementação para conseguir tais resultados, *i.e.*, a utilização desta técnica de predição de desempenho só é valida quando o desenvolvimento de uma aplicação encontra-se já em fase de implementação. Ainda, percebeu-se que muitos parâmetros de cada nó devem ser considerados, como memória, número de instruções de ponto flutuante, etc.

Dois anos mais tarde, Anglano [2] apresentou uma metodologia para estimativa do tempo de execução de aplicações através de modelos de Redes de Petri Temporizados (*Temporized Petri Nets* - TPN). Neste trabalho, Anglano utiliza uma mistura de análise empírica e modelagem analítica para representar os efeitos da disputa de rede de máquinas agregadas não-dedicados. Apesar deste trabalho apresentar resultados positivos quanto a predição de desempenho da aplicação alvo (multiplicação de matrizes), observou-se que os modelos são muitos específicos à aplicação, *i.e.*, para cada aplicação a ser executada em uma máquina agregada um novo modelo deve ser gerado. Ainda, constatou-se que muitos detalhes de implementação devem ser conhecidos *a priori*, como por exemplo o número de iterações de um laço "*for*".

A utilização de modelos estocásticos para predição de desempenho utilizando o formalismo SAN parece lidar com aspectos que não são tratados, ou são às vezes muito específicos em outros formalismos estudados. Por exemplo, com modelos SAN não é necessário o conhecimento do código de uma implementação, *i.e.*, é possível construir modelos SAN e realizar uma análise com sucesso ainda na fase de modelagem de uma aplicação. Outro ponto chave na escolha deste formalismo é a abstração de muitos parâmetros dos nós envolvidos na computação. Em contraste ao trabalho apresentado em [2], neste trabalho são apresentados modelos SAN genéricos que podem ser adaptados a diferentes máquinas agregadas. Outro motivo que justifica a escolha do formalismo SAN - diferentemente de outros formalismos tais como Redes de Petri e Redes de fila de espera - é o fato deste permitir uma modelagem mais apropriada da comunicação entre processos independentes, representando de maneira mais clara o funcionamento de aplicações paralelas. Ainda, para resolver modelos SAN utiliza-se distribuição exponencial, não necessitando assim parâmetros de entrada com valores absolutos. Em outras palavras, a parametrização de modelos SAN é realizada através de valores aproximados balisados por uma distribuição exponencial que permitem a geração de resultados bem próximos do esperado.

Capítulo 3

Redes de Autômatos Estocásticos

Redes de Autômatos Estocásticos é um formalismo que surgiu na década de 80 para modelar sistemas, especialmente aplicações paralelas, baseado na teoria de Cadeias de Markov. Também chamada de SAN, essa técnica permite que modelos *markovianos* possam ser descritos de forma compacta e eficiente. SAN permite inferir medidas de desempenho antes da implementação, como por exemplo tempo de resposta, *throughput*, atraso de sincronização efetivo, e outras características relacionadas à predição de desempenho em aplicações paralelas. Este capítulo apresenta apenas uma breve descrição do formalismo SAN. O leitor interessado em maiores detalhes sobre o mesmo poderá consultar [9, 15, 20, 22, 24, 25].

3.1 Descrição do Formalismo

No contexto deste trabalho, troca de mensagens será considerado como um modelo de execução, já que este é capaz de avaliar o comportamento de uma aplicação paralela, desde que modelos de máquina e de programação sejam pré-definidos. A troca de mensagens nos modelos SAN são modeladas através do uso de eventos sincronizantes.

A idéia básica das Redes de Autômatos Estocásticos é que esta descreva um modelo global de um sistema em diversos subsistemas (submodelos) quase independentes entre si, onde cada dois ou mais submodelos interagem somente em alguns casos. Estes subsistemas, definidos como autômatos estocásticos, são caracterizados por três aspectos: estados, transições e eventos.

Estados são definidos como um conjunto de comportamentos do sistema, onde cada estado corresponde a um comportamento específico. Estes são indicados como um círculo em um modelo SAN. Uma transição é uma mudança de um estado para outro, ou seja, a mudança de comportamento do sistema. Transições são representadas como um arco em modelos SAN, onde este arco vai de um estado origem a um estado destino. Um subsistema modifica seu estado somente se um evento ocorrer, definido por uma função de probabilidade. Por exemplo, na Figura 1 temos dois autômatos $A^{(1)}$ e $A^{(2)}$. Este

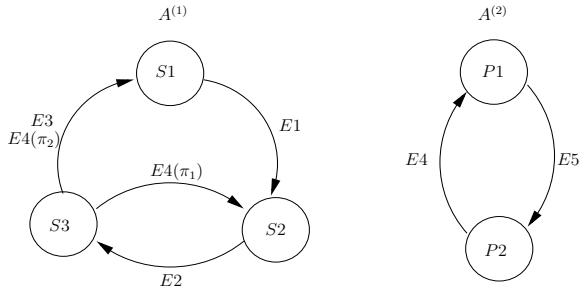


Figura 1: Exemplo 1: rede de autômatos estocásticos.

primeiro autômato estocástico tem três estados ($S1$, $S2$, $S3$) e quatro eventos ($E1$, $E2$, $E3$ e $E4$). O segundo autômato possui dois estados ($P1$ e $P2$) e dois eventos ($E4$ e $E5$).

Dois tipos de eventos podem ser encontrados em uma SAN: locais e sincronizantes. Eventos locais modificam somente o estado de um submodelo. Por outro lado, eventos sincronizantes modificam dois ou mais estados em dois ou mais autômatos. Ainda observando a Figura 1, o evento $E2$ é um exemplo de um evento local. Já $E4$ é um evento sincronizante. O evento $E4$ é considerado sincronizante, pois modifica o estado dos autômatos $A^{(1)}$ e $A^{(2)}$. No autômato $A^{(1)}$, o evento $E4$ pode modificar seu estado de duas formas: do estado $S3$ para o estado $S2$ com uma probabilidade π_1 e do estado $S3$ para o estado $S1$ com uma probabilidade π_2.

Eventos locais são aqueles que modificam somente o estado de um autômato estocástico. A Cadeia de Markov pode ser construída combinando todos os estados de todos os subsistemas. Como pode ser visto na Figura 2, este autômato possui 6 estados globais, ao contrário do modelo estocástico que possui, para o autômatos $A^{(1)}$ e $A^{(2)}$, 3 e 2 estados locais, respectivamente.

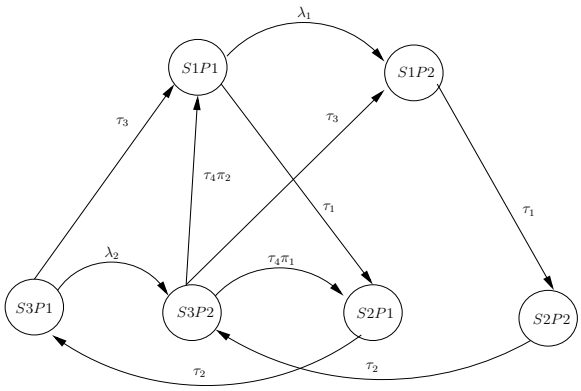


Figura 2: Cadeia de Markov equivalente ao modelo SAN do exemplo 1.

Esta possibilidade dá-se pelo fato do formalismo possuir primitivas paralelas e sincronizantes. Porém, a definição das taxas dos eventos de um modelo ainda é complexa, exigindo um grande conhecimento do formalismo e grande experiência de quem irá utilizá-

lo. Mesmo assim, acreditamos que é possível o uso de SAN para a modelagem de aplicações paralelas e servirá neste trabalho como validação das escolhas de implementação.

3.2 Análise Transiente

Os resultados apresentados neste trabalho foram obtidos através de modelos SAN executados na ferramenta PEPS (*Performance Evaluation of Parallel Systems*) [8]. Esta ferramenta possui métodos iterativos para resolução de sistemas de equações. Com o PEPS, dois tipos de análise podem ser realizadas sobre os modelos SAN gerados: análise estacionária e análise transiente. Esta segunda análise foi a adotada, pois com análise transiente consegue-se estimar o tempo médio de execução de aplicações modeladas. Por esta razão, ela foi adotada neste trabalho uma vez que em computação de alto desempenho o objetivo principal é observar o tempo de execução. O leitor com interesse nas diferentes soluções transientes pode procurar em [27].

A **análise transiente** foi implementada na ferramenta PEPS utilizando um modelo iterativo onde o usuário especifica o tempo que ele deseja executar seu modelo SAN. Com este tempo, o PEPS calcula o número de iterações necessárias para executar o modelo. Os modelos SAN devem possuir um ou mais estados absorvente, ou seja, um estado que não possui transições de saída. Neste trabalho o estado absorvente para cada modelo é o estado *Fim*. A figura 3 apresenta o exemplo de um modelo SAN com este estado absorvente.

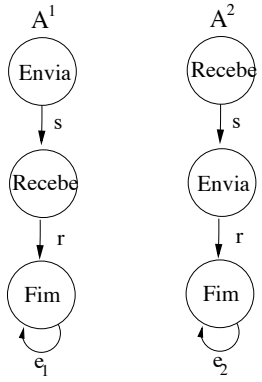


Figura 3: Exemplo de modelo SAN para análise transiente.

Este Modelo tem dois autômatos A^1 e A^2. A função deste modelo SAN é modelar a troca de dados entre o autômato A^1 para o autômato A^2, através dos eventos s e r. No momento que o evento r ocorre, a transmissão de dados é finalizada e os autômatos passam para o estado absorvente *Fim*.

Para executar um modelo SAN utilizando um método transiente, é necessário especificar o tempo de duração dessa execução. Com este tempo, o método transiente calcula o número de iterações a serem executadas. No final de uma execução, o método transi-

ente retorna um vetor de probabilidades de todos os estados do modelo SAN. Neste caso, analisa-se somente a probabilidade do estado absorvente (estado *Fim* da figura 3). Se na i-ésima execução do modelo SAN a probabilidade do estado *Fim* não estiver por exemplo em 99%, o vetor de probabilidades da i-ésima execução é recarregado pelo método transiente e uma nova execução é realizada, até que o critério seja satisfeito (probabilidade do estado *Fim* estar em 99%).

Desta forma, tem-se as probabilidades parciais em cada execução do modelo SAN. Com estas probabilidades e com os tempos de execução passados para o método transiente, pode-se calcular o tempo total de execução da aplicação modelada através de somas ponderadas dessas diversas durações. Em outras palavras, é realizado um somatório de sucessivas multiplicações entre a probabilidade e o tempo do método transiente. A tabela 3.2 apresenta um exemplo de como realizar o cálculo do tempo total (em segundos) de execução de uma aplicação modelada através das probabilidades parciais do estado absorvente.

Tabela 1: Tabela exemplo de cálculo do tempo de execução utilizando análise transiente

Tempo do método transiente	Número de iterações	Probabilidade	Tempo parcial (tp)
5	15	$0,1$	$(0,1-0)\times 5 = 0,5$
10	30	$0,35$	$(0,35-0,1)\times 10 = 2,5$
15	45	$0,59$	$(0,59-0,35)\times 15 = 3,6$
20	60	$0,73$	$(0,73-0,59)\times 20 = 2,8$
25	75	$0,87$	$(0,87-0,73)\times 25 = 3,5$
30	90	$0,91$	$(0,91-0,87)\times 30 = 1,2$
35	105	$0,93$	$(0,93-0,91)\times 35 = 0,7$
40	120	$0,95$	$(0,95-0,93)\times 40 = 0,8$
45	135	$0,97$	$(0,97-0,95)\times 45 = 0,9$
50	150	$0,99$	$(0,99-0,97)\times 50 = 1,0$
-	-	-	tempo total $= \sum_1^{10} tp = 17,5$

Observe que o tempo parcial na i-ésima execução é a diferença das probabilidades do estado i e $i-1$ multiplicados pelo tempo de execução parcial do modelo. É necessário realizar esta diferença entre as probabilidades para saber o quanto a probabilidade do estado absorvente evoluiu, uma vez que em cada execução o vetor de probabilidades é recarregado com valores da execução anterior. Assim, tem-se a evolução da probabilidade do estado absorvente multiplicado pelo tempo que o modelo SAN já executou, resultando em um tempo parcial. A soma de todos os tempos parciais resulta no tempo total que a aplicação modelada irá demorar pra executar.

Capítulo 4

Modelos SAN Genéricos para Máquinas Agregadas

Nesta seção, os modelos SAN genéricos para máquinas agregadas são apresentados focando na troca de dados e no tempo de processamento, uma vez que a relação entre essas duas características são o ponto chave para determinar o sucesso de implementações paralelas sobre a referida plataforma. Quatro modelos SAN são discutidos nesta seção: modelo mestre/escravo, modelo de fases paralelas, modelo *pipeline* e modelo divisão e conquista. Primeiramente, será realizada uma breve apresentação dos modelos de programação paralela escolhidos, seguido de uma descrição detalhada de cada modelo SAN.

4.1 Modelos de Programação Paralela

Diversos modelos de programação são encontrados hoje na literatura. Dentre eles, os que mais se destacam são os modelos mestre/escravo, fases paralelas, *pipeline* (também conhecido como produtor/consumidor), divisão e conquista, grafo de tarefas e decomposição geométrica [14, 21]. Os quatro primeiros foram escolhidos por serem os mais utilizados pela comunidade científica em aplicações desenvolvidas para máquinas agregadas e por apresentarem características diferentes entre eles, tais como comportamento dos processos e quantidade de informação trocada entre eles.

O modelo mestre/escravo tem como característica a presença de um processo coordenador responsável pela geração de trabalho e alocação destes para outros processos, denominados escravos. Neste modelo, o processo mestre pode utilizar uma técnica de balanceamento de carga para que todos os processos tenham tarefas de pesos próximos, evitando que processos escravos fiquem sub-carregados ou sobrecarregados (*i.e.*, processos ociosos). A utilização do modelo mestre/escravo deve ser realizada de forma que o processo mestre (centralizador) não se torne um gargalo, resultando assim em perda de desempenho.

O segundo modelo escolhido é o modelo de fases paralelas. Este padrão de desenvolvimento tem como características a divisão da execução em duas fases básicas: a fase de processamento e a fase de transmissão (sincronização). Neste modelo, todos os processos realizam processamento de tarefas, não existindo um processo centralizador. O balanceamento de carga neste caso é distribuído, ou seja, todos os processos executam um algoritmo de balanceamento de carga para definir suas tarefas. A fase de transmissão (troca de dados de todos para todos) é utilizada para sincronização de todos processos, onde estes mudam de fase ao mesmo tempo. A desvantagem deste modelo está no alto custo de comunicação, uma vez que todos os processos devem trocar informações si.

No modelo *pipeline*, o conjunto de tarefas é passado entre processos sucessivamente e cada processo realiza a computação de uma tarefa. Neste modelo, os processos são organizados em forma de fila, onde um processo recebe tarefas de seu predecessor, executa-as, e as envia para seu sucessor. O modelo *pipeline* é o mais difícil de aplicar devido às características de decomposição do problema em etapas quase seqüenciais. Mesmo assim, este modelo foi escolhido por possuir características diferentes dos outros.

Por fim, o último modelo escolhido é o modelo de divisão e conquista. Neste modelo, um problema é dividido em sub-problemas que são resolvidos independentemente e seus resultados são unificados em uma fase final. No modelo divisão e conquista, unidades (processos) são agrupadas em uma hierarquia de árvore. Os processos pais dividem suas tarefas e repassam uma parte para seus filhos. Os resultados são integrados recursivamente. A figura 4 apresenta um exemplo dessa hierarquia com 4 processos, indicando como é realizada a divisão de trabalho e comunicação neste modelo.

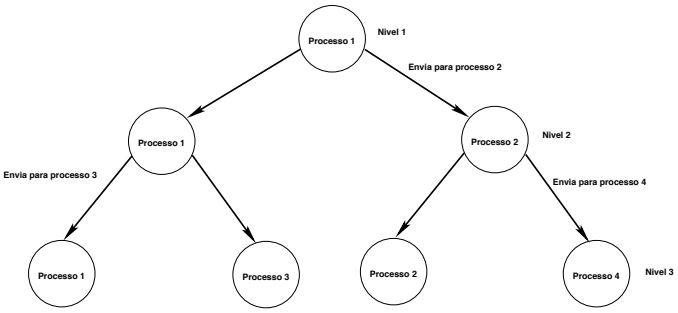


Figura 4: Organização dos processos no modelo divisão e conquista.

Nesta figura, percebe-se que todos os processos chegam ao último nível, no qual é realizado o processamento. Após o processamento, todos os processos que receberam tarefa devem retornar o resultado para o processo enviador (denominado processo pai). Assim, uma série de transferência de dados é realizada até que o processo que está no primeiro nível da árvore contenha todos os resultados. Desta forma, todos os processos realizam trabalhos paralelamente e somente na fase de unificação dos resultados que os processos enviam seus resultados para níveis superiores da árvore.

4.2 Modelo SAN para Mestre/Escravo

O modelo SAN para mestre/escravo é mostrado na figura 5. Este modelo contém um autômato *Mestre*, um autômato *Tarefas*, e P autômatos $Escravos^{(i)}$ $(i = 1..P)$.

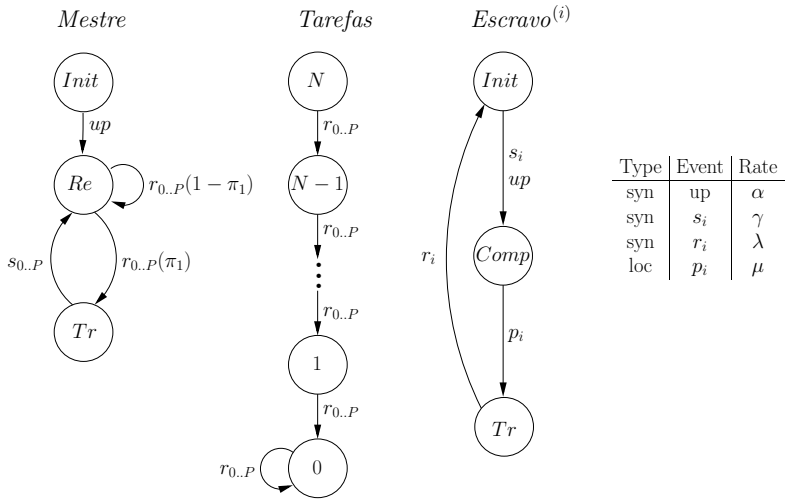


Figura 5: SAN genérico para mestre/escravo.

Para esta abordagem (figura 5), o nó mestre é responsável pela distribuição de trabalho e armazenamento dos resultados das tarefas processadas pelos escravos. No modelo correspondente, o autômato *Mestre* possui três estados (*Init*, *Re* e *Tr*) que significam, respectivamente, o estado inicial, a recepção dos resultados das tarefas enviadas pelos escravos ou requisição dos escravos de uma nova tarefa, e transmissão de uma nova tarefa para um escravo. A ocorrência do evento *up* representa o envio da primeira tarefa para cada nó escravo. Esta condição é uma característica particular do ambiente de máquinas agregadas, uma vez que sabe-se que todos os nós escravos estarão prontos para processar tarefas no início da execução de uma aplicação.

Pela ocorrência do evento sincronizante s_i, o nó mestre envia uma nova tarefa para o i-ésimo escravo. De uma maneira similar, a recepção dos resultados avaliada pelo i-ésimo escravo é feita através da ocorrência do evento sincronizante r_i. Quando o nó mestre recebe uma resposta de um nó escravo, ele pode alternativamente enviar uma nova tarefa para este escravo (se ainda existirem tarefas a serem processadas - representado pela probabilidade π_1) ou somente receber e armazenar o resultado (se não existirem mais tarefas a serem processadas - representado pela probabilidade $1 - \pi_1$).

