Performance Prediction of Parallel Applications with Parallel Patterns using Stochastic Methods

Mateus Raeder, Dalvan Griebler, Lucas Baldo, Luiz Gustavo Fernandes
Programa de Pós-Graduação em Ciência da Computação
Av. Ipiranga 6681, prédio 32, sala 601 - Porto Alegre, RS - Brasil
{mateus.raeder,dalvan.griebler,lucas.baldo}@acad.pucrs.br, luiz.fernandes@pucrs.br

Abstract

One of the main problems in the high performance computing area is to find the best strategy to parallelize an application. In this context, the use of analytical methods to evaluate the performance behavior before the real implementation of such applications seems to be an interesting alternative and can help to identify better directions for the implementation strategies. In this work, the Stochastic Automata Network (SAN) formalism is adopted to model and evaluate the performance of parallel applications. The methodology used is based on the construction of generic SAN models to describe classical parallel programming patterns, like Master/Slave, Pipeline and Divide and Conquer. Those models are adapted to represent cases of a real application through the definition of input parameters values. Finally, we present a comparison between the results of the SAN models and a real application, aiming at verifying the accuracy of the adopted technique.

1 Introdução

No cenário da Computação de Alto Desempenho, quando aplicações apresentam altos tempos de execução, a condução de experimentos repetitivos para avaliar seus resultados torna-se uma tarefa custosa e muitas vezes inviável, dificultando a avaliação de desempenho em sistemas reais. Uma alternativa atraente para este problema trata-se da predição de desempenho, com a qual se consegue obter informações sobre a execução da aplicação (tempo de execução, por exemplo) antes mesmo que a aplicação seja implementada. Desta forma, predizer o desempenho de programas paralelos surge como uma possibilidade de aperfeiçoar o processo de desenvolvimento, pois o programador obtém uma estimativa do tempo de execução da aplicação com menos custo e evitando recodificações desnecessárias.

Para prever o desempenho de aplicações paralelas sem a necessidade de implementá-las, comumente realizam-se modelagens através do uso de algum formalismo que descreve o comportamento geral do sistema. Da resolução do modelo criado são efetuadas determinadas análises, que permitem prever o comportamento de uma dada implementação em diversos aspectos, variando desde probabilidades de transmissão e porcentagem de utilização dos nós no *cluster* até estimativas de tempo de execução. Neste estudo utilizou-se Modelagem Analítica, que se refere uma representação matemática do sistema. Neste tipo de modelagem as métricas de desempenho são obtidas através de modelos construídos com certos parâmetros que vêm da aplicação.

O formalismo *Stochastic Automata Network* (SAN) [1] tem sido utilizado para descrever estruturas interdependentes complexas [2, 3, 4, 5]. Baseado na teoria de Cadeias de Markov [6], SAN possui a vantagem de expressar modelos markovianos de maneira mais intuitiva e compacta. Através da modelagem com SAN, medidas de desempenho podem ser incluídas, tais como *throughput*, atraso de sincronização, tempo de resposta, dentre outras, mesmo antes da implementação da aplicação [7, 8].

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. Neste contexto, este trabalho tem por objetivo gerar modelos SAN genéricos para os padrões de programação paralela mais utilizados na área de Processamento de Alto Desempenho. Os modelos genéricos criados são a contribuição principal do trabalho, pois são ferramentas úteis para o programador predizer o desempenho, evitando a construção de um novo modelo para cada aplicação, bastando a adaptação do modelo para a arquitetura e para a aplicação desejada.

O restante do documento está organizado da seguinte maneira: a Seção 2 descreve os padrões de programação paralela utilizados na modelagem; a Seção 3 apresenta os modelos SAN genéricos para os padrões de programação escolhidos; a Seção 4 relata como foram conduzidos os experimentos; a Seção 5 descreve uma análise comparativa dos resultados obtidos; a Seção 6 elenca alguns trabalhos relacionados ao presente estudo; e, finalmente, a Seção 7

conclui o trabalho com algumas considerações finais.