O autômato *Tarefas* é utilizado para contar o número de tarefas restantes a serem processadas pelos escravos. Este autômato possui $N + 1$ estados, aonde N representa o número total de tarefas a serem executadas. Quando o mestre recebe a resposta de um escravo (ocorrência de um dos eventos $r_{0..P}$), ele desconta o número de tarefas restantes, mudando o estado deste autômato.

O autômato $Escravo^i$ representa o i-ésimo escravo com estados: $Init$ (*inicial*), $Comp$

(*processando*) e Tr (*enviando*). O evento sincronizante s_i representa a recepção de uma nova tarefa pela i-ésimo escravo enviada pelo nó mestre. O $Escravo^i$ finaliza o processamento de uma tarefa pela ocorrência do evento local p_i. O evento sincronizante r_i representa o envio dos resultados de uma tarefa processada para o nó mestre.

4.3 Modelo SAN para Fases Paralelas

O modelo SAN para fases paralelas é mostrado na figura 6. Este modelo específico contém dois autômatos representando o processo i e o número de tarefas a serem processadas. Vale lembrar que o número de processos pode variar, acrescentando ou diminuindo o número de autômatos no modelo.

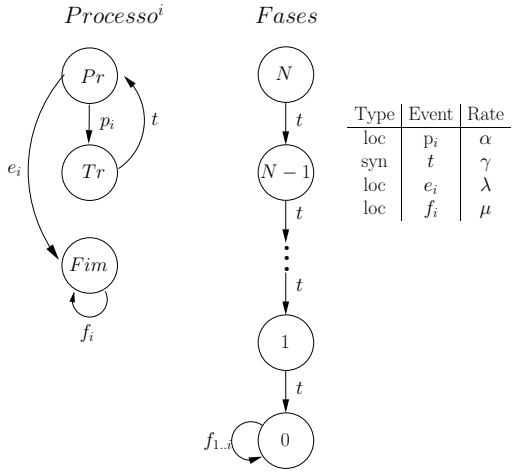


Figura 6: SAN genérico para fases paralelas.

Cada autômato *Processo* possui 3 estados, denominados *Pr*, *Tr* e *Fim* que representam, respectivamente, o processamento, a transmissão e a finalização da execução. Com o evento local p_i, o processo i termina o seu processamento, iniciando assim a fase de sincronização (toca de dados) entre todos os processos. Com o evento sincronizante t todos os processos finalizam a fase de troca de dados e retornam para o processamento da próxima fase. A finalização da execução paralela é indicada através do evento e_i. Os eventos $f_{1..i}$ são necessários para que o modelo SAN se torne cíclico, uma condição que deve ser utilizada sempre quando estes modelos forem resolvidos utilizando a ferramenta PEPS. Neste caso, o estado *Fim* dos autômatos tem característica de estado absorvente, ou seja, conforme o tempo de execução aumenta, a probabilidade deste estado também aumenta. Esta condição é importante, quando utilizada a análise transiente, para que se possa encontrar o tempo de execução aproximado da aplicação modelada.

O autômato *Fases*, diferentemente do modelo SAN mestre escravo, representa o número de fases restantes a serem processadas. Observe que aqui cada vez que a ocorrência do evento t é realizada o número de fases decresce. Ainda, o número de tarefas realizadas

por cada processo é pré-determinada, *i.e.*, cada processo realizará a computação de uma ou mais tarefas em todas as fases paralelas.

4.4 Modelo SAN para *Pipeline*

O modelo SAN para *pipeline* é mostrado na figura 7. Este modelo específico contém três autômatos representando os processos 1, i e o número de tarefas a serem processadas. Vale lembrar que o número de processos pode variar, acrescentando ou diminuindo o número de autômatos no modelo.

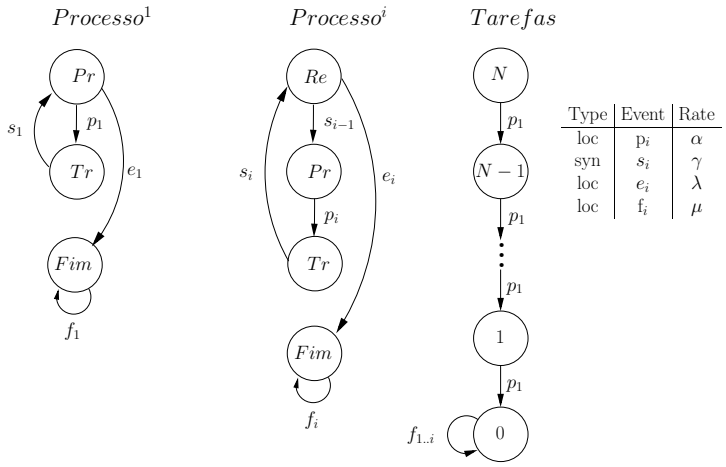


Figura 7: SAN genérico para *pipeline*.

Neste modelo, o autômato $Processo^1$ possui 3 estados, denominados Pr (Processando), Tr (transmitindo), e Fim (Finalizando). Já os outros processos possuem um estado a mais (Re - Recebendo), que serve para receber tarefas de seu processo antecessor. Esta característica se dá pelo fato do modelo de programação *pipeline* determinar que um processo somente inicia execução de uma tarefa quando o processo anterior a tiver terminado. Assim, o processo 1 tem como estado inicial Pr enquanto os outros processos aguardam o recebimento da tarefa em Re. Pela ocorrência do evento p_1, o processo 1 finaliza seu processamento e inicia o envio da tarefa para o processo 2. O evento $s1$ caracteriza o fim do envio do processo 1 para o processo 2 e início do processamento deste. O processo i, por sua vez, finaliza seu processamento através da ocorrência do evento p_i. Da mesma forma o envio de uma tarefa de um processo antecessor para um sucessor é realizado sucessivamente até que a tarefa chegue ao último processo da fila. A ocorrência do evento e_i em cada autômato $Processo$ indica a finalização da execução paralela.

Semelhante ao modelo de fases paralelas, todos os autômatos necessitam de um evento $f_{1..3}$ para que a tomada de tempo de execução da aplicação possa ser realizada. Além disso, um autômato $Tarefas$ é utilizado para contar o número de tarefas já processadas. Este autômato é necessário para definir o critério de parada do modelo SAN.

4.5 Modelo SAN para Divisão e Conquista

O modelo SAN para divisão e conquista é mostrado na figura 8. Este modelo contém 3 autômatos que representam o processo raiz, processo intermediário e processo folha. Por possuir características diferentes dos outros modelos apresentados até então, tais como organização dos processos e divisão das tarefas, este modelo de programação foi o mais complexo de ser modelado utilizando SAN. Assim, para acrescentar processos neste modelo, deve-se alterar os autômatos já existentes, uma vez que a divisão de trabalho deve ser escalonada conforme o número de processos envolvidos.

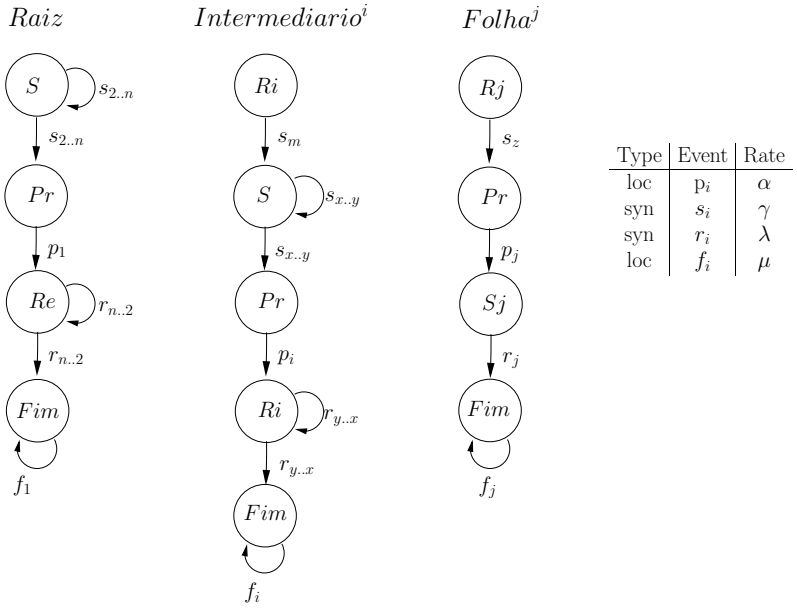


Figura 8: SAN genérico para divisão e conquista.

Este modelo, como dito anteriormente, pode ser visto também como uma árvore de processos, onde a comunicação de cada nível significa uma transmissão de dados entre dois processos. O estado S do autômato $Raiz$ significa a divisão e envio de trabalho para seus processos filhos. Em outras palavras, o *processo* $Raiz$ divide a tarefa inicial em duas sub-tarefas, envia uma sub-tarefa para um processo filho e fica com uma sub-tarefa. Se ainda houverem processos que não possuem uma tarefa, o processo $Raiz$ divide novamente a sua sub-tarefa em duas sub-sub-tarefas, enviando uma delas para outro processo filho. Esta divisão é realizada até que todos os processos obtenham uma tarefa para processar. Observe que processos filhos do $Raiz$ podem ser tanto processos intermediários quanto processos folhas.

Os processos intermediários, representados pelo autômato $Intermediario^i$ recebem uma tarefa de seu processo pai, quebram em sub-tarefas e enviam para seus filhos. Já os processos folhas, representado pelo autômato $Folha^j$ não possuem processos filhos. Estes recebem uma tarefa, processam-a e enviam a resposta para seu processo pai. Note que utilizou-se uma generalização quanto ao número de envios e recebimentos de tarefas nos

autômatos $Raiz$ e $Intermediario^i$. Porém, quando o número de processos for definido em uma execução, estes estados de envio e recebimento serão quebrados em mais estados, cada um representando o envio/recebimento de uma tarefa para/de um processo filho.

Pela ocorrência o evento s_i, o envio de uma tarefa de diferente tamanho é realizado. O evento r_i representa o recebimento de um resultado processado por um processo filho. O evento p_i ocorre quando um processamento de uma tarefa é finalizado. Por fim, a ocorrência do evento f_i é necessária para que o modelo torne-se cíclico, igualmente aos outros modelos SAN apresentados até agora.

Capítulo 5

Como Parametrizar Modelos SAN

O próximo passo para completar a elaboração dos modelos SAN é a determinação de valores numéricos associados às taxas dos eventos e probabilidades. Alguns parâmetros são dados pelo desenvolvedor (valores de entrada do modelo), enquanto outros são avaliados utilizando estes valores de entrada. Algumas variáveis são criadas para auxiliar a definição dos parâmetros:

- CS - tempo de comunicação que um processo leva para enviar dados para outro processo;
- PS - tempo de processamento que um processo consome para concluir uma tarefa;

Nos modelos descritos neste trabalho considera-se que diferentes tarefas (e seus resultados) têm a mesma média de custo de comunicação e processamento. Todavia, sem nenhuma perda de generalidade nos modelos propostos, taxas médias diferentes poderiam ser associadas aos diferentes nós para definir características específicas do ambiente paralelo utilizado na modelagem.

Os tempos CS e PS podem ser calculados de duas formas: utilizando programas simples que coletam estes tempos ou utilizando fórmulas com características específicas das máquinas e da rede alvo.

A primeira forma de calcular o tempo CS é utilizando um programa com o código descrito no algoritmo 1.

Algoritmo 1 Algoritmo para calcular o tempo de envio de uma tarefa.

```
1: se processo = 0 então
2:     Pega tempo inicial
3:     Envia bytes para processo 1
4:     Pega tempo final
5:     Tempo de Envio = Tempo inicial - Tempo final
6: senão
7:     Recebe bytes do processo 0
8: fim se
```

Observe que este programa não possui complexidade e pode ser utilizado para coletar o valor para CS em diferentes máquinas agregadas. A segunda forma de calcular este tempo é utilizando a fórmula 1.

$$CS = NB \times BS \tag{1}$$

Onde NB é o número de *bytes* a ser enviado e BS é o tempo de envio de um byte em uma rede. Este segundo tempo pode ser extraído dos manuais da rede que será utilizada. Independente do método adotado para calcular os tempos de comunicação, ambos podem representa a realidade de forma aproximada.

A primeira forma de calcular PS é utilizando o programa seqüencial a ser paralelizado. Em outras palavras, geralmente um programa que necessite ser paralelizado já foi implementado seqüencialmente. Assim, pode-se identificar dentro deste programa qual a parte a ser paralelizada, calcular o tempo de execução desta parte de código e dividir este tempo pelo número de processos que irão participar do processamento paralelo. Desta forma, tem-se o tempo aproximado de processamento paralelo de cada processo, ou seja, PS.

A segunda forma de calcular PS é utilizando também uma fórmula simples, apresentada em 2.

$$PS = NO \times TO \tag{2}$$

Onde NO é o número de operações aritméticas que uma tarefa terá e TO é o tempo que um processador demora para processar uma operação aritmética. Novamente, este tempo pode ser retirado do manual dos processadores a serem utilizados. Com estes dois tempos, é possível calcular todas as taxas dos quatro modelos SAN apresentados anteriormente. Estas taxas são definidas a seguir.

5.1 Obtenção das taxas

Nesta seção serão apresentadas as formas de obtenção das taxas para cada modelo SAN genérico descrito no capítulo 4.

5.1.1 Modelo Mestre/Escravo

Os eventos s_i, p_i, e r_i do modelos SAN mestre/escravo têm suas taxas (γ, μ, e λ respectivamente) definidas numericamente por (equação 3):

$$\gamma = \frac{1}{CS} \qquad \mu = \frac{1}{PS} \qquad \lambda = \frac{1}{CS} \tag{3}$$

Para o modelo mestre/escravo, a taxa α do evento *up* é numericamente definida considerando o envio das tarefas iniciais para todos os P escravos (equação 4):

$$\alpha = \frac{1}{P \times CS} \tag{4}$$

A probabilidade de ainda haver tarefas a serem distribuídas para os escravos (π_1) é uma probabilidade funcional a qual depende diretamente do estado do autômato *Tarefas*. Esta probabilidade assegura que o nó mestre irá enviar novas tarefas para os nós escravos somente se ainda houver tarefas a serem processadas, *i.e.*(equação 5):

$$\pi_1 = \begin{cases} 1 & \text{if } \mathit{Tarefas} \quad != 0 \\ 0 & \text{if } \mathit{Tarefas} \quad == 0 \end{cases} \tag{5}$$

5.1.2 Modelo de Fases Paralelas

No modelo de fases paralelas, as taxas são utilizada de acordo com os eventos mostrados na tabela da figura 6. A taxa do evento p_i deste modelo é calculada da mesma forma que a taxa do evento p_i no modelo mestre/escravo. Já a taxa γ, relacionada ao evento sincronizante t, é calculada da seguinte forma (equação 6):

$$\gamma = \frac{1}{P \times CS} \tag{6}$$

Sendo P o número total de processos e CS, neste caso, o tempo de transmissão do resultado de um processo para todos os processos restantes. Por fim, tanto a taxa λ como a taxa α são taxas funcionais, descritas da seguinte forma (equação 7):

$$(st\ \mathit{Fases} == 0) \tag{7}$$

Onde mais uma vez P é o número total de processos. Esta taxa funcional indica que os eventos $f_{1..P}$ e $e_{1..P}$ ocorrerão somente quando todos os processos estiverem no estado *Fim*, ou seja, finalizado a execução.

5.1.3 Modelo *Pipeline*

O modelo *pipeline* também possui quatro taxas, mostradas na tabela da figura 7. Os eventos locais $p_{1..P}$ possuem mesma taxa α igual aos outros dois modelos apresentados anteriormente. Já a taxa γ é diferente, e pode ser calculada da seguinte forma (equação 8):

$$\lambda = \frac{1}{CS} \tag{8}$$

Onde CS, neste caso, é o tempo de transmissão de uma tarefa de um processo para o outro. Por outro lado, a taxa λ e a taxa μ são as mesmas do modelo de fases paralelas, taxas funcionais onde todos os processos devem estar no estado *Fim*.

5.1.4 Modelo Divisão e Conquista

O modelo divisão e conquista tem a taxa α igual à descrita nos outros modelos SAN. A taxa γ de envio de tarefas é calculada da mesma forma que no modelo *pipeline*. Porém, as tarefas possuem tamanhos diferentes, modificando a taxa de cada evento. A taxa λ é, para este estudo de caso, a mesma que a taxa γ. Isto ocorre porque o número de dados a serem retornados são os mesmos que enviados em uma tarefa. Por fim, a taxa μ, relacionada aos eventos locais de fim, são taxas funcionais de mesma sintaxe apresentadas em outros modelos SAN.

5.2 Escolha dos Estados Iniciais e Finais

Para apresentar a análise transiente dos modelos, é necessário assumir um conjunto de estados iniciais e finais para cada modelo SAN. Os estados inicias e finais são descritos nesta seção utilizando a sintaxe da ferramenta PEPS. Por exemplo, a operação *st* <autômato> resulta o estado local do autômato mencionado. A operação *nb* [<*conjunto de autômatos*>] <*state*> resulta no número de autômatos no conjunto citado que estão no estado mencionado. Uma descrição detalhada sobre a sintaxe utilizada na ferramenta PEPS pode ser encontrada em [23].