2 Padrões de Programação Paralela

Padrões de programação paralela objetivam auxiliar o programador na exploração de paralelismo para melhor aproveitar os recursos computacionais, provendo aplicações de alto desempenho. Para tanto, devem ser levados em conta diversos aspectos, como número de nodos de processamento disponíveis, complexidade da implementação sequencial, interdependência de dados, dentre outros fatores que envolvem este processo. Os padrões mais utilizados para estruturar códigos paralelos são: Mestre/Escravo, *Pipeline* e Divisão e Conquista.

2.1 Mestre/Escravo

O padrão 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 padrão, o processo mestre pode utilizar uma técnica de balanceamento de carga para que todos os processos tenham tarefas de pesos semelhantes, evitando que processos escravos fiquem sobrecarregados (*i.e.*, processos ociosos). A utilização do padrão Mestre/Escravo deve ser realizada de forma que o processo mestre (centralizador) não se torne um gargalo, podendo resultar em perda de desempenho [9].

2.2 *Pipeline*

O padrão *Pipeline* consiste de uma computação baseada em estágios, visto também como uma sequência de dados através de uma sequência de estágios [10]. Embora o fluxo seja contínuo e o problema seja executado sequencialmente, o paralelismo é obtido através da execução de mais de um estágio. Basicamente, cada estágio recebe uma entrada, processa e transmite o resultado para o próximo estágio, até que a computação seja concluída. Na estruturação da computação, pode-se usar o ID (identificador) de cada processo para definir o comportamento em um determinado caso, o qual corresponde a um estágio do *Pipeline* (processar o resultado final da computação, por exemplo).

2.3 Divisão e Conquista

O padrão de Divisão e Conquista é uma estratégia utilizada em vários algoritmos sequenciais [9]. Consiste em dividir o problema em subproblemas menores, resolvendo-os independentemente e fundindo todas as subsoluções em uma solução para o problema [11]. Neste padrão, o primeiro passo é dividir o problema em subproblemas. Assim sendo, o resultado desta modelagem será um grafo de tarefas independentes. Depois da divisão, ocorre a conquista: os resultados retornados do nível abaixo são fundidos e, logo em seguida, devolvidos a um nível acima do grafo. Desta forma, as tarefas filhas retornam o resultado para a tarefa pai, logo após, estas tarefas devolverão o seu problema resolvido para a raiz, que obterá a solução final.

3 Modelos SAN Genéricos

Os modelos SAN apresentados nesta seção podem ser utilizados de acordo com o padrão de programação escolhido, sendo necessárias apenas alterações nos seus parâmetros, que devem estar de acordo com o ambiente de execução e com a aplicação a ser modelada. Tais modelos auxiliam o programador a verificar a viabilidade da sua aplicação antes de estar implementada. Para um maior detalhamento do formalismo SAN, sugere-se a leitura de [1].

3.1 Considerações Gerais Sobre a Parametrização

Antes do detalhamento da modelagem, os estados finais e iniciais dos modelos SAN são descritos, assim como a maneira na qual os valores numéricos (taxas) de cada modelo são calculados. Alguns destes valores são dados pelos desenvolvedores (valores de entrada do modelo), enquanto outros são avaliados utilizando estes valores de entrada.

Inicialmente, algumas variáveis são criadas para auxiliar a definição dos parâmetros:

- **CS**: tempo de comunicação de dados entre dois processos;
- **OS**: tempo de processamento de uma tarefa por um processo.

Os tempos CS e OS 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. Nos modelos criados 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.

Nas próximas seções os modelos genéricos para cada padrão são apresentados, juntamente com suas taxas e seus estados iniciais e finais.

3.2 Mestre Escravo

Nesta seção, os modelos SAN genéricos para o padrão de programação Mestre/Escravo são apresentados. Além da definição do modelo SAN, a parametrização é discutida, definido as taxas e os estados inicial e final do modelo SAN genérico.