5.2.1 Modelo Mestre/Escravo

O modelo SAN que representa o modelo mestre/escravo assume todos os escravos dedicados à execução de tarefas da aplicação paralela. Então, o estado inicial para este modelo reproduz todos os nós (incluindo o nó mestre) no estado inicial, *i.e.*, ($st\ Mestre == Init$) e para $i = 1..P$ ($st\ Escravo^{(i)} == Init$). O autômato *Tarefas* inicia como todas as N tarefas a serem processadas, *i.e.*, ($st\ Tasks == N$). Assim, existe somente um possível estado inicial para estes modelo. Este estado inicial é definido na equação 9:

$$\begin{array}{ll} (\text{st } \mathit{Mestre} == \text{Init}) & \&\& \\ (\text{nb } [\mathit{Escravo}[1]..\mathit{Escravo}[\text{P}]] \text{ Init} == \text{P}) & \&\& \\ (\text{st } \mathit{Tarefa} == \text{N}) & \end{array} \tag{9}$$

O único estado final possível para o modelo mestre/escravo indica os nós escravos retornando ao seus estados locais originais com todas as tarefas processadas, *i.e.*, ($st\ Tasks == 0$) e cada nó escravo não tendo mais nenhuma tarefas para enviar ($st\ Master == Recv$). Usando a sintaxe da ferramenta PEPS, o estado final corresponde a (equação 10):

$$\begin{array}{ll} (\text{st } \mathit{Master} == \text{Recv}) & \&\& \\ (\text{nb } [\mathit{Slave}[1]..\mathit{Slave}[\text{P}]] \text{ Init} == \text{P}) & \&\& \\ (\text{st } \mathit{Tasks} == 0) & \end{array} \tag{10}$$

5.2.2 Modelo de Fases Paralelas

Estabelecida as taxas do modelo de fases paralelas, faltam definir o estado inicial e final deste modelo. O estado inicial é o primeiro estado de cada autômato, como mostra a equação 11.

$$\begin{array}{l}(\text{nb } [\mathit{Processo}[1]..\mathit{Processo}[\text{P}]] \text{ Pr} == \text{P}) \quad \&\& \\ (\text{st } \mathit{Tasks} == \text{N})\end{array} \tag{11}$$

Por outro lado, o estado final do modelo SAN de fases paralelas é o somatório de todos os processos no estados *Fim*. Assim, o estado final, descrito na sintaxe do PEPS é o apresentado na equação 12.

$$\begin{array}{l}(\text{nb } [\mathit{Processo}[1]..\mathit{Processo}[\text{P}]] \text{ Fim} == \text{P}) \quad \&\& \\ (\text{st } \mathit{Tasks} == 0)\end{array} \tag{12}$$

5.2.3 Modelo *Pipeline*

O modelo *pipeline* tem como estado inicial e final, respectivamente, a soma do primeiro estado de cada autômato e todos os autômatos no estado *Fim*. Assim, o estado inicial ficará (equação 13):

$$\begin{array}{ll}(\text{st } \mathit{Processo\ 1} == \text{Pr}) & \&\& \\ (\text{nb } [\mathit{Processo}[2]..\mathit{Processo}[\text{P}]] \text{ Re} == (\text{P-1})) & \&\& \\ (\text{st } \mathit{Tasks} == \text{N}) & \end{array} \tag{13}$$

Já como estado final para o modelo *Pipeline*, tem-se (equação 14):

$$\begin{array}{l}(\text{nb } [\mathit{Processo}[1]..\mathit{Processo}[\text{P}]] \text{ Fim} == \text{P}) \quad \&\& \\ (\text{st } \mathit{Tasks} == 0)\end{array} \tag{14}$$

5.2.4 Modelo Divisão e Conquista

O estado inicial para o modelo divisão e conquista é a soma de todos os primeiros estados dos autômatos deste modelo. A equação 15 mostra como ficaria a descrição dos estados iniciais deste modelo SAN.

$$\begin{array}{ll}(\text{st } \mathit{Raiz} == \text{S}) & \&\& \\ (\text{nb } [\mathit{Intermediario}[\text{i}]..\mathit{Intermediario}[\text{j}]] \text{ Ri} == \text{N}) & \&\& \\ (\text{nb } [\mathit{Folha}[\text{x}]..\mathit{Folha}[\text{y}]] \text{ Rj} == \text{M}) & \&\& \end{array} \tag{15}$$

Onde N é o número de processos intermediários e M o número de processos folhas. Semelhante aos dois últimos modelos, o modelo de divisão e conquista possui também um estado de finalização em todos os autômatos. Novamente, o estado final deste modelo é a soma de todos os autômatos localizados no estado *Fim*, como mostra a equação 16.

$$
\begin{array}{ll}
(\text{st } \textit{Raiz} == \text{Fim}) & \&\& \\
(\text{nb } [\textit{Intermediario}[\text{i}]..\textit{Intermediario}[\text{j}]] \text{ Fim} == \text{N}) & \&\& \\
(\text{nb } [\textit{Folha}[\text{x}]..\textit{Folha}[\text{y}]] \text{ Fim} == \text{M}) & \&\&
\end{array} \tag{16}
$$

Capítulo 6

Estudos de Caso - Modelagem

Nesta seção são realizadas breves descrições das aplicações escolhidas para validação dos modelos propostos. A primeira aplicação escolhida, considerada clássica em programação paralela, é a multiplicação de matrizes. A segunda aplicação, também bem difundida em computação de alto desempenho, é a ordenação de vetores. Estas duas aplicações foram utilizadas por apresentarem soluções paralelas bem definidas na literatura. As duas últimas aplicações são problemas reais e foram implementadas dentro do Centro de Pesquisa de Aplicações Paralelas (CAP), um centro apoiado pela Pontifícia Universidade Católica do Rio Grande do Sul e pela HP Brasil. São elas: simulação da trajetória de elétrons em um dispositivo *Field Emission Display* (FED), e Renderização de documentos utilizando *Format Object Processing* (FOP). Cabe ressaltar que as aplicações escolhidas não terão desempenho idêntico em todos os modelos utilizados. Com isto, pretende-se verificar se os modelos SAN serão capazes de captar os diferentes comportamentos de uma aplicação segundo o modelo de programação utilizado.

6.1 Estudo de Caso 1 - Multiplicação de Matrizes

A multiplicação de matrizes é uma das aplicações mais clássicas paralelizadas. Diversas maneiras de paralelizar esta aplicação podem ser encontradas na literatura [14, 21]. A implementação seqüencial desta aplicação recebe duas matrizes de mesma ordem, denominadas matriz A e matriz B, e multiplica linhas da matriz A por colunas da matriz B. Matrizes de mesma ordem foram utilizados neste trabalho para facilitar o entendimento do leitor. Porém, a multiplicação de matrizes pode ser realizada com matrizes de ordem diferente. A figura 14 ilustra a idéia do funcionamento da aplicação seqüencial.

Observe que cada multiplicação de uma linha de A por uma coluna de B resulta em um elemento da matriz resposta (matrix C). Assim, percebe-se que todas as linhas de A devem ser multiplicadas por todas as colunas de B para completar a matriz C. Ainda, consegue-se observar que a multiplicação de diferentes linhas de A pelas colunas da matriz de B não possuem interdependência, ou seja, podem ser computadas em paralelo.

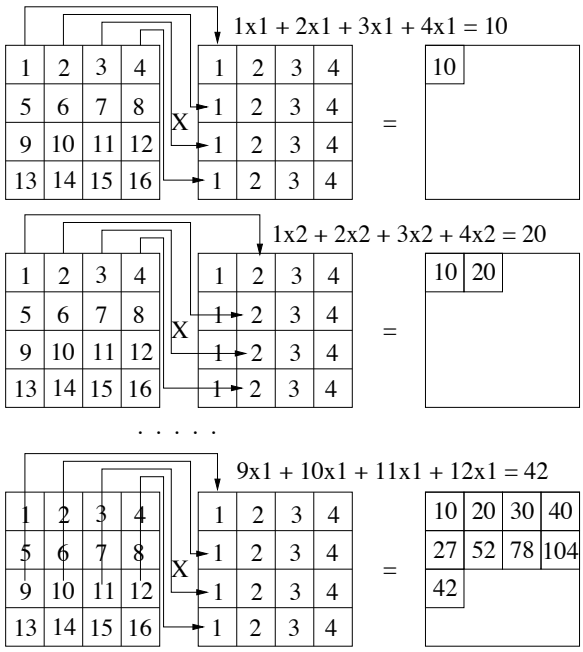


Figura 9: Ilustração do funcionamento da multiplicação de matrizes seqüencial.

Portanto, a implementação paralela deste estudo de caso, utilizando qualquer modelo de programação, baseia-se na divisão das multiplicações das linha de A pela matriz B. A seguir são apresentadas as soluções paralelas para a multiplicação de matrizes em paralelo usando os quatro modelos discutidos na seção 4: mestre/escravo, fases paralelas, *pipeline* e divisão e conquista [1, 30].

6.1.1 Modelo Mestre/Escravo

No modelo de programação Mestre/Escravo uma unidade (mestre) é responsável por distribuir tarefas e receber as respostas das outras unidades (escravos). Além disso, o mestre é capaz realizar uma fase de pré-processamento, ou inicialização, antes da distribuição de tarefas e outra fase de pós-processamento para unir as respostas ou finalizar. Adotaremos aqui a versão paralela mestre/escravo onde escravos recebem linhas da primeira matriz e a segunda matriz inteira, como mostra a figura 10.

Nesta figura pode-se observar que cada escravo possui processamentos diferentes. Além disso, parte-se do princípio que cada escravo conhece a matriz B antes do início da execução da aplicação. O mestre envia tarefas (linhas da matriz A) para cada escravo à medida que ele fica ocioso esperando por processamento. Os escravos retornam resultados de uma tarefa para o mestre, que representam linhas da matriz resultante da multiplicação (matriz C).

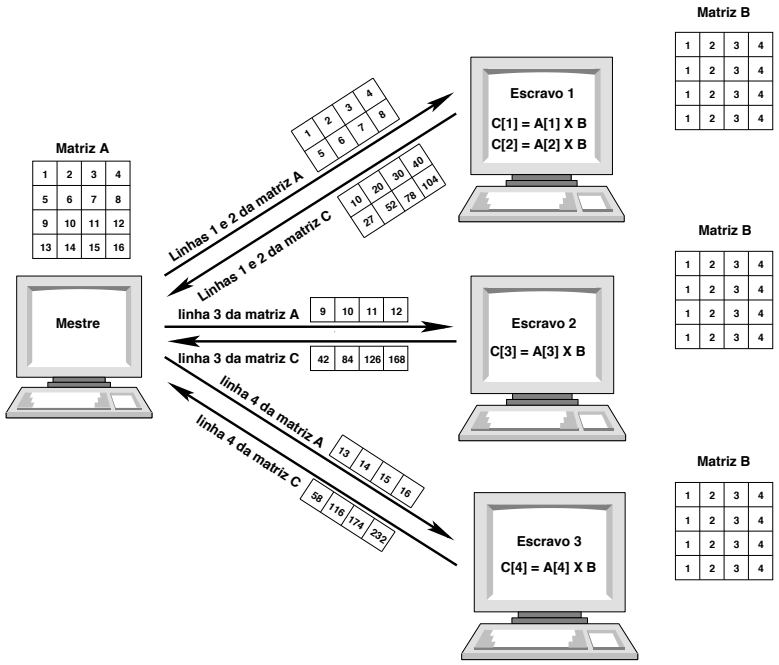


Figura 10: Multiplicação de matrizes com o modelo mestre/escravo.

Parametrização do Modelo

Os eventos do modelo mestre/escravo foram parametrizados da seguinte forma: a taxa μ do evento p_i foi coletada através da utilização da fórmula mostrada na equação 17.

$$\mu = \frac{1}{TP \times NE} \tag{17}$$

Onde TP corresponde ao tempo de execução de uma operação matemática em uma máquina do *cluster* e NE o número de elementos a serem multiplicados da matriz A. Este tempo aproximado invertido representa a taxa de processamento de uma tarefa por um escravo. Para as taxas γ e λ dos respectivos eventos s_i e r_i, utilizou-se programas semelhantes ao descrito no algoritmo 1. Por fim, a taxa α do evento *up* é uma taxa funcional com a seguinte descrição (equação 18).

$$\begin{aligned} &(\text{st } \mathit{Mestre} == \text{Init}) && \&\& \\ &(\text{nb } [\mathit{Escravo}[1]..\mathit{Escravo}[\text{P}]] \text{ Init} == \text{P}) && \&\& \\ &(\text{st } \mathit{Tarefa} == \text{N}) \end{aligned} \tag{18}$$

6.1.2 Modelo de Fases Paralelas

O modelo de programação Fases Paralelas consiste de fases de processamento seguidas de sincronização por todas as unidades. A sincronização termina após todos os processos terem finalizado sua fase de processamento. Na versão de fases paralelas, semelhante à versão Mestre/Escravo, todos os processos conhecem a matriz B. Ainda, todos os processos realizam processamento de tarefas, não existindo um processo centralizador. A figura 11

apresenta este modelo de programação utilizando a aplicação alvo.

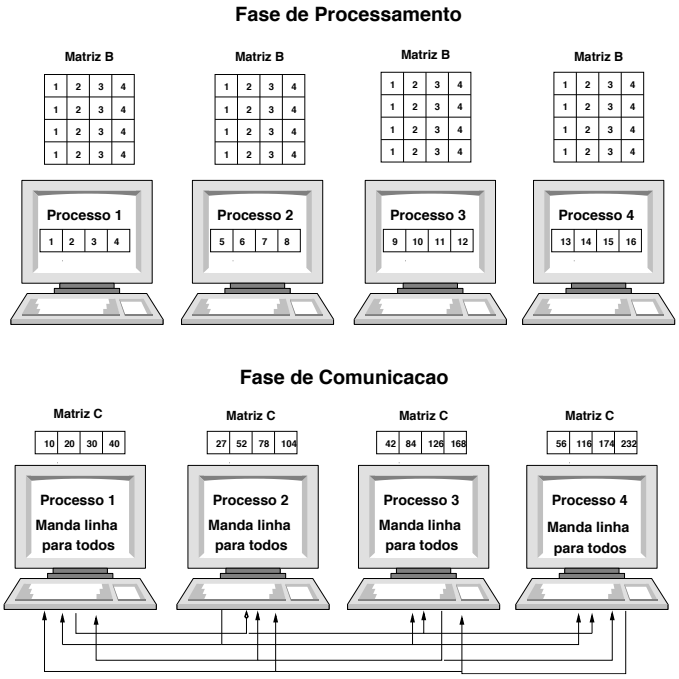


Figura 11: Multiplicação de matrizes com o modelo de fases paralelas.

Os processos determinam, através de um algoritmo de escalonamento, quais são as linhas que estes devem processar. Após o processamento, os processos trocam os elementos resultantes de suas multiplicações, para que todos os processos tenham a matriz resultante completa.

Parametrização do Modelo

Semelhante ao modelo mestre/escravo, a taxa correspondente ao evento p_i do modelo de fases paralelas foi obtida utilizando a mesma fórmula que calcula o tempo de processamento de uma tarefa utilizando os fatores de número de elementos e tempo de uma operação de multiplicação em um computador. Além disso, a taxa do evento t também foi resolvida com a utilização de um programa simples de cálculo de envio de *bytes*. Tanto a taxa λ quanto a taxa μ são taxas funcionais, porém com fórmulas diferentes. A taxa λ do evento e_i é apresentada na equação 19, enquanto a taxa μ do evento f_i é mostrada na equação 20

$$\begin{aligned}&(\text{nb } [Processo[1]..Processo[\text{P}]] \text{ Pr} == \text{P}) \quad \&\& \\ &(\text{st } Fases == 0)\end{aligned} \tag{19}$$

$$\begin{aligned}&(\text{nb } [Processo[1]..Processo[\text{P}]] \text{ Fim} == \text{P}) \quad \&\& \\ &(\text{st } Fases == 0)\end{aligned} \tag{20}$$

Neste estudo de caso utilizou-se o modelo SAN de fases paralelas com somente uma

fase de processamento e uma fase de comunicação, pois a aplicação de multiplicação de matrizes em paralelo não exige o uso de mais fases. Lembrando o funcionamento da multiplicação de matrizes em paralelo cada processo multiplica linhas da matriz A por colunas da matriz B (processamento) e troca os resultados entre si (comunicação), não havendo necessidade de um novo processamento.

6.1.3 Modelo *Pipeline*

No modelo *pipeline*, as unidades cooperam entre si formando uma estrutura linear e seqüencial, na qual cada uma realiza uma parte do trabalho. Após o processamento, a tarefa é enviada para a próxima unidade. Utilizando este modelo com o estudo de caso em questão, sua implementação paralela teria o seguinte esquema (figura 12).

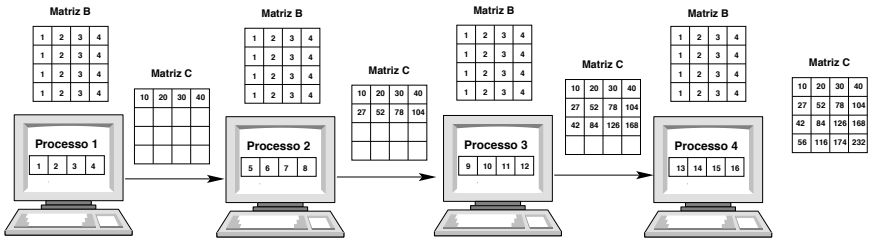


Figura 12: Multiplicação de matrizes com o modelo *pipeline*.

Na figura 12, podemos observar que os processos estão organizados em forma de uma fila. O i-ésimo processo recebe do seu antecessor a matriz resultante (matriz C) incompleta, realiza seu processamento incluindo seu resultado na matriz C e envia essa matriz para seu sucessor. A execução termina quando o último processo da fila termina a inclusão de seu resultado na matriz C. Com esta ilustração, fica claro **que este modelo não é o mais apropriado** para este tipo de aplicação, uma vez que os processamentos estão sendo realizados seqüencialmente.

Parametrização do Modelo

Neste modelo, apresar de possuir um comportamento diferente dos outros até então apresentados não possui diferenciação nas taxas dos eventos p_i e s_i, pois a idéia de processamento e de envio de dados é a mesma. Por outro lado, a taxa funcional do evento e_i varia do processo 1 para os outros processos da seguinte forma (equação 21):

$$\begin{aligned} &\text{Processo} == 1 \\ &(\textit{st Processo 1} == \text{Pr}) \;\&\&\; (\text{st } \textit{Tasks} == 0) \\ &\text{Processo} \mathrel{!}= 1 \\ &(\textit{st Processo 1} == \text{Re}) \;\&\&\; (\text{st } \textit{Tasks} == 0) \end{aligned} \tag{21}$$

Para este estudo de caso utilizou-se somente a multiplicação de uma matriz A por uma matriz B. Logo, a execução deste modelo é seqüencial. O modelo SAN *pipeline* genérico

prevê a multiplicação de várias matrizes A pela matriz B. Assim, o *pipeline* ficaria cheio e apresentaria algum desempenho. Da forma como foi utilizado este modelo (uma matriz A por uma matriz B) o desempenho da aplicação deverá ser pior cada vez que o número de processos aumenta. Esta situação foi realizada propositalmente, para verificar se o modelo SAN consegue acompanhar os resultados da implementação paralela.

6.1.4 Modelo Divisão e Conquista

No modelo divisão e conquista unidades são agrupadas em uma hierarquia de árvore. Os pais dividem suas tarefas e repassam uma parte para seus filhos. Os resultados são integrados recursivamente. A figura 13 apresenta um exemplo dessa hierarquia com 4 processos, indicando como é realizada a divisão de trabalho e comunicação neste modelo.

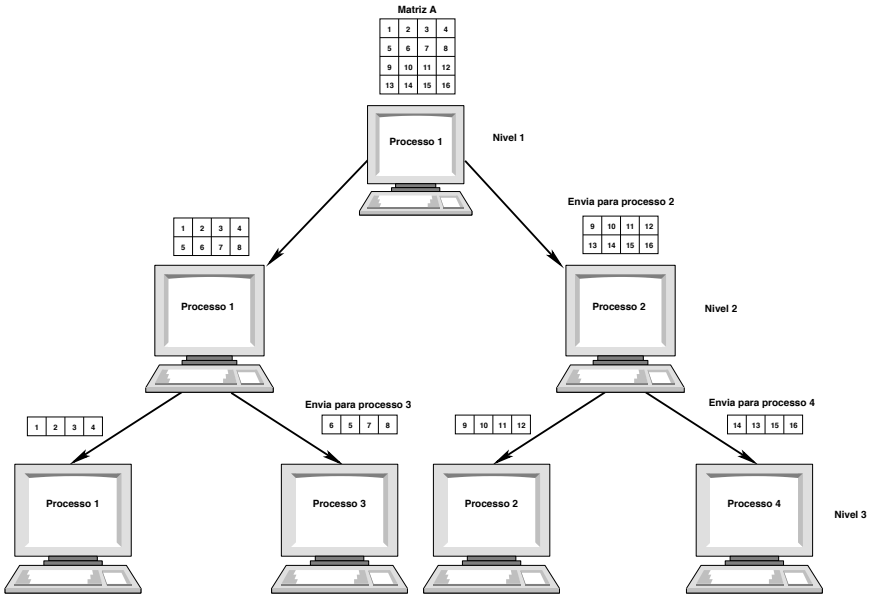


Figura 13: Multiplicação de matrizes com o modelo divisão e conquista.

Nesta figura, percebe-se que todos os processos chegam ao último nível, o qual é realizado o processamento. Após o processamento, todos os processos que receberam tarefa devem retornar o resultado para o processo enviador (denominado processo pai). Assim, uma série de transferência de dados é realizada até que o processo que está no primeiro nível da árvore contenha todos os resultados. Desta forma, todos os processos realizam trabalhos paralelamente e somente na fase de unificação dos resultados que os processos enviam seus resultados para níveis superiores da árvore. Observe que somente o processo raiz conterá a matriz C completa.

Parametrização do Modelo

Pelo fato de tarefas serem quebradas em sub-tarefas, estas terão as vezes um custo computacional diferente influenciando diretamente no cálculo das taxas α, γ e λ que

correspondem aos eventos p_i, s_i e r_i. Todavia, a forma de calcular essas taxas foi a mesma utilizada nos outro modelos SAN supracitados. Por fim, o evento f_i possui uma taxa funcional descrita na equação 22.

$$\begin{array}{ll} (\text{st } \mathit{Raiz} == \text{Fim}) & \&\& \\ (\text{nb } [\mathit{Intermediario}[\text{i}]..\mathit{Intermediario}[\text{j}]] \text{ Fim} == \text{N}) & \&\& \\ (\text{nb } [\mathit{Folha}[\text{x}]..\mathit{Folha}[\text{y}]] \text{ Fim} == \text{M}) & \&\& \end{array} \quad (22)$$

6.2 Estudo de Caso 2 - Ordenação de Vetores

Semelhante à multiplicação de matrizes, a ordenação de vetores também é uma das aplicações mais clássicas paralelizadas, apresentando diversas soluções possíveis. Na implementação seqüencial desta aplicação, o vetor pode ser ordenado utilizando diferentes algoritmos, tais como *bubble sort*, *merge sort* e *quick sort*. Neste estudo de caso, utilizou-se um algoritmo diferente que facilitou a paralelização nos diferentes modelos de programação. O algoritmo funciona da seguinte forma: para cada elemento do vetor, o algoritmo percorre o todo vetor comparando este elemento com todos os outros. No final, cada elemento x terá associado um número identificando quantos elementos são menores que ele. Para um melhor entendimento, este número será chamado de **número associado**. A figura 14 apresenta o funcionamento do algoritmo de ordenação de vetores utilizado neste trabalho.

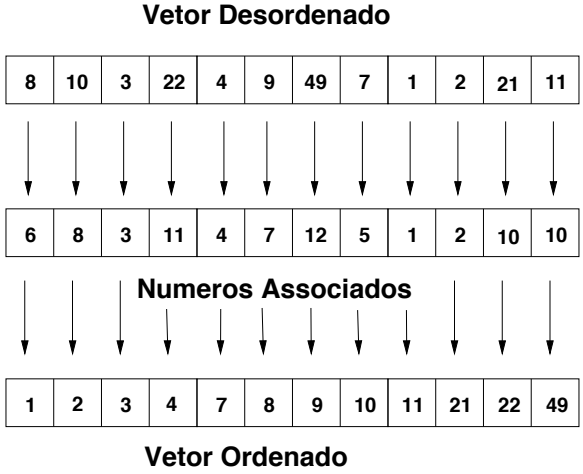


Figura 14: Ilustração do funcionamento da ordenação de vetores seqüencial.

O número associado de x é a posição final que x deverá estar no vetor. Por exemplo, o elemento 10 tem como número associado 8. Logo, o número 10 deverá estar na posição 8 quando o vetor estiver ordenado. Assim, basta percorrer mais uma vez o vetor, colocando todos os elementos nas posições finais. A seguir, serão apresentadas as formas que este algoritmo foi quebrado para adaptar-se nos modelos de programação escolhidos.

6.2.1 Modelo Mestre/Escravo

Como dito anteriormente, o modelo mestre/escravo possui um processo centralizador, que não realizada cálculo (processamento) de tarefas. Neste caso, o processo mestre realiza a distribuição de carga de cada processo escravo e envia esta distribuição para os escravos, como mostra a figura 15.

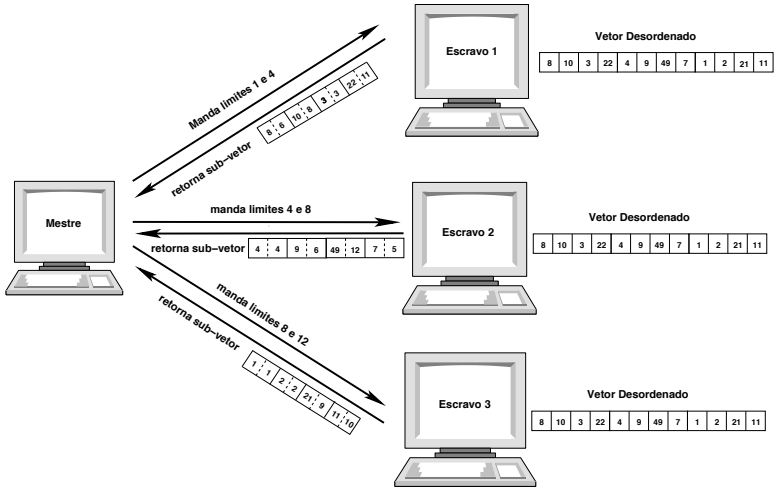


Figura 15: Multiplicação de matrizes com o modelo mestre/escravo.

Os escravos, por sua vez, realizam o cálculo dos números associados de um sub-vetor. Ao final do processamento, cada escravo envia um conjunto de elementos do vetor juntamente com seus números associados. Note que todos os processos possuem o vetor desordenado completo. Isto deve acontecer para que o algoritmo funcione corretamente. Por fim, o mestre ordena o vetor inteiro utilizando os números associados calculados pelos escravos.

Parametrização do Modelo

O modelo SAN mestre/escravo para este estudo de caso foi parametrizado da seguinte forma. O evento p_i possui uma taxa obtida através da fórmula 23.

$$\mu = \frac{1}{(TV + TE) \times NE} \tag{23}$$

Onde TV significa o tempo de percorrer um vetor e TE o tempo de escrever um elemento em um vetor. Novamente, NE indica o número de elementos a serem ordenados. Para os eventos s_i e r_i, utilizou-se o mesmo programa de cálculo de envio de *bytes*. Por fim, a taxa do evento up é funcional, extraída através da mesma equação 18.

6.2.2 Modelo de Fases Paralelas

figura 16 apresenta o funcionamento do modelo de fases paralelas para a aplicação de ordenação de um vetor em paralelo.

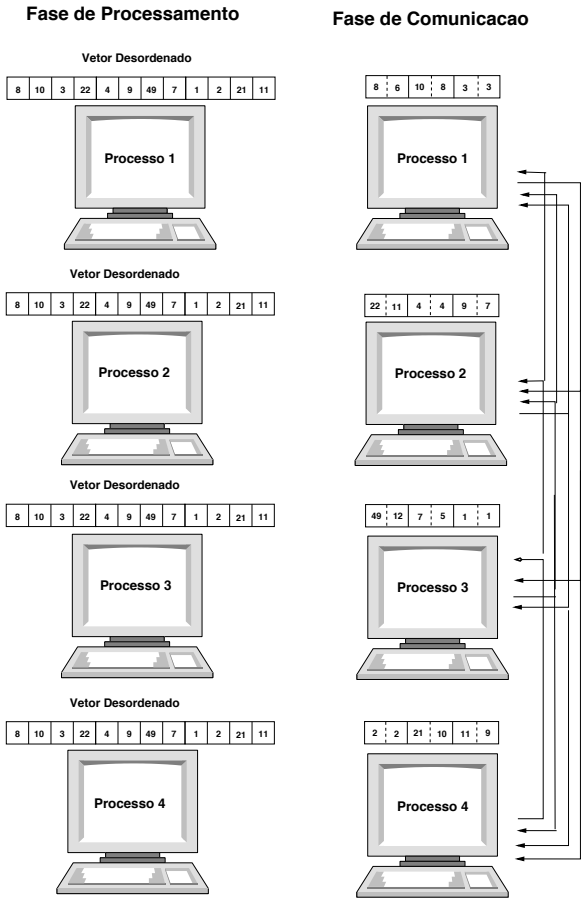


Figura 16: Multiplicação de matrizes com o modelo de fases paralelas.

No modelo de fases paralelas para este estudo de caso os processos também conhecem todo o vetor desordenado. Diferentemente do modelo mestre/escravo, neste modelo os processos identificam localmente suas tarefas.

Como característico no modelo de fases paralelas, todos os processos calculam tarefas. A final do calculo de todas as tarefas, todos os processo trocam informações enviando elementos e números associados calculados. Observe que depois da fase de comunicação todos os processos terão todos os números associados. Assim, o cálculo final de ordenação do vetor é realizado paralelamente por todos os processos.

Semelhante ao estudo de caso anterior, este estudo de caso utiliza somente uma fase de processamento e uma fase de comunicação, pois O algoritmo utilizado para ordenar um vetor em paralelo utilizado na implementação não necessita de mais de uma fase de processamento.

Parametrização do Modelo

A taxa α relacionada ao evento p_i foi também calculada utilizando a equação 23. Como neste modelo em particular os processos trocam informações simultaneamente, é necessário calcular o tempo de um envio e multiplicá-lo pelo número de processos envolvidos para que se obtenha a taxa deste evento t. Portanto, o evento sincronizante teve sua taxa γ obtida utilizando o algoritmo 1, que calcula o tempo do envio de uma resposta por um processo, multiplicado pelo número de processos envolvidos. Por fim, as taxas λ e μ dos eventos e_i e f_i, respectivamente, são funcionais obtidas através das mesmas equações do modelo SAN de fases paralelas utilizando a aplicação de estudo de caso 1 (multiplicação de matrizes).

6.2.3 Modelo *Pipeline*

Semelhante ao funcionamento do modelo *pipeline* no estudo de caso de multiplicação de matrizes, as unidades cooperam entre si formando uma estrutura linear e seqüencial, na qual cada uma realiza uma parte do trabalho, como mostra a figura 17.

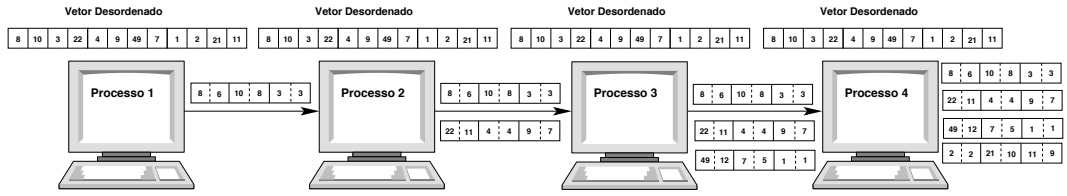


Figura 17: Multiplicação de matrizes com o modelo *pipeline*.

Novamente, este modelo parece não ser adequado para este tipo de aplicação, uma vez que o cálculo dos números associados é realizado seqüencialmente. Como dito anteriormente, este modelo servirá para mostrar a exatidão que os modelos SAN propostos podem alcançar.

Parametrização do Modelo

Mais uma vez, a taxa α do evento p_i foi obtida com a fórmula 23. As taxas γ, λ e μ foram calculadas da mesma forma que no estudo de caso 1 (multiplicação de matrizes).

Como dito no estudo de caso anterior, este modelo de programação utilizará somente um vetor para ordenar. Assim, esta implementação obterá desempenho piores com o aumento de processos. Novamente, este fato foi realizado propositalmente para verificar o comportamento do modelo SAN *pipeline*.

6.2.4 Modelo Divisão e Conquista

Por fim, o modelo divisão e conquista também é utilizado para modelar a aplicação de ordenação de um vetor, com sua estrutura apresentada na figura 18.

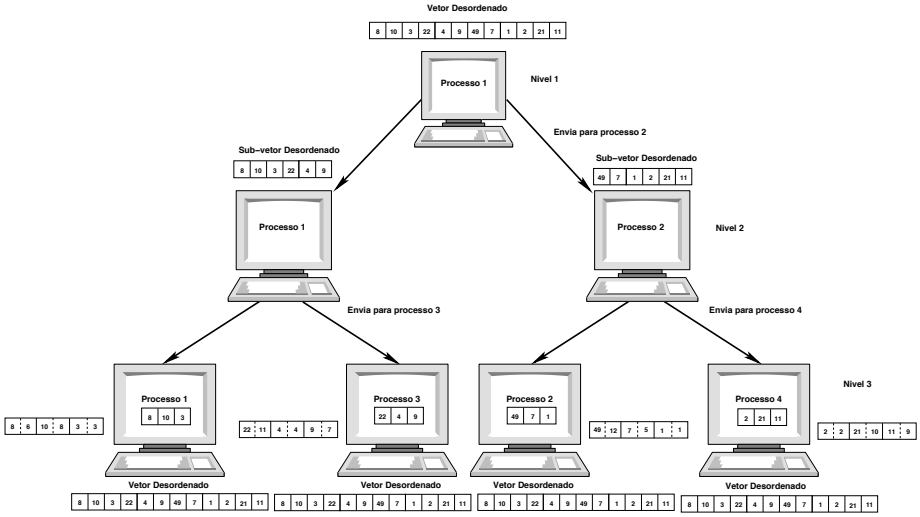


Figura 18: Multiplicação de matrizes com o modelo divisão e conquista.

Neste modelo, o processo no primeiro nível divide o vetor em dois sub-vetores, enviando um sub-vetor para seu processo filho. Este esquema é realizado até que todos os processos contenham uma parte do vetor desordenado. Igualmente a todos os outros modelos e para que o algoritmo de ordenação funcione corretamente, todos os processos conhecem o vetor inteiro. Cada processo calcula os números associados de um conjunto de elementos e retorna estes resultados para seu processo pai. Ao final, o processo no primeiro nível realiza a ordenação propriamente dita.

Parametrização do Modelo

Como dito anteriormente, este modelo de programação pode ter tarefas de tamanho diferente. Contudo, o cálculo da taxa α do evento p_i é realizado da mesma forma que nos outros modelos, ou seja, através equação 23. Nesta fórmula, a variável NE que corresponde ao número de elementos a serem ordenados pode variar em uma mesma execução, neste caso especial. Os eventos s_i e r_i podem também sofrer variações em suas taxas pelo mesmo motivo que o evento p_i. O cálculo dessas taxas mais a taxas funcional μ do evento f_i são calculados igualmente ao estudo de caso 1 (multiplicação de matrizes).

6.3 Estudo de Caso 3 - Simulação de Elétrons em um Dispositivo FED

A primeira aplicação é um estudo da trajetória de eltétrons em um dispositivo do tipo FED [12]. Resumidamente, os chamados *displays* ou mostradores *Cathode Ray Tube* (CRT) e FED possuem, em seu mecanismo de funcionamento, canhões de elétrons e uma superfície reagente a eles. A Figura 19 exibe o funcionamento de um dispositivo CRT.

Em um mostrador CRT os elétrons são disparados por três diferentes canhões, um para

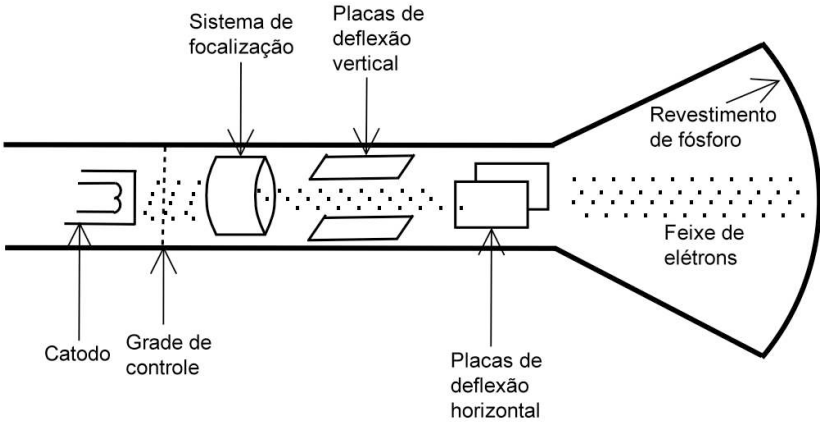


Figura 19: Cathode Ray Tube (CRT).

cada componente de cor RGB (*Red*, *Green* e *Blue*). Basicamente os elétrons disparados são guiados através de um tubo condutor até colidir contra uma superfície de fósforo, fazendo com que esta se ilumine gerando a cor que será visualizada na tela. Por sua vez, a geração de imagem é dada pela varredura desses feixes de elétrons em toda a superfície da tela controlados por um campo magnético aplicado logo após a emissão dos elétrons pelo catodo.

Dispositivos FED, similarmente ao CRT, também possuem canhões de elétrons. No entanto, cada ponto da tela possui um conjunto dedicado de canhões, em escala micrométrica, onde a distância entre estes canhões e a superfície é significativamente reduzida, fazendo com que o monitor seja mais fino, não sendo necessário um componente que faça a varredura da tela. A idéia básica é mostrada na Figura 20.

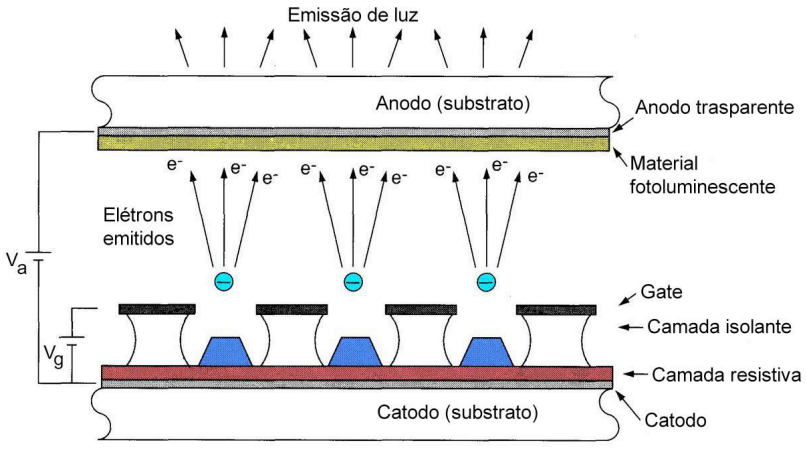


Figura 20: Field Emission Display (FED).

A aplicação simula a trajetória dos elétrons em um FED. O objetivo de sua criação é reduzir custos e ser capaz de avaliar a eficiência em uma dada configuração de dispositivo.

A eficiência é traduzida pela porcentagem de elétrons que foram disparados e atingiram a superfície de fósforo. Por ser uma tecnologia recente, prototipar experimentalmente diferentes configurações do dispositivo é relativamente custoso, o que justifica o uso desta aplicação afim de auxiliar na obtenção da geometria final do dispositivo.

Para executar uma simulação deve ser determinada uma série de parâmetros que especificam a configuração do dispositivo. Com estes parâmetros o simulador é capaz de calcular as trajetórias para um dado número de elétrons. É importante notar que, para se obter resultados mais significativos, os parâmetros devem ser ajustados corretamente. No entanto, os parâmetros, incluindo o número de elétrons, influenciam diretamente no tempo de execução da aplicação.

A aplicação gera um conjunto de dados que permitem visualizar graficamente o comportamento dos elétrons dentro do dispositivo e analisar estatisticamente os resultados. A Figura 21 mostra a visão geral do sistema de simulação.