3.2.1 Modelo SAN Genérico. O modelo SAN para o padrão Mestre/Escravo é ilustrado na Figura 1. Este

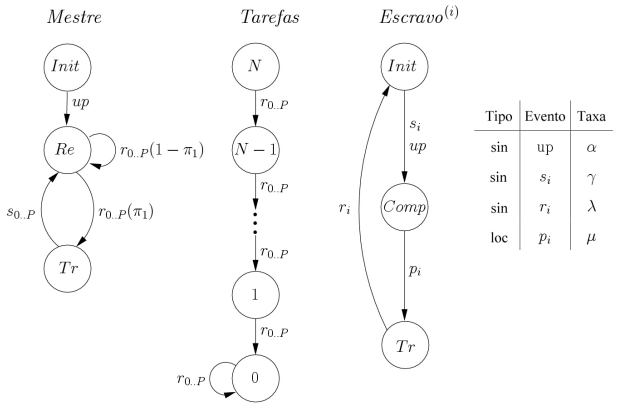

Figura 1. SAN genérico: Mestre/Escravo

modelo contém um autômato Mestre, um autômato Tarefas, e P autômatos Escravo$^{(i)}$ *(i = 1..P)*.

Na modelagem proposta (Figura 1), 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 os 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. Pela ocorrência do evento sincronizante s_i, o nó Mestre envia uma nova tarefa para o i-ésimo Escravo. De maneira similar, a recepção dos resultados 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 (caso não existam mais tarefas a serem processadas, o que é 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, onde 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 decrementa 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 não 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.

3.2.2 Parametrização. Para completar a modelagem SAN do padrão Mestre/Escravo, as definições das taxas de cada evento e os estados inicial e final devem ser definidos. Os eventos s_i, p_i e r_i têm suas taxas (γ, μ e λ respectivamente) definidas numericamente por (Equação 1):

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

Para este padrão, a taxa α do evento *up* é numericamente definida considerando o envio das tarefas iniciais para todos os P Escravos (Equação 2):

$$\alpha = \frac{1}{P \times CS} \tag{2}$$

A probabilidade de ainda haver tarefas a serem distribuídas para os Escravos 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.

O modelo SAN que representa o padrão 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. O autômato Tarefas inicia com todas as N tarefas a serem processadas. Assim, existe somente um estado inicial possível para este modelo: autômatos Mestre e Escravo^i no estado Init e o autômato Tarefas no estado N.

O único estado final possível para o modelo Mestre/Escravo indica os nós Escravos retornando aos seus estados locais originais com todas as tarefas processadas (Tarefas no estado 0) e cada nó escravo não tendo mais nenhuma tarefa para enviar (ou seja, a quantidade de autômatos do tipo Escravo^i no estado Init é igual a P).

3.3 *Pipeline*

Na sequência, a modelagem genérica para o padrão de programação *Pipeline* é definida, assim como os parâmetros que completam o modelo.

3.3.1 Modelo SAN Genérico. O modelo SAN para aplicações que utilizem o padrão *Pipeline* é mostrado na Figura 2. Este modelo específico contém três autômatos representando os processos *1*, *i* e o número de tarefas a serem processadas. Novamente, o número de processos a serem utilizados pode variar, o que acarretará na alteração do número de autômatos no modelo.

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 recebe tarefas de seu processo

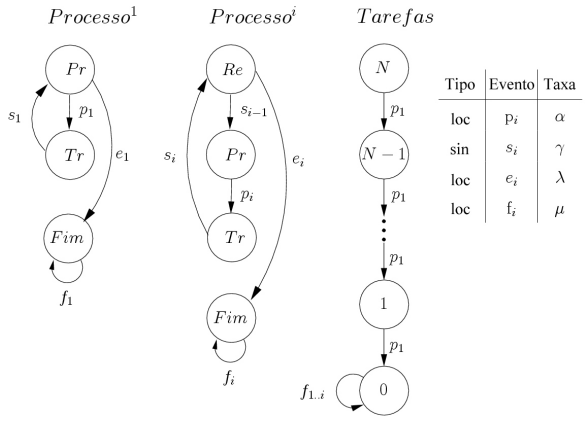

Figura 2. SAN genérico: *Pipeline*

antecessor. Esta característica ocorre pois o padrão de programação *Pipeline* determina que um processo somente inicia a 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 próximo processo. O evento s_1 caracteriza o fim do envio do Processo^1 e o inicio do processamento do processo seguinte. 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. Todos os autômatos deste modelo 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 contabilizar o número de tarefas já processadas. Este autômato define o critério de parada do modelo SAN.