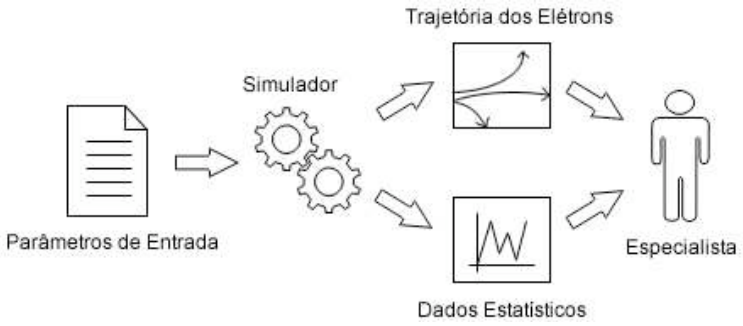


Figura 21: Visão geral do simulador de trajetória de elétrons.

No módulo de simulação (Algoritmo 2), a linha dois do algoritmo representa o sorteio de valores que dirão como os elétrons serão lançados dentro do dispositivo. O laço na linha seguinte realiza o cálculo da trajetória, que tem seu término quando o elétron se choca contra alguma parede do dispositivo.

Algoritmo 2 Algoritmo de simulação de elétrons em um dispositivo FED.

```
1: para e = 0 até número de elétrons faça
2:     Cria condições iniciais de velocidade e posição do elétrons
3:     enquanto Elétron dentro do dispositivo faça
4:         Escreve posição atual em arquivo
5:         Calcula nova posição
6:     fim enquanto
7: fim para
8: Escreve estatísticas
```

6.3.1 Modelo Mestre/Escravo

A aplicação paralela baseado no modelo mestre/escravo considera que todos os processos conhecem o vetor que contém as condições de cada elétron, diminuindo assim a quantidade de dados a ser trocado entre o mestre e um escravo. Os escravos pedem tarefas para o mestre e este retorna a posição do vetor ele deve usar, ou seja, qual elétron deve ser simulado. Os escravos retornam a posição atual de um elétron simulado para o mestre que escreve em um arquivo. A figura 22 ilustra o fluxo de execução dessa aplicação paralela.

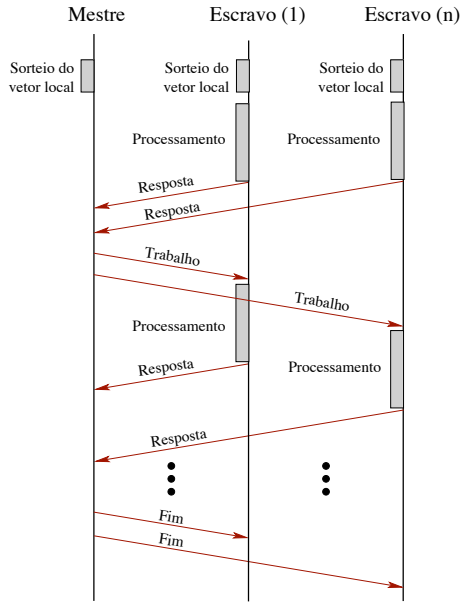


Figura 22: Funcionamento da ferramenta FED em paralelo

Parametrização do Modelo

Diferentemente dos outros estudos de caso apresentados até agora, a parametrização do modelo SAN para simulação de elétrons em paralelo utilizou o código seqüencial desta aplicação para calcular a taxa μ do evento p_i. Ainda, os eventos s_i e r_i também obtiveram suas taxas utilizando o algoritmo 1. Neste caso, foi necessário identificar na versão seqüencial a parte do código a ser paralelizada e o número de informações que os processos trocariam. A taxa do evento *up* continua sendo funcional calculada pela fórmula 18.

6.4 Estudo de Caso 4 - Renderização de Documentos utilizando FOP

A segunda aplicação real que será utilizada como como estudo de caso neste trabalho renderiza documentos com dados variáveis através da ferramenta FOP. Esta ferramenta é uma aplicação com código aberto que trabalha com uma variedade de formatos de saída,

é flexível e facilmente extensível. É uma aplicação Java que lê um objeto de entrada e renderiza para diferentes formatos de saída específicos, como PDF (*Portable Document Format*) , PS (*Post Script*) , e SVG (*Scalable Vector Graphics*) . O arquivo de entrada deve ser formatado pelo padrão XSL-FO (*Extensible Style Sheet Language - Formatting Objects*) e, como resposta, gera um arquivo no formato de saída requisitado, com o conteúdo descrito pelos objetos de formatação (figura 23).

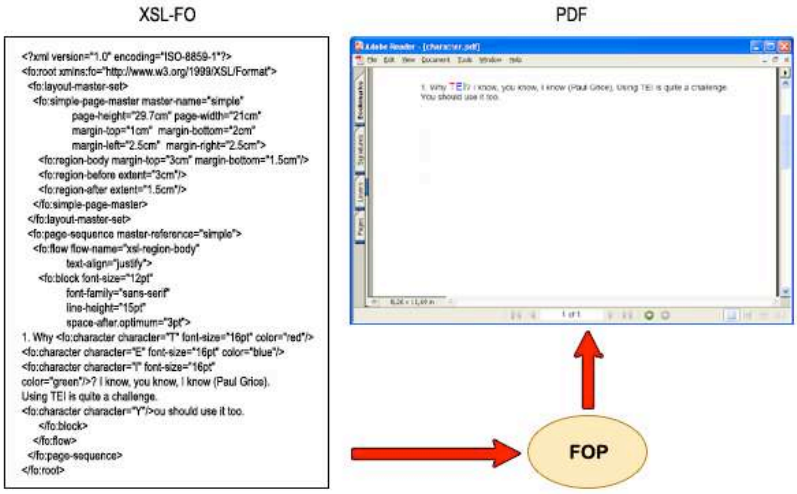


Figura 23: Exemplo de Renderização de documentos com FOP.

A ferramenta FO+Processor (*Formatting Objects Plus Processor*) é responsável por extrair as porções renderizáveis do documento de entrada e entregá-las ao FOP, como mostra a figura 24. A ferramenta FO+Processor pode ser subdividida em duas partes principais: FOExtractor e FO+Iterator. Num primeiro momento o FOExtractor carrega o documento de entrada e armazena-o em uma estrutura interna. Quando o documento termina de ser carregado, começa um processo de busca no conteúdo da estrutura armazenada. Durante este processo, o FOExtractor identifica porções não-renderizáveis e renderizáveis do documento. Ao ser constatada uma porção não-renderizável, é iniciado o armazenamento do conteúdo. Este procedimento continua até que se identifique uma porção variável. Neste momento, todo o conteúdo armazenado (porções não-variáveis) é escrito no arquivo de saída. A seguir, a porção variável é armazenada e enviada para o módulo FO+Iterator. O FOExtractor é bloqueado até que receba a parte renderizada do FO+Iterator.

Nesta segunda parte do processo, o módulo FO+Iterator recebe apenas o XSL-FO pronto para ser renderizado. Este módulo pré-formata o objeto, preparando-o para ser enviado ao FOP. O FOP, então, recebe o objeto de formatação (XSL-FO) e tenta renderizá-lo. No momento em que se consegue renderizar com sucesso o XSL-FO, a porção renderizada é enviada do FOP para o FO+Iterator, que a repassa ao FOExtractor.

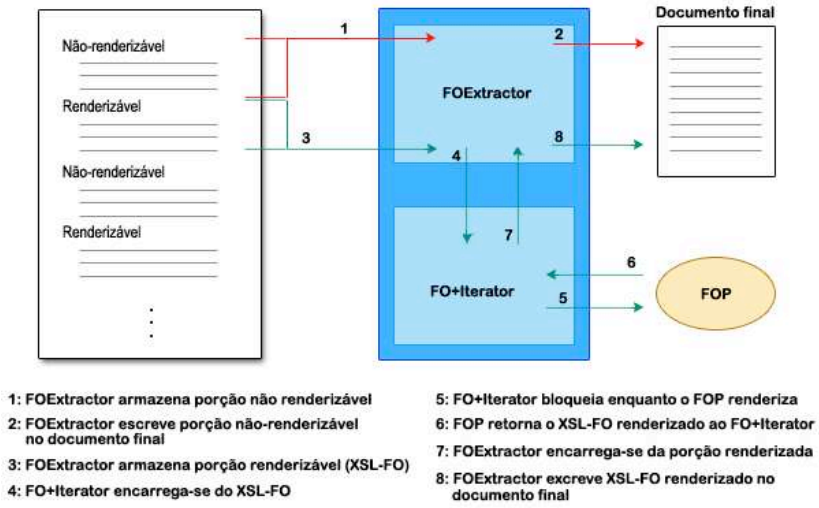


Figura 24: Funcionamento da ferramenta FOP.

6.4.1 Modelo Mestre/Escravo

O fato do FO+Processor ter sua execução bloqueada enquanto a ferramenta FOP renderiza um XSL-FO, aparece como o maior problema no resultado final do seu tempo de execução. Podemos perceber que a ferramenta provavelmente apresentaria uma melhora significativa em sua velocidade e em seu desempenho se, ao invés de ter sua execução bloqueada enquanto aguarda a resposta da outra ferramenta, o FOExtractor pudesse continuar seu fluxo de execução normal paralelamente à renderização dos XSL-FOs já encontrados. Assim, uma nova abordagem da ferramenta foi utilizada para a resolução do problema apresentado, onde a figura 6.4.1 ilustra a idéia dessa paralelização.

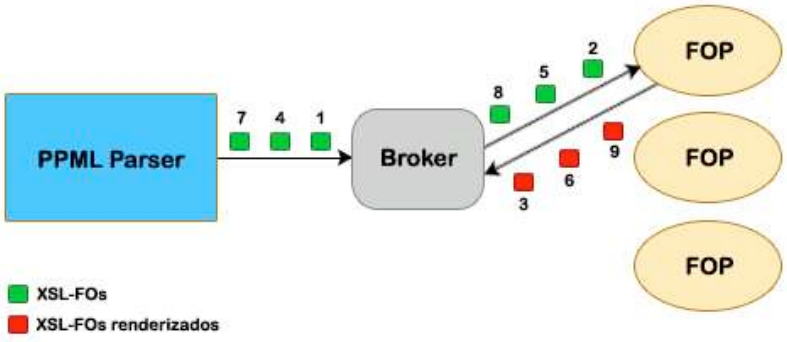


Figura 25: Funcionamento da ferramenta FOP em Paralelo.

A ferramenta divide-se basicamente em três componentes principais: PPML Consumer, Broker e FOP. O componente PPML Consumer é responsável pela leitura e identificação de FOs no documento PPML de entrada e também pela escrita do documento com estas porções variáveis já renderizadas no arquivo de saída. O próximo componente,

o Broker, é responsável por receber os FOs do PPML Consumer e guardá-los em uma fila. Estes objetos devem ser repassados para um componente FOP que esteja livre para renderizá-los. O último componente é o responsável pela de renderização, o FOP. Este processo renderiza os FOs recebidos e, ao término deste processo, envia o resultado para o PPML Consumer e avisa o componente Broker que está livre para receber (se for o caso) outro FO.

Parametrização do Modelo

Semelhante ao estudo de caso de simulação de elétrons em paralelo, este estudo de caso foi parametrizado utilizando o código seqüencial da aplicação. Assim, houve a necessidade de identificar o trecho de código a ser paralelizado e a quantidade de informações a ser trocada entre os processos. O evento p_i teve sua taxa obtida pela tomada de tempo de processamento da aplicação seqüencial. Já os eventos s_i e r_i tiveram suas taxas obtidas utilizando o algoritmo 1. Novamente, o evento up tem sua taxa calculada através da equação 18.

Capítulo 7

Análise Quantitativa

Neste capítulo, os resultados de tempo de execução estimados através dos modelos SAN são comparados com tempos obtidos através de implementações reais dos estudos de caso utilizando os quatro modelos de programação mostrados anteriormente. O experimentos paralelos atuais foram executados em um *cluster* com nós Itanium-2 de 64 bits e 900 MHz biprocessados, com 3 GB RAM e inerconectados por uma rede Myrinet. Cabe ressaltar que para cada tempo de execução das implementações paralelas apresentado neste capítulo foram obtidos através de uma **média de dez execuções**, onde **retira-se** os valores extremos. Foram realizadas somente dez execuções por achar-se suficiente, *i.e.*, considerou-se boa a média com este número de execuções. Todas as implementações das aplicações utilizadas como casos de estudo neste trabalho foram realizadas utilizando a linguagem de programação C++ e as primitivas MPI da biblioteca MPICH. Vale lembrar que utilizou-se troca de mensagens do tipo síncrona através da função *MPI_Send*, uma vez que o envio assíncrono é difícil de ser modelado utilizando o formalismo SAN.

Com o objetivo de validar as estratégias dos estudos de caso em paralelo escolhidos e modelados usando SAN, alguns modelos SAN foram resolvidos com diferentes conjuntos de parâmetros de entrada (*e.g.*, número de escravos). Além disso, três diferentes grãos (números de tarefas) foram considerados. Para o primeiro estudo de caso (multiplicação de matrizes) foram utilizados os seguintes grãos (número de tarefas): 10.000 (pequeno), 15.000 (médio) e 20.000 (grande) tarefas. Vale lembrar que a ordem das matrizes para cada um dos três casos é igual ao número de tarefas, pois as matrizes são quadradas e cada tarefa representa uma linha da matriz A. Para o segundo estudo de caso (ordenação de vetores) os grãos variaram entre 50.000 (pequeno), 100.000 (médio) e 150.000 (grande). Neste segundo estudo de caso também a ordem dos vetores é igual ao número de tarefas. Os valores para os vetores foram gerados de forma aleatória através do uso da função *random* da linguagem C/C++. Já para o terceiro estudo de caso (FED) variou-se o número de elétrons por tarefa entre 10 elétrons (grão grosso), 5 elétrons (grão médio) e 2 elétrons (grão fino). Por fim, para o último estudo de caso (FOP) utilizaram-se dados de entrada prontos que resultam em 165 (pequeno) e 337 (grande) tarefas. Variou-se o

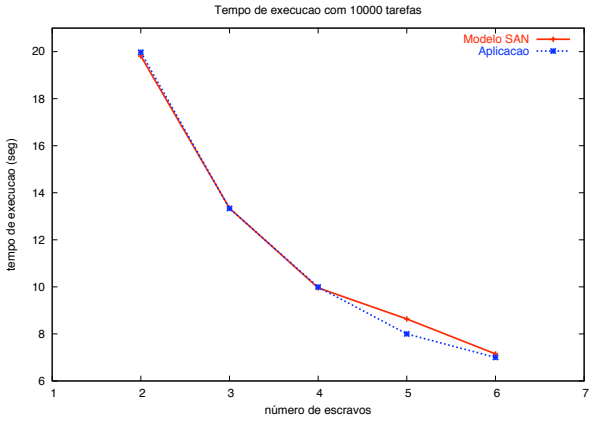
número de processos entre 2 e 7. Esta variação teve o limite de 7 processos devido ao tamanho do modelo SAN. Em outras palavras, a explosão do espaço de estado de um modelo SAN com mais do que 7 processos torna inviável sua execução.

A seguir, a comparação entre os resultados da implementação e os resultados obtidos através do modelo SAN correspondente é apresentada para cada estudo de caso, considerando a variação do número de tarefas. Ao final da apresentação dos resultados de cada estudo de caso, será realizada uma análise a respeito da adaptabilidade dos modelos SAN ao seu respectivo estudo de caso.

7.1 Estudo de Caso 1 - Multiplicação de Matrizes

7.1.1 Modelo Mestre/Escravo

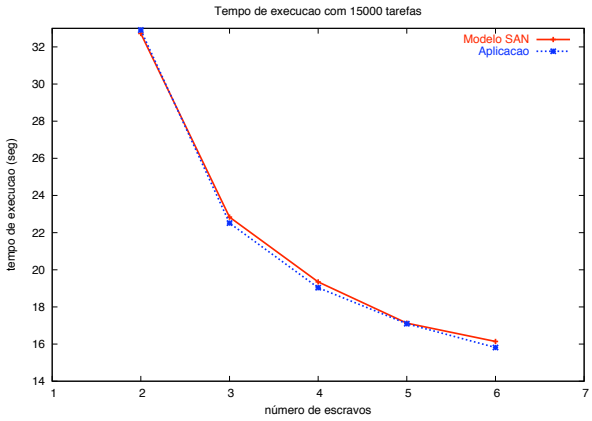
A figura 26 mostra os tempos de execução para o modelo SAN mestre escravo executando um número pequeno de tarefas (pequeno) com diferente número de escravos (de 2 à 6). Neste gráfico, ressalta-se os tempos de execução obtidos da atual implementação paralela e os resultados computados do modelo SAN proposto. É possível observar que para todas as configurações da implementação paralela o modelo SAN e os resultados experimentais são bem similares. Mesmo que a forma da curvas não seja a mesma para um número maior de escravos, ela oferece tempos de execução muito próximos (uma diferença menor que 1 segundo). Este fato mostra que o modelo de programação mestre escravo para o primeiro estudo de caso é eficiente. Ainda, tanto pelo modelo SAN quanto pela aplicação paralela, pode-se observar que o desempenho da aplicação cresce conforme o número de escravos aumenta, indicando um provável aumento de desempenho com mais de 6 escravos.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	19.8146	13.3426	9.9577	8.6348	7.1490
Aplicação (seg.)	19.9723	13.3313	9.9935	8.2459	7.0356
Seqüencial: 0.0339					

Figura 26: Resultados do modelo mestre/escravo (10.000 tarefas).

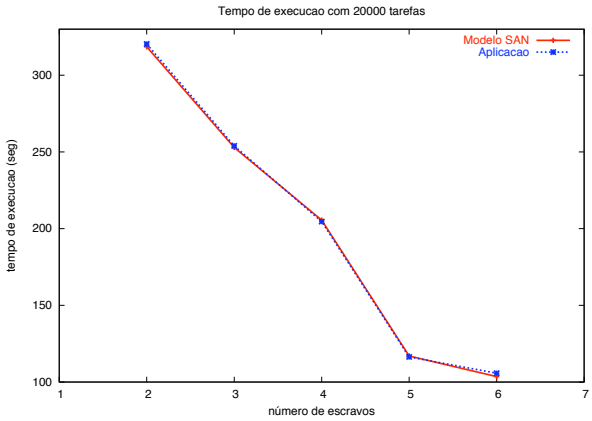
Os outros dois casos, que apresentam um aumento no número de tarefas (figura 27 e figura 28), mostram mesmas semelhanças nos resultados, resultando em um comportamento geral (formato das curvas) muito similar. Observando a figura 27, pode-se afirmar que para este número de tarefas, a melhor configuração tende a ser também com mais de 6 escravos, ou seja, com um número maior de escravos esta aplicação provavelmente obterá ganhos relevantes de desempenho.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	32.7234	22.8345	19.3461	17.1256	16.1394
Aplicação (seg.)	32.9084	22.5134	19.0358	17.0958	15.8134
Seqüencial: 0.3012					

Figura 27: Resultados do modelo mestre/escravo (15.000 tarefas).

A figura 28 reforça esta afirmação relacionada à equivalência das tendências das curvas. Além disso, este gráfico mostra que a aplicação quando executada com um número muito grande de tarefas irá obter bons desempenhos com mais do que 6 escravos. Vale lembrar que os gráficos aqui apresentados não passam de 7 processos (6 escravos e um mestre neste caso) devido a limitação do formalismo escolhido.