3.3.2 Parametrização. O modelo *Pipeline* também possui quatro taxas, apresentadas na tabela da Figura 2. Os eventos locais $p_{1..P}$ possuem o valor da taxa α do modelo Mestre/Escravo. Já a taxa λ é diferente, e pode ser calculada da seguinte forma (Equação 3):

$$\lambda = \frac{1}{CS} \tag{3}$$

Onde CS, neste caso, é o tempo de transmissão de uma tarefa de um processo para outro. Por outro lado, a taxa γ e a taxa μ são taxas funcionais onde todos os processos devem estar no estado Fim.

O estado inicial do modelo SAN do padrão *Pipeline* ocorre quando o Processo^1 encontra-se no estado Pr, os demais autômatos encontram-se no estado Re e o autômato Tasks está em N. Já para o estado final, o autômato Tasks deve estar no estado 0 (todas as tarefas concluídas) e todos os demais autômatos devem estar no estado Fim.

3.4 Divisão e Conquista

O último padrão de programação paralela modelado trata-se do Divisão e Conquista. Seguindo a mesma linha dos modelos anteriores, o modelo SAN genérico e a definição dos parâmetros serão apresentados a seguir.

3.4.1 Modelo SAN Genérico. O modelo SAN para Divisão e Conquista é mostrado na Figura 3. 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 padrão de programação é o que possui a modelagem SAN mais complexa. Assim, para que processos sejam acrescentados 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. Este modelo 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 subtarefas, envia uma subtarefa para um processo filho e fica com uma subtarefa para processar. Se ainda houver processos que não possuem uma tarefa, o processo Raiz divide novamente a sua subtarefa em duas subsubtarefas, enviando uma delas para outro processo filho. Esta divisão é realizada até que todos os processos obtenham uma tarefa para processar. Nota-se, assim, que processos filhos do Raiz podem ser tanto processos intermediários quanto processos folhas.

Os processos intermediários, representados pelo autômato Intermediário^i recebem uma tarefa de seu processo pai, quebram-na em subtarefas e enviam uma delas para seus filhos. Já os processos folhas, representados pelo autômato Folha, já não possuem processos filhos. Estes recebem uma tarefa, processam-na e enviam a resposta para seu processo pai. Utilizou-se uma generalização quanto ao número de envios e recebimentos de tarefas nos autômatos Raiz e Intermediário^i. Entretanto, 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, por sua vez, ocorre quando um processamento de determinada tarefa é finalizado. Por fim, a ocorrência do evento f_i é necessária

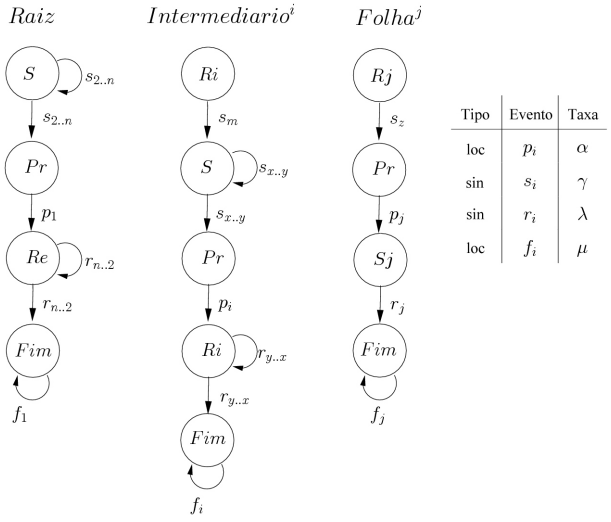

Figura 3. SAN genérico: Divisão e Conquista

para que o modelo torne-se cíclico, igualmente aos outros modelos SAN apresentados até agora.

3.4.2 Parametrização. O modelo Divisão e Conquista tem a taxa α assim como descrita nos demais 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, para este estudo de caso, é a mesma que a taxa γ. Isto ocorre porque o número de dados a serem retornados são os mesmos que são 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.

O estado inicial para o modelo SAN do padrão Divisão e Conquista acontece quando: (i) o autômato Raiz está no estado S; (ii) os autômatos Intermediários estão no estado Ri; e (iii) os autômatos folha estão no estado Rj. Quando todos os autômatos encontrarem-se no estado Fim, o modelo atinge seu estado final.

4 Descrição dos Experimentos

Com o intuito de comparar os tempos de execução obtidos com a utilização da modelagem SAN com resultados reais, foi utilizada uma simples aplicação de **multiplicação de matrizes**. Esta aplicação foi escolhida por ser de fácil implementação e poder ser encaixada em todos os padrões de programação paralela listados na Seção 2.

Primeiramente, foi definida a plataforma a ser utilizada, pois esta escolha influencia diretamente os parâmetros da modelagem realizada. A plataforma utilizada para os experimentos foi um *cluster*, no qual as aplicações utilizam troca de mensagens para realizar a comunicação entre os processos, uma vez que a memória não é compartilhada entre os processadores. Assim, os cálculos para obtenção das taxas dos modelos devem levar em conta este aspecto, utilizando bibliotecas e primitivas de comunicação através de troca de mensagens para medir os custos de comunicação. A escolha por um ambiente de máquinas agregadas (*cluster*) deu-se pelo caráter inovador e ao baixo custo da ideia, que interconecta diversos computadores de uso pessoal através de uma rede de alta velocidade.

Em seguida, a aplicação alvo (multiplicação de matrizes) foi modelada com o formalismo SAN nos três padrões de programação apresentados, ou seja, os três modelos genéricos foram adaptados para representar a aplicação e suas características. Para executar os modelos, *i.e.*, extrair as informações desejadas, foi utilizada a ferramenta PEPS [12]. Com esta ferramenta, tanto probabilidades de ocorrência de determinados estados ou conjuntos de estados, quanto estimativas de tempos de execução da aplicação modelada podem ser obtidos. Neste último caso, deve ser realizada uma análise transiente (também realizada no PEPS).

Obtidos os tempos de execução, partiu-se para a implementação da aplicação paralela, nos mesmos moldes do que foi modelado. Três implementações diferentes de multiplicação de matrizes foram realizadas (uma para cada padrão paralelo), e os tempos de execução foram obtidos.

Nos experimentos, os modelos foram resolvidos com diferentes conjuntos de parâmetros de entrada. Três diferentes grãos (número de tarefas) foram considerados: 10.000 (pequeno), 15.000 (médio) e 20.000 (grande). O número de processos variou entre 2 e 7 para o padrão Mestre/Escravo e entre 2 e 6 para os padrões *Pipeline* e Divisão e Conquista. Estes números (de tarefas e de processos) foram escolhidos devido a um grande aumento no espaço de estados da modelagem SAN quando da adição de novos estados nos autômatos, o que torna a resolução dos modelos mais lenta e até mesmo inviável em determinadas ocasiões.

5 Resultados Obtidos

Neste capítulo, os resultados de tempo de execução estimados através dos modelos SAN são comparados com tempos obtidos através de implementações reais da aplicação de multiplicação de matrizes, utilizando os três padrões de programação descritos anteriormente. A plataforma distribuída escolhida para a realização dos experimentos foi um *cluster* com nós Itanium-2 de 64 bits, biprocessados, com 3 GB RAM e interconectados por uma rede Myrinet. Cada tempo de execução das implementações paralelas foi obtido através de uma média de dez execuções, das quais foram retirados os valores extremos.

5.1 Padrão Mestre/Escravo

A Tabela 1 mostra os tempos de execução do modelo SAN Mestre/Escravo e da execução real da aplicação de

multiplicação de matrizes, com diferentes números de escravos (variando de 2 a 6).