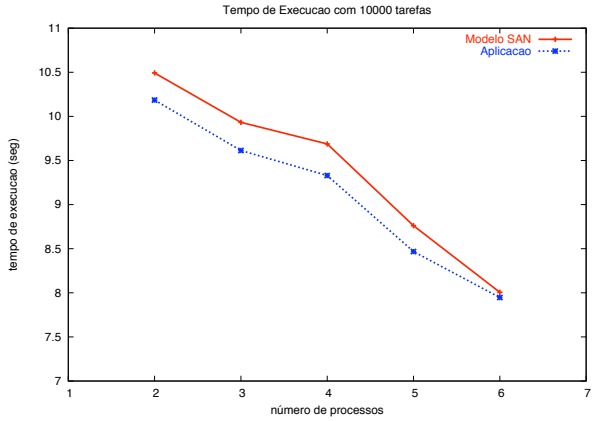
# escravos	2	3	4	5	6
Modelo SAN (seg.)	318.5786	252.9125	205.5421	116.9873	103.5609
Aplicação (seg.)	320.3468	253.9348	204.6087	116.3650	105.7512
Seqüencial: 1.3988					

Figura 28: Resultados do modelo mestre/escravo (20.000 tarefas).

Portanto, pode-se perceber que o modelo de programação mestre/escravo mostrou-se uma boa solução para a aplicação de multiplicação de matrizes em paralelo. Esta análise é comprovada tanto pelos resultados obtidos através da modelagem analítica como na implementação desta aplicação. Além disso, conseguiu-se comprovar a eficiência deste modelo de programação para este estudo de caso. O próximo passo para a validação deste modelo SAN será sua utilização em outros estudos de caso. A seguir, os resultados do modelo SAN de fases paralelas são apresentados, juntamente com uma análise detalhada do comportamento destes resultados.

7.1.2 Modelo de Fases Paralelas

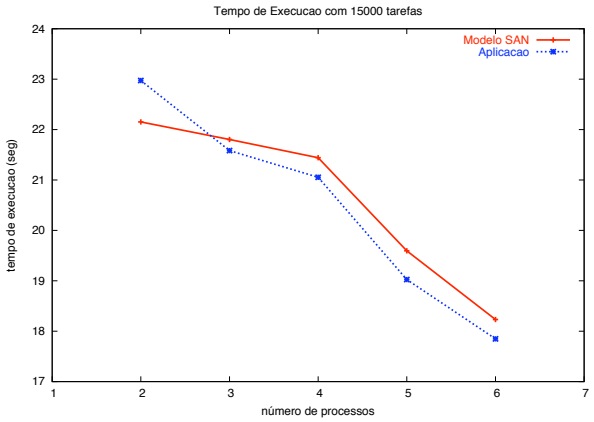
A figura 29 mostra os tempos de execução para o modelo SAN de fases paralelas executando um número pequeno de tarefas (pequeno) com diferente número de processos (de 2 à 6).



# escravos	2	3	4	5	6
Modelo SAN (seg.)	10.4927	9.9324	9.6880	8.7610	8.0060
Aplicação (seg.)	10.1842	9.6131	9.3296	8.4675	7.9467

Figura 29: Resultados do modelo de fases paralelas (10.000 tarefas).

Os resultados obtidos utilizando este modelo de programação apresentam melhor desempenho se comparados com os resultados do modelo SAN mestre/escravo, comprovando que o modelo de programação de fases paralelas é mais adequado para aplicações que utilizem multiplicação de matrizes. Além disso, quanto maior for o tamanho das matrizes, maior é a diferença de desempenho entre estes dois modelos de programação. As figuras 30 e 31 confirmam esta análise.

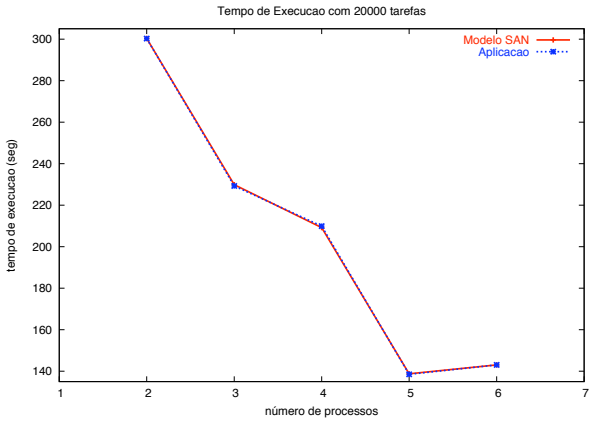


# escravos	2	3	4	5	6
Modelo SAN (seg.)	22.1524	21.8031	21.4419	19.5951	18.2308
Aplicação (seg.)	22.9732	21.5841	21.0533	19.0244	17.8483

Figura 30: Resultados do modelo de fases paralelas (15.000 tarefas).

Embora as curvas do modelo SAN e da aplicação tenham formatos parecidos, este modelo de programação perdeu desempenho entre 3 e 4 processos. Este fato ocorre devido a distribuição do número de tarefas para cada escravo. Observe que tanto para o número de tarefas pequeno (10.000 tarefas) quanto para o médio (15.000 tarefas), o número de tarefas não é divisível por 4. Portanto, alguns processos ficarão com mais tarefas do que outros, indicando uma queda de desempenho. Observe ainda que com 5 processos, o desempenho aumenta em todos os gráficos devido a exata divisão do número de tarefas pelo número de processos.

Observe que no gráfico 30 há um cruzamento entre as curvas do modelo SAN e da implementação paralela. Uma explicação viável para este fato é o possível erro estatístico na realização das médias dos valores associados as taxas do modelo SAN. Em outras palavras, quando foram realizadas as médias dos valores para parametrizar as taxas, houveram diferenças significativas nas 10 execuções realizadas. Logo, este erro estatístico influenciou nos resultados dos modelos SAN. Mais adiante o leitor irá perceber que este cruzamento irá acontecer mais freqüentemente nas execuções com um número de tarefas pequeno. O motivo deste acontecimento é a baixa escala utilizada nestes gráficos, ou seja, nos gráficos com números de tarefas maiores possuem escalas maiores. Outro motivo é o fato de o erro estatístico influenciar mais em tempos de execução menores (número de tarefas pequeno).



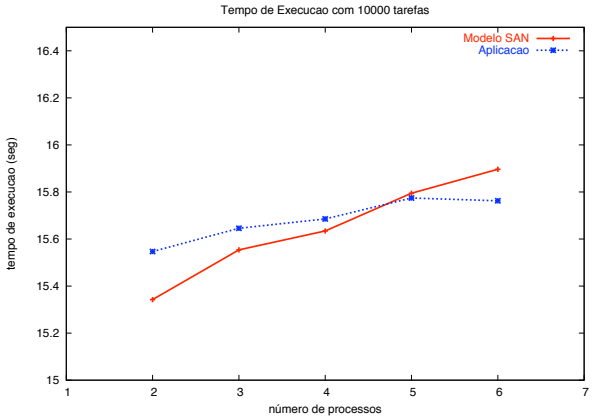
# escravos	2	3	4	5	6
Modelo SAN (seg.)	300.3462	229.8745	209.2051	138.7734	143.0566
Aplicação (seg.)	300.2563	229.2501	209.9381	138.5099	143.0454

Figura 31: Resultados do modelo de fases paralelas (20.000 tarefas).

Mesmo assim, o modelo de programação de fases paralelas conseguiu atingir tempos de execução bem próximos dos resultados da aplicação paralela. Esta análise demonstra a boa usabilidade do formalismo SAN para prever desempenho de aplicações paralelas. Observando os valores expressados nestas tabelas, pôde-se confirmar uma predição com um erro máximo de $\simeq 3\%$, indicando curvas com formatos bem próximos.

7.1.3 Modelo *Pipeline*

A figura 32 mostra os tempos de execução para o modelo SAN *pipeline* executando um número pequeno de tarefas (100 tarefas) com diferente número de processos (de 2 à 6).

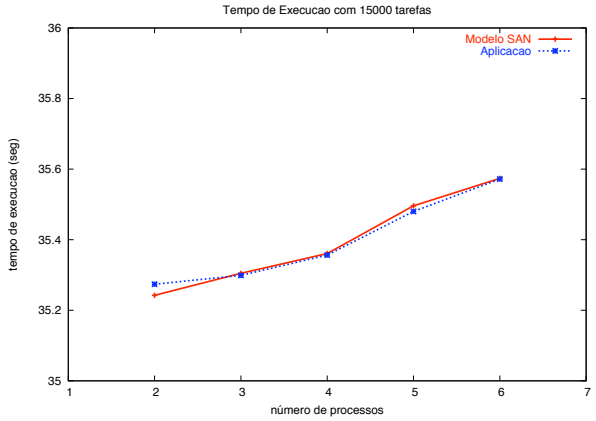


# escravos	2	3	4	5	6
Modelo SAN (seg.)	15.3424	15.5540	15.6342	15.7948	15.8965
Aplicação (seg.)	15.5469	15.6456	15.6853	15.7742	15.7623

Figura 32: Resultados do modelo *pipeline* (10.000 tarefas).

Este modelo de programação possui resultados que aumentam conforme aumenta o

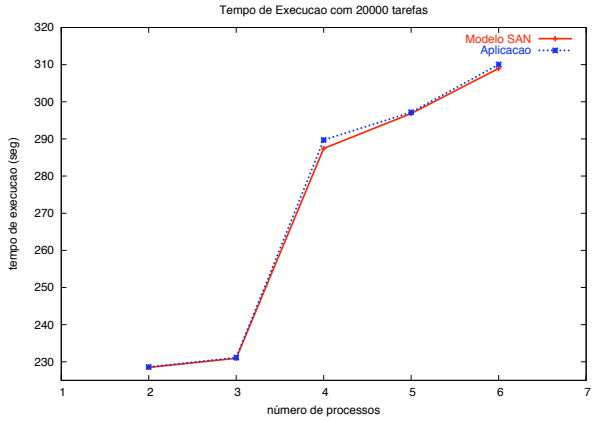
número de processos. Em outras palavras, não há ganho de desempenho para a multiplicação de matrizes utilizando o modelo de programação *pipeline*, pois este modelo de programação não é apropriado para este tipo de aplicação. Contudo, optou-se por escolher este modelo para mostrar que os modelos SAN conseguem identificar quais são as melhores e as piores escolhas em uma implementação paralela.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	35.2422	35.3047	35.3607	35.4959	35.5737
Aplicação (seg.)	35.2737	35.2989	35.3563	35.4799	35.5719

Figura 33: Resultados do modelo *pipeline* (15.000 tarefas).

A figura 33 e a figura 34 confirmam a má escolha do modelo de programação *pipeline* para a aplicação escolhida como estudo de caso. Observe que em nenhum momento, os modelos SAN apresentaram incoerência com os resultados coletados na execução das aplicações paralelas.

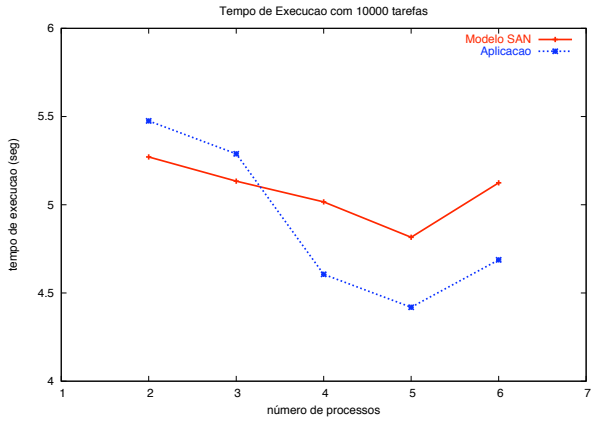


# escravos	2	3	4	5	6
Modelo SAN (seg.)	228.4824	230.9534	287.4237	296.8429	308.9943
Aplicação (seg.)	228.5900	231.1187	289.7240	297.1707	310.0497

Figura 34: Resultados do modelo *pipeline* (20.000 tarefas).

7.1.4 Modelo Divisão e Conquista

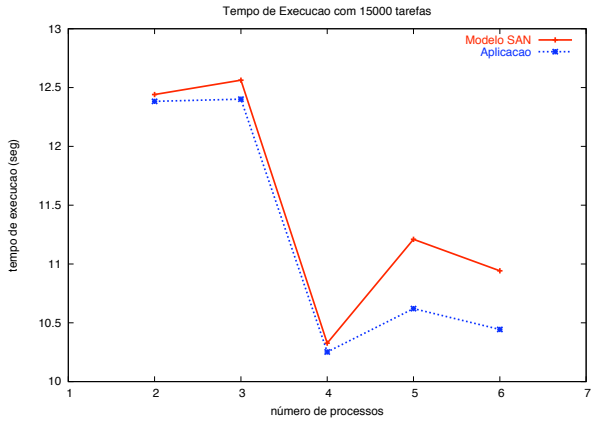
Por fim, esta seção apresenta os tempos de execução obtidos através do modelo SAN para divisão e conquista. A figura 35 mostra os tempos de execução para o modelo divisão e conquista executando um número pequeno de tarefas (pequeno) com diferente número de processos (de 2 à 6).



# escravos	2	3	4	5	6
Modelo SAN (seg.)	5.2711	5.1338	5.0161	4.8157	5.1244
Aplicação (seg.)	5.4750	5.2887	4.6059	4.4190	4.6875

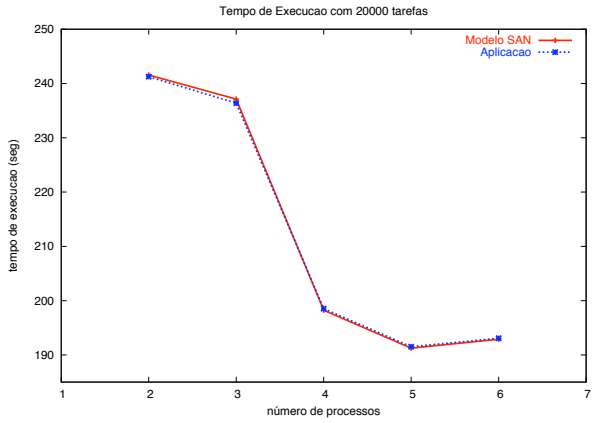
Figura 35: Resultados do modelo divisão e conquista (10.000 tarefas).

Por este modelo de programação ter uma característica diferenciada (balanceamento de carga diferente conforme o número de processos) em relação aos outros modelos apresentados anteriormente, os resultados apontam também resultados diferentes. Para a execução com o número de tarefas pequeno, este modelo de programação não apresenta ganho significativo de desempenho com o aumento de processos. A explicação deste fato é a má distribuição de carga entre os processos. Em outras palavras, a hierarquia exigida neste modelo possui um balanceamento de carga heterogêneo em quase todos os casos, afetando significativamente o desempenho da aplicação. O mesmo ocorre à medida que o número de tarefas aumenta, o que mostram a figura 36 e a figura 37.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	12.4410	12.5631	10.3269	11.2100	10.9421
Aplicação (seg.)	12.3826	12.4015	10.2518	10.6207	10.4433

Figura 36: Resultados do modelo divisão e conquista (15.000 tarefas).



# escravos	2	3	4	5	6
Modelo SAN (seg.)	241.5654	237.1268	198.2346	191.2468	192.8924
Aplicação (seg.)	241.2510	236.3827	198.5282	191.5144	193.0766

Figura 37: Resultados do modelo divisão e conquista (20.000 tarefas).

Neste dois casos, assim como no caso de pequeno número de tarefas, consegue-se obter tempos de execução menores com 4 processos tanto com o número médio de tarefas, como com o número de tarefas grande. O motivo pelo qual pode-se obter maiores quedas no tempo de execução com 4 processos é a divisão mais homogênea de tarefas, pois utilizando-se a hierarquia todos os processos possuirão carga semelhante, e assim pode-se atingir *speed-ups* consideráveis. Mesmo com esta mudança de desempenho, conforme o número de tarefas aumenta, os resultados do modelo SAN conseguem ser coerentes com os resultados da aplicação paralela. Por exemplo, os modelos SAN conseguiram encontrar os pontos de inflexão nos dois primeiros gráficos da mesma forma que as curvas das aplicações apresentaram.

Por este modelo SAN ser o mais complexo dos quatro modelos genéricos propostos, pôde-se perceber uma maior discrepância dos resultados obtidos. Da mesma forma que

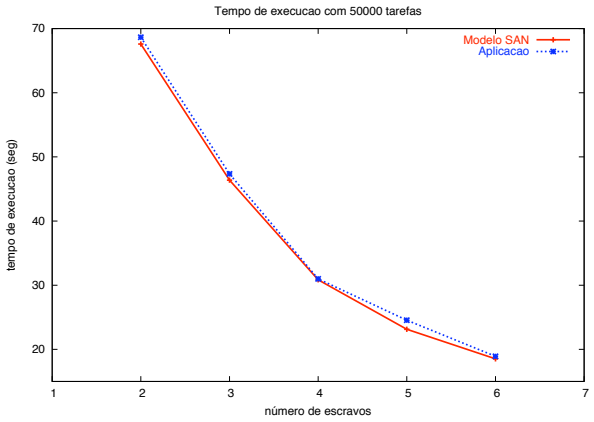
os outros modelos SAN, para validar o modelo de divisão e conquista é necessário utilizá-lo em outros estudos de caso, o que será realizado a seguir. Por fim, observou-se que os modelos de programação mestre/escravo e fases paralelas foram os que apresentaram curvas com possível crescimento de desempenho conforme o número de escravos aumenta. Assim, pode-se dizer que estes dois modelos são os mais indicados para a aplicação deste estudo de caso.

7.2 Estudo de Caso 2 - Ordenação de Vetores

A segunda aplicação escolhida como estudo de caso é, similarmente à multiplicação de matrizes, considerada clássica em computação de alto desempenho, por ser facilmente paralelizável. Como teste, utilizou-se uma variação no número de tarefas de três formas distintas: 50.000 (pequeno), 100.000 (médio) e 150.000 (grande) tarefas. As execuções foram realizadas variando o número de processos de 2 até 6. Os quatro modelos SAN genéricos foram utilizados para modelar esta aplicação, com o intuito de verificar a eficiência e a aplicabilidade destes modelos neste tipo de aplicação. A seguir, os resultados do modelo SAN mestre/escravo é analisado detalhadamente.

7.2.1 Modelo Mestre/Escravo

A figura 38 apresenta os tempos de execução da ordenação de vetor em paralelo e da estimativa obtida através dos modelos SAN, utilizando pequeno número de tarefas.

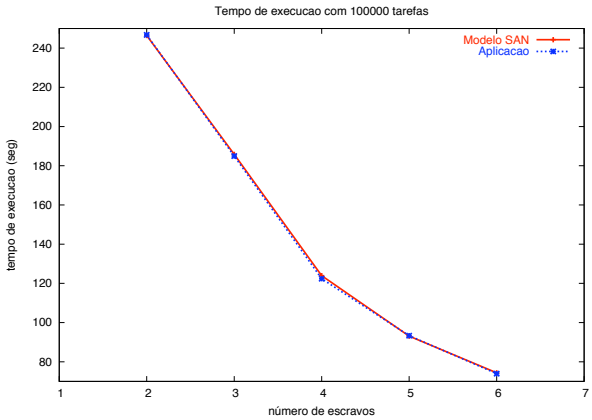


# escravos	2	3	4	5	6
Modelo SAN (seg.)	67.5937	46.3813	30.8435	23.1272	18.5227
Aplicação (seg.)	68.6465	47.3502	30.9842	24.5513	18.9213

Figura 38: Resultados do modelo mestre/escravo (50.000 tarefas).