Tabela 1. Resultados do padrão Mestre/Escravo

10.000 tarefas					
Número de 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
15.000 tarefas					
Número de 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
20.000 tarefas					
Número de 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

Nesta Tabela 1 é possível observar que para todas as configurações da implementação paralela o modelo SAN e os resultados experimentais apresentam valores bem próximos. Analisando os valores obtidos, constatou-se que o desempenho da aplicação cresce conforme o número de escravos aumenta, indicando um provável aumento de desempenho com mais de seis escravos.

Mesmo com variações no número de tarefas, a modelagem SAN continuou adequando-se a realidade. Com 10.000 tarefas, a maior diferença nos tempos entre o modelo e a execução real foi de 0,3889 segundos (com 5 escravos). Na execução com um grande número de tarefas (20.000), a maior diferença entre os valores é menor do que 2,5 segundos.

Os resultados obtidos para o padrão Mestre/Escravo mostram que este padrão de programação apresentou-se eficiente, pois a modelagem SAN conseguiu adequar-se ao ambiente de execução, prevendo os resultados da execução da aplicação com valores muito próximos dos reais.

5.2 Padrão *Pipeline*

Na Tabela 2 observam-se os tempos de execução para o padrão de programação paralela, o *Pipeline*.

Tabela 2. Resultados do padrão *Pipeline*

10.000 tarefas					
Número de processos	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
15.000 tarefas					
Número de processos	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
20.000 tarefas					
Número de processos	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

Nota-se que os tempos de execução para este caso de teste aumentam de acordo com o aumento do número de processos. Em outras palavras, não houve um ganho de desempenho para a multiplicação de matrizes utilizando o padrão de programação *Pipeline*, que apresentou-se como inapropriado para este tipo de aplicação. Contudo, de acordo com os tempos obtidos e apresentados, percebe-se que o modelo SAN conseguiu prever esta situação, identificando que com a escolha deste padrão não haveria ganho algum de desempenho. De acordo com os diferentes números de tarefas, observa-se que em nenhum momento os modelos SAN apresentaram incoerência com os resultados coletados na execução da aplicação paralela.

5.3 Padrão Divisão e Conquista

O último caso de teste trata-se do padrão de programação paralela Divisão e Conquista. A Tabela 3 contém os valores obtidos na execução do modelo SAN e na execução real da aplicação de multiplicação de matrizes para este caso.

Tabela 3. Resultados do padrão Divisão e Conquista

10.000 tarefas					
Número de processos	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
15.000 tarefas					
Número de processos	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
20.000 tarefas					
Número de processos	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

Mais uma vez, a modelagam SAN conseguiu atingir o objetivo proposto, conseguindo uma coerência nos resultados, até mesmo nos pontos de inflexão citados anteriormente. Por fim, a modelagem SAN mostrou que o padrão Divisão e Conquista apresentaria problemas de balanceamento e resultados discrepantes, o que auxilia o programador no momento da escolha por este padrão.

6 Trabalhos Relacionados

A utilização de Modelagem Analítica pode ser constantemente percebida em trabalhos científicos em diversas ocasiões. Em [13] os autores utilizam a técnica de Modelagem Analítica para avaliar o protocolo MAC IEEE 802.16, propondo um modelo que possibilita avaliar o desempenho do protocolo em questão de acordo com o atraso total de mensagens. O modelo analítico proposto utiliza Teoria de Filas e Cadeias de Markov para representar o comportamento do padrão IEEE 802.16. Diferentes análises são realizadas, e os resultados foram validados através de comparações com simulações no NS-2 (*Network Simulator*). Os autores mostram que a utilização de Modelagem Analítica é uma técnica importante no processo de avaliação de desempenho.

Em [14] são descritos alguns Modelos Analíticos para o padrão *Pipeline* baseado em Teoria de Filas. Os mode-

los criados permitem determinar parâmetros que, manualmente, exigem uma série de execuções com diferentes configurações para serem plenamente entendidos. Os resultados obtidos através dos modelos foram satisfatórios e se aproximaram do comportamento real.

A utilização de SAN tem surgido como uma ferramenta importante para a modelagem. Os autores de [15] utilizaram SAN para modelar e analisar o padrão de mobilidade *Random Waypoint*, um dos mais conhecidos em redes *wireless*. A modelagem realizada objetiva analisar a distribuição de nós da rede, melhorando a predição de desempenho. Foi demonstrado que a modelagem com SAN pode ser utilizada para padrões de mobilidade, validando os resultados através de comparações com outros modelos existentes.

Os autores de [16] utilizaram Modelagem Analítica para modelar o escalonamento do sistema operacional, descrevendo o comportamento dos processos e processadores em uma determinada migração. O objetivo principal é prever se uma mudança no algoritmo de escalonamento deve ou não ser implementada, analisando os resultados do modelo desenvolvido. Os autores utilizaram SAN para a modelagem, afirmando que trata-se de um formalismo interessante quando existem várias atividades sendo executadas em paralelo, além de suportar modelos com uma larga quantidade de estados. Foi modelado um algoritmo de escalonamento do sistema operacional Linux, em uma máquina NUMA (*No-uniform Memory Access*).

No trabalho descrito em [17], os autores propuseram o uso de SAN para desenvolver modelos que se aplicassem a programas do tipo Mestre/Escravo, considerando dois padrões de comportamento para descrever a comunicação: síncrono e assíncrono. Como caso de estudo, um algoritmo conhecido como *Propagation* foi utilizado. Esta aplicação trabalha com interpolação de imagens para gerar uma visão virtual entre dois pontos iniciais distintos. A aplicação foi modelada utilizando SAN para validar a estratégia de implementação escolhida. Segundo os autores, a modelagem de programas paralelos é facilitada com o formalismo SAN, e pode dar uma visão geral dos resultados que serão obtidos com a paralelização do *Propagation*.

Em todos os trabalhos relatados, os autores utilizam a Modelagem Analítica como uma ferramenta extremamente importante para a avaliação do desempenho de seus experimentos. Nos trabalhos supracitados que utilizaram a modelagem com SAN, os resultados foram promissores e motivadores para a sua utilização neste trabalho. No entanto, os trabalhos que envolvem SAN modelam aplicações e soluções específicas, dentro de algum contexto, o que difere completamente da abordagem deste estudo. Neste trabalho, a modelagem realizada com o formalismo SAN não é restrita a um determinado uso, tratando-se de uma abordagem genérica para qualquer aplicação paralela que seja implementada através dos padrões de programação modelados.

7 Conclusão

Neste trabalho foram apresentados modelos SAN genéricos para diferentes padrões de programação paralelas. Para a área do Processamento de Alto Desempenho, o uso de formalismos estocásticos é frequentemente limitado pelo tamanho do problema (em número de estados). Acredita-se que a modelagem com SAN é uma boa alternativa para contornar este problema e o estudo apresentado neste trabalho, modelando aplicações para ambientes de alto desempenho, reforça a facilidade da utilização deste formalismo para prever o desempenho de aplicações em diferentes áreas.

Três modelos SAN genéricos foram apresentados neste trabalho, cada um representando um padrão de programação diferente: Mestre/Escravo, *Pipeline* e Divisão e Conquista. A aplicação de multiplicação de matrizes foi utilizada como estudo de caso para verificar a adaptabilidade do formalismo SAN para modelar cada um dos padrões. Em todos os experimentos obteve-se resultados que confirmam que a modelagem realizada com o formalismo SAN consegue aproximar o comportamento da aplicação, obtendo tempos bem próximos dos reais. A Tabela 4 confirma este fato, apresentando os erros máximos para cada modelo SAN em relação aos tempos obtidos com a aplicação utilizada.