Para o número de tarefas pequeno, pode-se observar curvas com tendências bem próximas, indicando uma boa qualidade quanto a predição de resultados do modelo SAN mestre/escravo. Além disso, as duas curvas apresentam resultados favoráveis quanto ao

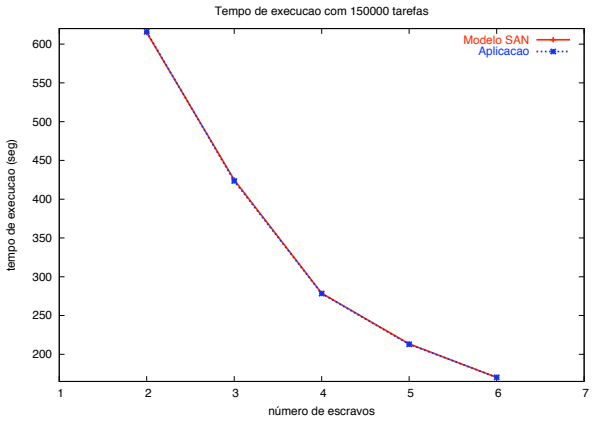
uso do modelo de programação mestre/escravo para a aplicação de ordenação de vetores em paralelo, informando um possível aumento de desempenho se executado com mais de 6 escravos. Somente na execução com 5 escravos as duas curvas obtiveram uma distância um pouco maior. Mesmo assim, a diferença não é significativa de forma que influencie em alguma análise.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	246.4589	185.7836	123.8980	93.1127	74.3511
Aplicação (seg.)	246.7582	184.9314	122.3856	93.2985	73.9596

Figura 39: Resultados do modelo mestre/escravo (100.000 tarefas).

A figura 39 e a figura 40 apresentam os tempos de execução da ordenação de vetor em paralelo utilizando médio número de tarefas e grande número de tarefas, respectivamente. Nestes dois gráficos, confirma-se as afirmações realizadas com número de tarefas pequeno. Na verdade, estes resultados enfatizam os uso do modelo de programação mestre/escravo quando há a necessidade de ordenação de vetores.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	614.9147	424.9083	278.6913	213.3996	170.1613
Aplicação (seg.)	615.8890	423.4599	278.3421	212.8724	170.3277

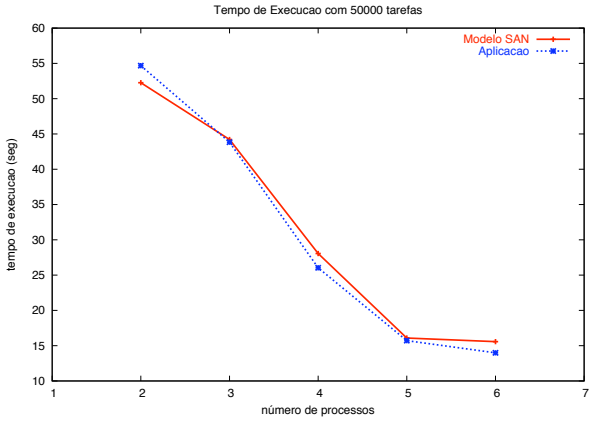
Figura 40: Resultados do modelo mestre/escravo (150.000 tarefas).

Os resultados obtidos com os modelos SAN foram em todos os momentos próximos dos resultados da implementação, ou seja, sem perda considerável de precisão. Assim, para

todos os números de tarefas testados, o modelo de programação mestre/escravo conseguiu reduzir consideravelmente o tempo de execução da aplicação de estudo de caso conforme esperado.

7.2.2 Modelo de Fases Paralelas

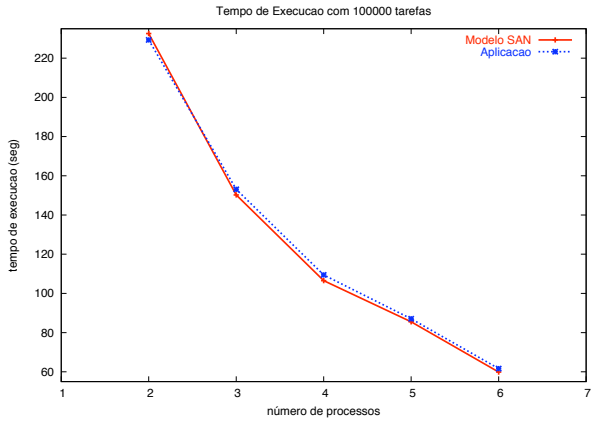
A figura 41 apresenta os tempos de execução da ordenação de vetor em paralelo e da estimativa obtida através dos modelos SAN de fases paralelas, utilizando pequeno número de tarefas.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	52.2573	44.2089	28.0569	16.0791	15.5665
Aplicação (seg.)	54.6739	43.8347	26.0387	15.7098	13.9865

Figura 41: Resultados do modelo de fases paralelas (50.000 tarefas).

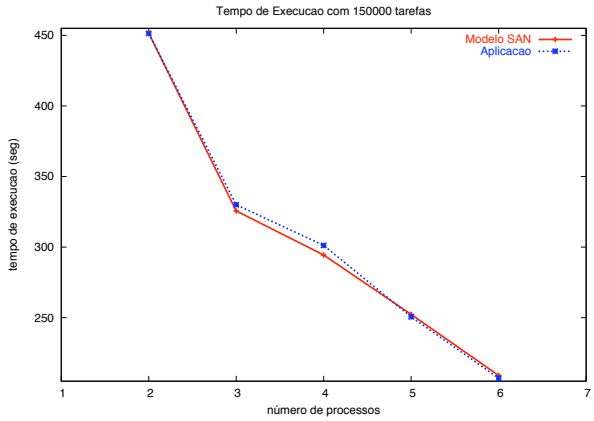
Os resultados obtidos com este modelo de programação indicam a tendência positiva no uso do formalismo SAN para modelar aplicações paralelas utilizando o modelo de programação de fases paralelas. Em outras palavras, tanto para o primeiro estudo de caso como para este estudo de caso em questão, o modelo SAN de fases paralelas mostrou-se preciso. Utilizando um pequeno número de tarefas (aproximadamente 50.000 tarefas), o modelo de fases paralelas apresentam ganho de desempenho com até 5 processos, o que não ocorre para os outros tamanhos de tarefas utilizados, como mostra a figura 42 e a figura 43.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	232.5549	150.2105	106.5309	85.5340	59.9001
Aplicação (seg.)	229.3065	153.1093	109.4250	87.0589	61.6392

Figura 42: Resultados do modelo de fases paralelas (100.000 tarefas).

Como analisado anteriormente, o modelo de programação de fases paralelas pode apresentar aumento de desempenho com mais de 6 processos, utilizando número médio e grande de tarefas. Além disso, a semelhança das curvas para os dois últimos grãos (médio e grande) justifica o bom uso deste modelo, pois em todas as execuções houve ganho de desempenho. Porém, para o grão pequeno houve uma pequena diferença nos resultados se analisados ponto a ponto. Mesmo assim, a tendência das curvas são similares, indicando que o modelo SAN de fases paralelas conseguiu atingir os resultados esperados.



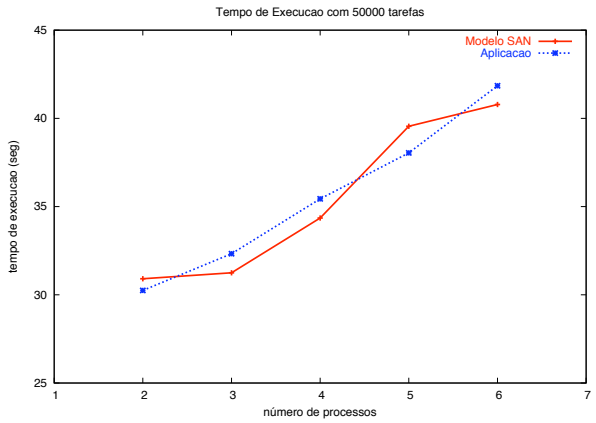
# escravos	2	3	4	5	6
Modelo SAN (seg.)	451.6765	325.6239	294.4325	252.3290	209.0552
Aplicação (seg.)	451.3449	330.0284	301.1893	250.5698	207.1293

Figura 43: Resultados do modelo de fases paralelas (150.000 tarefas).

Por fim, para a estudo de caso de ordenação de vetores o modelo de fases paralelas mostrou-se tão útil quanto o modelo mestre/escravo, pois a aceleração das curvas entre os modelos são parecidas. Assim, tanto o modelo de fases paralelas quanto o modelo mestre/escravo podem ser utilizados para implementar a ordenação de vetores em paralelo.

7.2.3 Modelo *Pipeline*

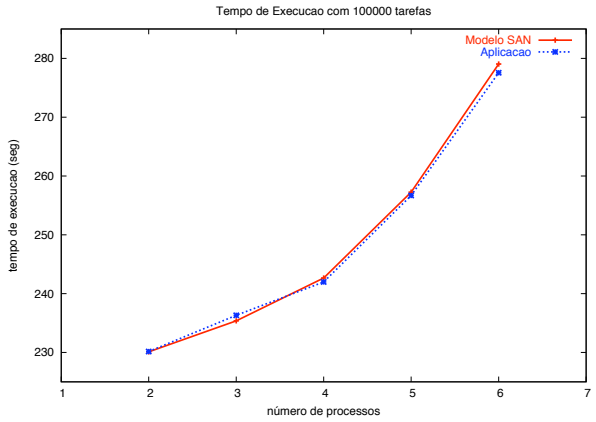
A figura 44 apresenta os tempos de execução da ordenação de vetor em paralelo e da estimativa obtida através dos modelos SAN *pipeline*, utilizando pequeno número de tarefas.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	30.9106	31.2498	34.3509	39.5505	40.7876
Aplicação (seg.)	30.2487	32.3265	35.4390	38.0435	41.8462

Figura 44: Resultados do modelo *pipeline* (50.000 tarefas).

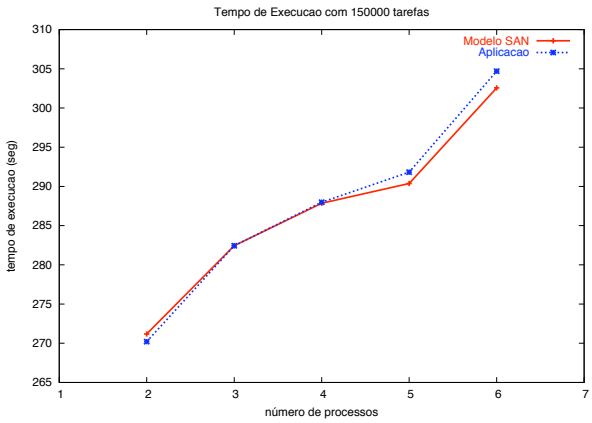
Como previsto e mostrado no estudo de caso de multiplicação de matrizes em paralelo, o modelo de programação *pipeline* não é adequado para a implementação deste tipo de aplicação paralela. Teoricamente, quanto maior for o número de processos maior será o tempo de execução da aplicação. Este fato é comprovado nos gráficos do modelo *pipeline*. Mesmo assim, o modelo SAN conseguiu apresentar novamente precisão nos resultados, se aproximando dos valores extraídos da implementação paralela. Porém, o modelo SAN para o grão pequeno obteve resultados menos precisos do que com o grão médio (figura 45) e grão grande (figura 46).



# escravos	2	3	4	5	6
Modelo SAN (seg.)	230.0939	235.3687	242.6545	257.2543	279.0428
Aplicação (seg.)	230.1520	236.3160	241.9914	256.6857	277.5548

Figura 45: Resultados do modelo *pipeline* (100.000 tarefas).

Uma justificativa plausível para o fato da menor precisão utilizando grão pequeno é o inferior tempo de execução, como mencionado anteriormente. Quando o tempo de execução aumenta, o número de fatores coletados para formar um resultado é maior, ou seja, a precisão do resultado é maior. Um número de tarefas pequeno traz como conseqüência um tempo de execução pequeno se comparados a números de tarefas maiores. Portanto, o grão pequeno aqui utilizado pode apresentar precisão nos resultados menor do que os outros dois grãos.



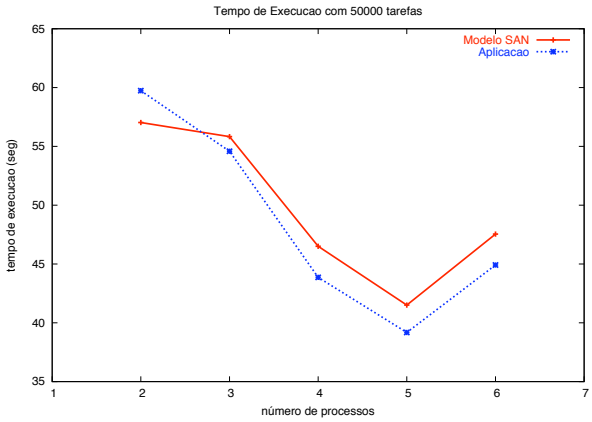
# escravos	2	3	4	5	6
Modelo SAN (seg.)	271.1874	282.4379	287.8546	290.3684	302.5684
Aplicação (seg.)	270.2109	282.4309	287.9804	291.8186	304.6968

Figura 46: Resultados do modelo *pipeline* (150.000 tarefas).

Desta forma, o modelo de programação *pipeline* não é aconselhável para implementar as duas aplicações de estudo de caso analisadas até aqui. Por fim, vale salientar que este modelo foi utilizado sabendo o seu desempenho prévio. Como dito anteriormente, este modelo (assim como os outros) foi utilizado com o intuito de validar a utilização do formalismo SAN para modelar aplicações paralelas, tanto nas boas como nas más escolhas de modelos de programação.

7.2.4 Modelo Divisão e Conquista

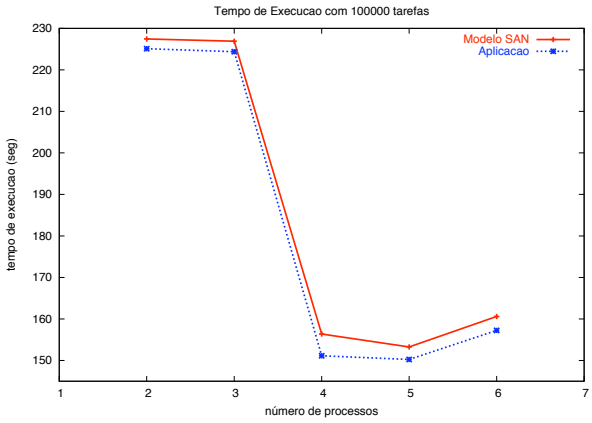
A figura 47 apresenta os tempos de execução da ordenação de vetor em paralelo e da estimativa obtida através dos modelos SAN divisão e conquista, utilizando pequeno número de tarefas.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	57.0264	55.8271	46.5098	41.5092	47.5402
Aplicação (seg.)	59.7349	54.5879	43.8621	39.1786	44.9147

Figura 47: Resultados do modelo divisão e conquista (50.000 tarefas).

Mais uma vez, o modelo SAN que implementa a divisão e conquista conseguiu atingir um formato de curva semelhante ao extraído da implementação real. No caso da ordenação de vetor em paralelo utilizando este modelo de programação e um vetor de entrada de tamanho igual a 50.000, o desempenho com mais do que 5 processos não é aconselhável, o que indica as duas curvas através do ponto de inflexão entre 5 e 6 processos. Além disso, como analisado no estudo de caso da multiplicação de matrizes em paralelo, o maior ganho de desempenho foi entre 3 e 4 processos. Este fato fica mais claro à medida que o tamanho do vetor (número de tarefas) aumenta, conforme mostram a figura 48 para o número de tarefas médio e a figura 49 para o número de tarefas grande.

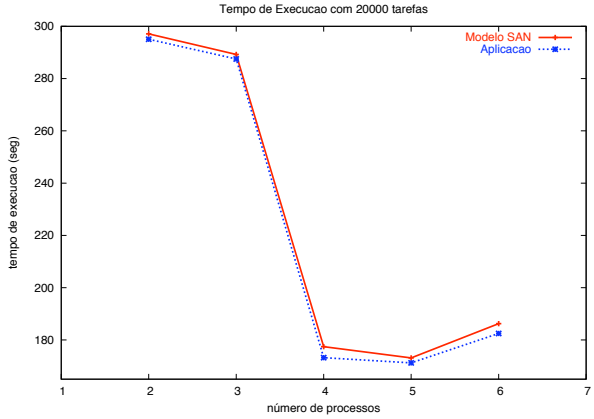


# escravos	2	3	4	5	6
Modelo SAN (seg.)	227.4508	226.9028	156.3729	153.2476	160.6033
Aplicação (seg.)	225.0985	224.3694	151.1409	150.2410	157.2367

Figura 48: Resultados do modelo divisão e conquista (100.000 tarefas).

A explicação para o fato deste modelo de programação apresentar melhor desempenho com 4 processos é igual ao descrito no primeiro estudo de caso, *i.e.*, com este número de

processos a distribuição de carga fica equilibrada, enquanto que com outros processos ela fica desbalanceada.



# escravos	2	3	4	5	6
Modelo SAN (seg.)	297.0984	289.2407	177.4480	173.1274	186.2387
Aplicação (seg.)	295.0412	287.5098	173.2154	171.2409	182.4982

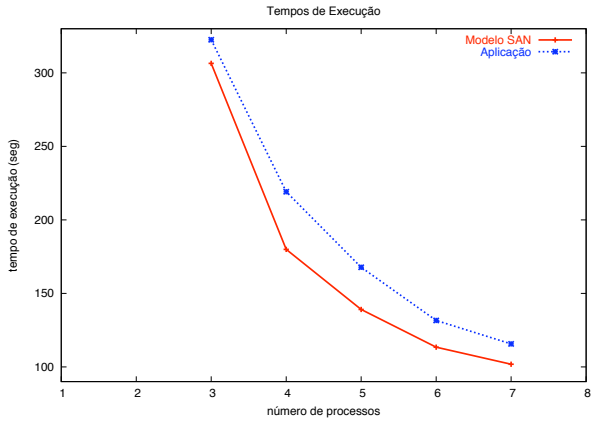
Figura 49: Resultados do modelo divisão e conquista (150.000 tarefas).

De qualquer forma, o modelo SAN de divisão e conquista conseguiu reduzir significativamente o tempo de execução dos dois estudos de caso apresentados até agora. Porém, fazendo uma análise geral de todos os modelos aqui apresentados, aconselha-se o uso tanto do modelo mestre/escravo como do modelo de fases paralelas para estes dois estudos de caso, uma vez que estes podem ainda apresentar maiores reduções nos tempos de execução com mais processos. Por fim, vale ressaltar que em todos os casos de estudo com o quatro modelos SAN obteve-se resultados semelhantes aos das aplicações reais, indicando que o formalismo SAN pode ser utilizado perfeitamente para modelar aplicações paralelas e tomar decisões de seus resultados para uma eventual implementação.

7.3 Estudo de Caso 3 - Simulação de Elétrons em um Dispositivo FED

Como dito anteriormente, a aplicação escolhida como terceiro estudo de caso tem como objetivo simular paralelamente a emissão de elétrons em um dispositivo FED. Como teste utilizou-se uma variação de elétrons por tarefas de três formas distintas: 2 elétrons (grão pequeno), 5 elétrons (grão médio) e 10 elétrons (grão grande) por tarefa. Diversas execuções foram realizadas variando o número de processos de 3 até 7. A figura 50 apresenta os tempos de execução da simulação paralela e da estimativa obtida através dos modelos SAN.