Tabela 4. Erros máximos

Mestre/Escravo	*Pipeline*	Divisão e Conquista
4.7162%	1.3153%	8.9059%

A maior dificuldade desta abordagem está ligada a importância atribuída à fase de parametrização, principalmente no que diz respeito à 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. Assim sendo, a técnica de modelagem auxilia o programador informando quais taxas devem ser informadas. O programador, então, deve levar em consideração todos os parâmetros da aplicação e do ambiente de execução.

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 resolver determinados problemas da aplicação, antes mesmo da implementação. Resultados como os apresentados na Seção 5 podem claramente identificar o melhor número de tarefas de acordo com o número de processos envolvidos. Tal informação pode ser útil para um escalonador, ou processo distribuidor de tarefas que pode automaticamente escolher quantos nós de processamento do ambiente modelado devem ser alocados para uma determinada aplicação.

Apesar dos modelos SAN aqui propostos apresentarem resultados positivos quando comparados com as execuções reais, alguns trabalhos futuros ainda são necessários. Um

destes trata-se da escolha de novas aplicações reais assim como o desenvolvimento de uma ferramenta amigável ao usuário para facilitar a construção de modelos SAN por programadores com pouco (ou nenhum) conhecimento da abordagem, uma vez que SAN apresentou-se como uma ferramenta muito útil em termos de predição de desempenho.

Referências

[1] B. Plateau and K. Atif. Stochastic Automata Network of Modeling Parallel Systems. *Software Engineering, IEEE Transactions on*, 17(10):1093 –1108, October, 1991.

[2] 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.

[3] O. Gusak, T. Dayar, and J.-M. Fourneau. Iterative disaggregation for a class of lumpable discrete-time stochastic automata networks. *Perform. Eval.*, 53:43–69, June, 2003.

[4] 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.

[5] 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.

[6] G. Bolch, S. Greiner, H. Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA, 1998.

[7] L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Analysis Issues for Parallel Implementations of Propagation Algorithm. In *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD '03, pages 183–190, Washington, DC, USA, 2003. IEEE Computer Society.

[8] L. Baldo, L. G. Fernandes, P. Roisenberg, P. Velho, and T. Webber. Parallel PEPS Tool Performance Analysis Using Stochastic Automata Networks. In *Euro-Par '04*, pages 214–219, Pisa, Italy, 2004. Springer.

[9] T. G. Mattson, B. A. Sanders, and B. L. Massingill. *Patterns for Parallel Programming*. Addison-Wesley, Boston, MA, 2005.

[10] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Pearson (Addison-Wesley), Boston, MA, 2003.

[11] M. J. Quinn. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, New York, 2004.

[12] A. Benoit, L. Brenner, P. Fernandes, B. Plateau, and W. J. Stewart. The peps software tool. In P. Kemper and W.H. Sanders, editors, *13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation TOOLS 2003*, pages 98–115, Urbana, Illinois, USA, 2003.

[13] L. F. M. de Moraes and D.L.F.G. Vieira. Analytical Modelling and Message Delay Performance Evaluation of the IEEE 802.16 MAC Protocol. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '10, pages 182–190, Washington, DC, USA, 2010. IEEE Computer Society.

[14] A. Navarro, R. Asenjo, S. Tabik, and C. Cascaval. Analytical Modeling of Pipeline Parallelism. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 281–290, Washington, DC, USA, 2009. IEEE Computer Society.

[15] F. Delamare, F. L. Dotti, P. Fernandes, C. M. Nunes, and L. C. Ost. Analytical Modeling of Random WayPoint Mobility Patterns. In *Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, PE-WASUN '06, pages 106–113, New York, NY, USA, 2006. ACM.

[16] R. Chanin, M. Corrêa, P. Fernandes, A. Sales, R. Scheer, and A. F. Zorzo. Analytical Modeling for Operating System Schedulers on NUMA Systems. *Electron. Notes Theor. Comput. Sci.*, 151:131–149, June, 2006.

[17] L. Baldo, L. Brenner, L. G. Fernandes, P. Fernandes, and A. Sales. Performance Models For Master/Slave Parallel Programs. *Electron. Notes Theor. Comput. Sci.*, 128:101–121, April, 2005.