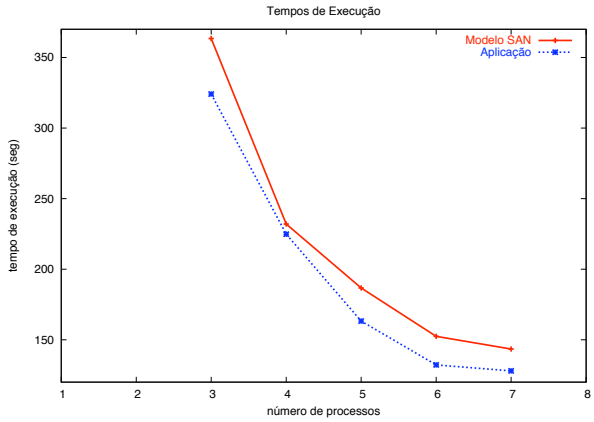
Com esses resultados, pode-se afirmar que os modelos SAN conseguiram atingir resultados bem próximos aos da simulação paralela, indicando uma boa modelagem e, acima



# processos	3	4	5	6	7
Modelo SAN (seg.)	306.4356	179.9348	139.1018	113.4254	101.8423
Aplicação (seg.)	322.51	219.136	167.774	131.612	115.679

Figura 50: Aplicação FED com 2 elétrons por tarefa

de tudo, parametrização das taxas. Contudo, ainda existe uma discrepância nos resultados, uma vez que estes modelos não preveem interferências de outros dispositivos (*e.g.*, *IO*) que podem afetar o tempo de execução da aplicação. Mesmo assim, podemos afirmar que as tendências das duas curvas são praticamente a mesma, mostrando que a simulação paralela da aplicação FED obterá desempenhos significativos com mais de 7 processos.

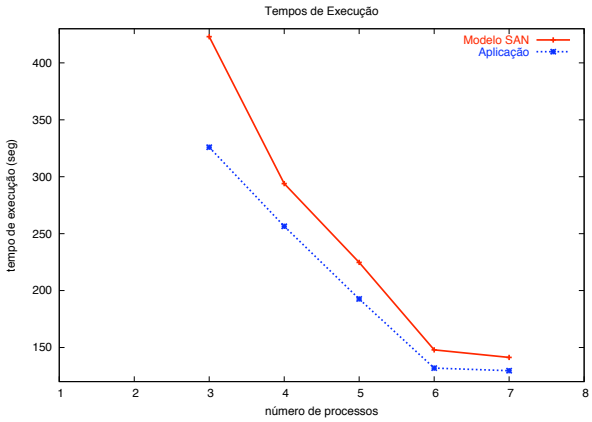


# processos	3	4	5	6	7
Modelo SAN (seg.)	363.4751	232.0454	186.7421	152.4418	143.4534
Aplicação (seg.)	324.08	224.8	163.262	132.19	127.978

Figura 51: Aplicação FED com 5 elétrons por tarefa

A figura 51 apresenta os resultados utilizando o grão médio. Note que nesta figura as curvas estão invertidas se comparadas com os resultados apresentados anteriormente. Este fato ocorre devido aos diferentes valores coletados para parametrização, isto é, as taxas para os diferentes grãos não são iguais. Porém, a tendência das curvas é semelhante, reforçando a afirmação supracitada.

Vale lembrar que a mudança da posição das curvas da figura 50 para a figura 51 não é um fato inesperado e não influência na análise dos resultados, uma vez que a principal



# processos	3	4	5	6	7
Modelo SAN (seg.)	423.0753	293.9822	224.7410	147.9227	141.2340
Aplicação (seg.)	325.915	256.478	192.595	131.87	129.629

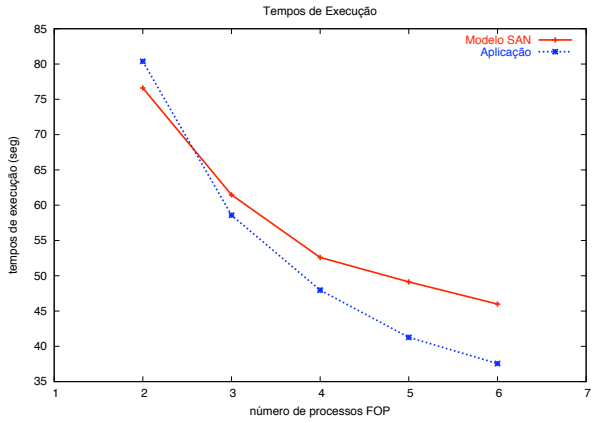
Figura 52: Aplicação FED com 10 elétrons por tarefa

preocupação é a avaliação comportamental das curvas. Assim, podemos dizer que o grão pequeno é a melhor opção se comparado com o grão médio e o grão grande (figura 52), ao qual apresentou melhores tempos de execução. Observe que para os dois grãos com pior desempenho, o tempo de execução com 3 processos apresentado pelos modelos SAN são muito distantes dos resultados da simulação. Uma explicação para isto é a interferência de fatores externos na medida das taxas.

7.4 Estudo de Caso 4 - Renderização de Documentos utilizando FOP

Para esta aplicação, foram escolhidos casos de testes variando o número de tarefas de duas formas: 165 (primeiro teste) e 337 (segundo teste) tarefas ao todo. Para cada tarefa, variou-se o número de processos FOP de 1 até 6 e manteve-se o número de processos Broker e Consumer em 1 cada. Ainda, para o primeiro teste o número de FOs por tarefa é em média 25. Já para o segundo teste o número de FOs fica em média de 11. Assim, temos no primeiro teste uma média de 4.125 FOs a serem renderizados e para o segundo teste uma média de 3.707 FOs. A figura 53 apresenta os resultados obtidos com o modelo SAN e a aplicação para o caso de teste com 165 tarefas.

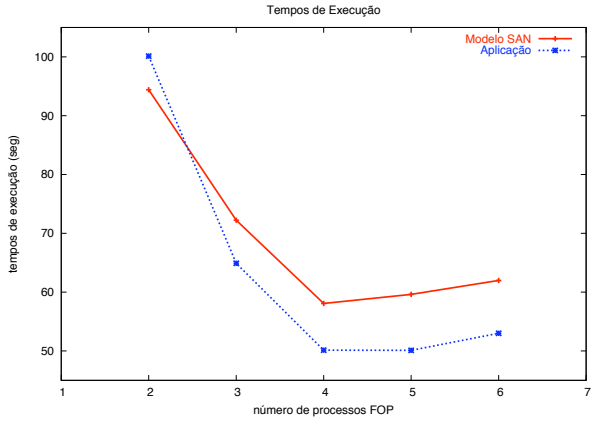
No gráfico com 165 tarefas, pode-se observar que os tempos de execução do modelo SAN e da aplicação são semelhantes, indicando uma boa modelagem da realidade. Em uma análise mais detalhada, as duas curvas apresentadas se cruzam, o que não aconteceu nos resultados apresentados na seção anterior (estudo de caso 3). Entretanto, mesmo com esse cruzamento - que pode ter sido novamente influenciado pela parametrização das taxas - as tendências das duas curvas são semelhantes, indicando que para este estudo de caso mais de 6 processos FOP serão necessário para obter um desempenho ótimo. Mais



# processos	3	4	5	6	7
Modelo SAN (seg.)	76.6103	61.4700	52.5974	49.1345	45.9881
Aplicação (seg.)	80.3804	58.5874	47.9659	41.2703	37.5643

Figura 53: Aplicação FOP com 165 tarefas

adiante, na figura 54 pode-se observar os resultados extraídos dos modelos SAN e da aplicação para o outro caso de teste (com 337 tarefas).



# processos	3	4	5	6	7
Modelo SAN (seg.)	94.4119	72.2253	58.0697	59.6177	61.9834
Aplicação (seg.)	100.1283	64.8983	50.1278	50.0933	53.0000

Figura 54: Aplicação FOP com 337 tarefas

Esta última figura reforça a análise dos resultados realizados anteriormente. A primeira diferença apresentada nesta figura é o aumento no tempo de execução já com 4 FOPs, indicando que a aplicação não obterá desempenhos melhores com mais de 4 processos FOP. Apesar do segundo teste apresentar número de FOs a serem renderizados (3.707 FOs) menor do que o primeiro teste (4.125 FOs), os tempos de execução para este segundo teste são maiores. Uma justificativa para este fato está relacionado a granularidade das tarefas, *i.e.*, a utilização de 25 FOs por tarefa apresenta resultados melhores do que 11 FOs por tarefa. Por fim, pode-se notar que os modelos SAN conseguiram mostrar características semelhantes as coletadas da aplicação e analisadas nesta seção, como por exemplo, o ponto de inflexão apresentado na figura 54.

Capítulo 8

Conclusão

Este trabalho apresentou modelos SAN genéricos para diferentes modelos de programação de aplicações paralelas em máquinas agregadas. O uso de um formalismo estocástico para modelar sistemas complexos nos parece fornecer vantagens em comparação a outras modelagens matemáticas, tais como complexidade de algoritmos paralelas através da utilização de modelos PRAM ou BSP, por exemplo. O uso de complexidade de algoritmos necessita de valores absolutos para que o cálculo dos resultados sejam mais corretos. De outra forma, modelos estocásticos trabalham com distribuição exponencial e, assim, necessitam somente de valores aproximados, ou médias estatísticas, para o cálculo de resultados.

Na área de processamento paralelo, o uso de formalismos estocásticos é freqüentemente limitado pelo tamanho (em número de estados) do problema. Nós acreditamos que modelos SAN são uma boa abordagem para contornar este problema e o estudo de modelagem apresentado neste trabalho para modelar aplicações para ambientes de alto desempenho, ou mais especificamente máquinas agregadas, pôde ser desenvolvido por não especialistas em avaliação de desempenho. Este fato reforça a facilidade do uso de modelos SAN para prever desempenho de aplicações em diferentes áreas.

Quatro modelos SAN genéricos foram apresentados neste trabalho, cada um representando um modelo de programação diferente: mestre/escravo, fases paralelas, *pipeline* e divisão e conquista. Utilizou-se quatro aplicações como estudo de caso para verificar a adaptabilidade do formalismo de Redes de Autômatos estocásticos para modelar aplicações paralelas. Em todos os experimentos, obteve-se precisão nos resultados bem próximas dos tempos de execução reais das aplicações. A tabela 8 afirma este fato, onde apresenta os erros máximos para cada modelo SAN em cada estudo de caso realizado neste trabalho.

A maior desvantagem desta abordagem está ligada à importância atribuída à fase de parametrização, *i.e.*, a escolha das taxas para os eventos de um modelo SAN. A maior parte da precisão dos modelos se encontra no mapeamento das características conhecidas pelo usuário para os valores das taxas numéricas dos eventos. Na verdade, a técnica de modelagem pode somente auxiliar o programador indicando quais taxas devem ser

Tabela 2: Tabela com erros máximos para cada Estudo de Caso.

Estudo de Caso / Modelo SAN	Mestre/Escravo	Fases Paralelas	*Pipeline*	Divisão e Conquista
Multiplicação de Matrizes	4.7162%	3.5728%	1.3153%	8.9059%
Ordenação de Vetores	5.8005%	4.4309%	3.3307%	6.0341%
Simulação em FED	29.8115%	-	-	-
Renderização em FOP	22.4250%	-	-	-

informadas. Resta para o programador o trabalho de levar em consideração todos os parâmetros importantes.

As maiores vantagens desta técnica são os benefícios de se obter um modelo formal da aplicação. A indicação de possíveis **gargalos** podem ajudar o programador a prestar mais atenção a tais pontos durante a implementação. Resultados como os apresentados na seção anterior podem claramente identificar o melhor número de tarefas de acordo com o número de processos envolvidos. Tal informação sobre o melhor grão pode ser útil para um escalonador, ou processo distribuidor de tarefas, para automaticamente escolher quantos nós em uma máquina agregada devem ser alocados para uma determinada aplicação.

Apesar dos modelos SAN aqui propostos apresentarem resultados positivos quando comparados com os experimentos escolhidos, é de opinião do autor que há necessidade de uma maior validação dos mesmos. Assim, um trabalho futuro natural é a escolha de aplicações reais principalmente para os modelos de fases paralelas, *pipeline* e divisão e conquista. Pode-se também prever como trabalho futuro, o desenvolvimento de uma ferramenta amigável ao usuário para facilitar a construção de modelos SAN por programadores com pouca, ou nenhuma, habilidade em avaliação de desempenho. Ainda, os modelos SAN aqui propostos podem modelar aplicações não somente para máquinas agregadas, mas para qualquer tipo de arquitetura, desde que suas taxas sejam corretamente parametrizadas. Por exemplo, em uma máquina SMP o tempo de envio de uma informação entre processos é substituído pelo tempo de escrita na memória global. Assim, tem-se com outro trabalho futuro a validação desses modelos em outras arquiteturas.

Uma comparação de aplicações implementadas nos *benchmarks limpack* com os modelos SAN pode trazer resultados interessantes. Como comprovado nas experiências práticas da utilização da modelagem através de SAN apresentadas neste trabalho, o uso de modelos analíticos de Redes de Autômatos Estocásticos para implementações paralelas se mostra uma ferramenta teórica útil e viável em termos de tempo de aprendizado.

Durante o desenvolvimento desta dissertação de mestrado, dois trabalhos sobre este assunto foram submetidos e aceitos na comunidade científica. O primeiro trabalho realiza um estudo sobre o modelo mestre/escravo para a paralelização da ferramenta PEPS [4]. Já o segundo trabalho apresenta estudos iniciais sobre a generalização de modelos SAN para prever desempenho de aplicações paralelas, em [3]. Além disso, alguns trabalhos que não estão relacionados diretamente, mas que foram importantes para a formação acadêmica

do autor, formam elaborados e publicados. Dentre eles, os que mais se destacam podem ser vistos em [5, 6, 7].

Referências Bibliográficas

[1] G. R. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Wesley, University of Arizona, USA, 2000.

[2] C. Anglano. Predicting Parallel Applications Performance on Non-Dedicated Cluster Platforms. In *International Conference on Supercomputing*, pages 172–179, 1998.

[3] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Models for Master/Slave Parallel Programs. *Electronic Notes In Theoretical Computer Science*, 128(4):101–121, April 2005.

[4] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho, and T. Webber. Parallel PEPS Tool Performance Analysis using Stochastic Automata Networks. In M. Donelutto, D. Laforenza, and M. Vanneschi, editors, *Euro-Par 2004 International Conference on Parallel Processing*, volume 3149 of *Lecture Notes in Computer Science*, pages 214–219, Pisa, Italy, August/September 2004. Springer-Verlag Heidelberg.

[5] L. Baldo, L. G. Fernandes, P. Velho, M. Castro, and M. Raeder. A Parallel Version for the Propagation Algorithm. In *PaCT - International Conference on Parallel Computing Technologies*, volume 3606 of *Lecture Notes in Computer Science*, pages 403–412, Kranoyarsk, 2005. Springer-Verlag Heidelberg.

[6] L. Baldo, L. G. Fernandes, P. Velho, M. Kolberg, and D. Claudio. Optimizing a Parallel Self-verified Method for Solving Linear Systems. In *PARA - Workshop on State-of-the-art in Scientific and Parallel Computing*, accepted paper, 2006.

[7] L. Baldo, L. G. Fernandes, P. Velho, M. Kolberg, T. Webber, P. Fernandes, and D. Claudio. Parallel Selfverified Method for Solving Linear Systems. In *VECPAR - International Meeting on High Performance Computing for Computational Science*, Rio de Janeiro, Janeiro 2006. Proceedings of the VECPAR 2006.

[8] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The PEPS Software Tool. In *Computer Performance Evaluation / TOOLS 2003*, volume 2794 of *LNCS*, pages 98–115, Urbana, IL, USA, 2003. Springer-Verlag Heidelberg.

[9] C. Bertolini, L. Brenner, P. Fernandes, A. Sales, and A. F. Zorzo. Structured Stochastic Modeling of Fault-Tolerant Systems. In 12^{th} *IEEE/ACM Internacional Symposium on Modelling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS'04)*, pages 139–146, Volendam, The Netherlands, October 2004. IEEE Press.

[10] Scott A. Burgess. Stochastic Automata Network Models of Agents. In *IC-AI*, pages 358–364, 2004.

[11] P. Dmitruk, L. P. Wang, W. H. Matthaeus, R. Zhang, and D. Seckel. Scalable Parallel FFT for Spectral Simulations on a Beowulf Cluster. *Parallel Computing*, 27(14):1921–1936, 2001.

[12] L. R. C. Fonseca, P. von Allmen, and R. Ramprasad. Numerical Simulation of the Tunneling Current and Ballistic Electron Effects in Field Emission Devices. *Journal of Applied Physics*, 87(5):2533–2541, 2000.

[13] E. Gelenbe and H. Shachnai. On G-Networks and Resource Allocation in Multimedia Systems. In *IEEE Research in Data Engineering*, pages 104–110, Orlando, Florida, February 1998.

[14] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing.* Addison Wesley, 2003.

[15] O. Gusak, T. Dayar, and J.-M. Fourneau. Iterative Disaggregation for a Class of Lumpable Discrete-time Stochastic Automata Networks. *Performance Evaluation*, 53(1):43–69, 2003.

[16] L. Hu and Ian Gorton. Performance Evaluation for Parallel Systems: A Survey. Technical Report 9707, University of NSW, Sydney, Australia, 1997.

[17] W. Meira Jr. Modeling Performance of Parallel Programs. Technical Report 589, The University of Rochester, Rochester, New York, 1995.

[18] W. S. Kendall. Perfect Simulation for the Area-interaction Point Process. In L. Accardi and C.C. Heyde, editors, *Probability Towards 2000*, pages 218–234, Manchester, UK, 1988. Springer Verlag New York.

[19] I. H. Manssour, L. G. Fernandes, C. M. D. S. Freitas, G. Serra, and T. Nunes. A High Performance Approach for Inner Structures Visualization in Medical Data. *International Journal Of Computer Applications In Technology (IJCAT)*, 21(4):23–33, 2005.

[20] R. Marculescu and A. Nandi. Probabilistic Application Modeling for System-Level Performance Analysis. In *Design Automation & Test in Europe (DATE)*, pages 572–579, Munich, Germany, March 2001.

[21] T. G. Mattson, B. A. Sanders, and B. L. Massingill. *Patterns for Parallel Programming*. Addison Wesley, 2004.

[22] L. Mokdad, J. Ben-Othman, and A. Gueroui. Quality of Service of a Rerouting Algorithm Using Stochastic Automata Networks. In 6^{th} *IEEE Symposium on Computers and Communications*, pages 338–343, Hammamet, Tunisia, July 2001. IEEE Computer Society.

[23] PEPS. Performance Evaluation of Parallel Systems. http://www-id.imag.fr/Logiciels/peps/. último acesso: 15/01/2007.

[24] B. Plateau. On the Stochastic Structure of Parallelism and Synchronization Models for Distributed Algorithms. In *Proceedings of the 1985 ACM SIGMETRICS conference on Measurements and Modeling of Computer Systems*, pages 147–154, Austin, Texas, USA, 1985. ACM Press.

[25] B. Plateau and K. Atif. Stochastic Automata Networks for Modelling Parallel Systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[26] T. Sato. The Earth Simulator: Roles and Impacts. *Parallel Computing*, 30(12):1279–1286, 2004.

[27] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[28] Y. Sun, A. Wipat, M. Pocock, P. A. Lee, P. Watson, K. Flanagan, and J. T. Worthington. A Grid-based System for Microbial Genome Comparison and Analysis. In *5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, Cardiff, UK, May 2005.

[29] A. J. C. van Gemund. Symbolic Performance Modeling of Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(2):154–165, 2003.

[30] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Upper Saddle River, New Jersey, 1999.

[31] Yong Yan, Xiaodong Zhang, and Yongsheng Song. An Effective and Practical Performance Prediction Model for Parallel Computing on Nondedicated Heterogeneous NOW. *Journal of Parallel and Distributed Computing*, 38(1):63–80, 1996.