Simulação Paralela de Redes de Petri

Adriana de Andrade Oliveira Hao Chi Wong
Wagner Meira Júnior Virgílio A. F. Almeida*

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
31270-010 Belo Horizonte, MG, Brasil
e-mail: virgilio@dcc.ufmg.br

Resumo

As Redes de Petri (RP) vêm sendo cada vez mais utilizadas para modelar modernos sistemas de computação, devido à sua flexibilidade na representação de características como paralelismo, bloqueios e sincronização de processos. As RP estocásticas (RPE) introduzem a noção de tempo às RP convencionais, tornando-se adequadas para a análise de medidas de desempenho dos sistemas modelados.

Devido a problemas combinatoriais de explosão do espaço de estados do modelo, a técnica de solução utilizada é a simulação. Com o objetivo de reduzir o tempo de computação de uma solução de um modelo de RP, este trabalho apresenta a possibilidade de se usar vários processadores paralelos na execução de um programa de simulação de RP.

Abstract

Petri nets (PN) have been widely used to model novel computer systems. They can naturally represent important features of these systems, such as: concurrency, synchronization and blocking.

Stochastic Petri Nets (SPN) add the notion of time to the original Petri net model. SPN is an appropriate model to represent performance of computer systems. However, in some cases, due to the combinatorial explosion of the state space of the model, an analytic solution of the Petri net becomes unfeasible. Thus, simulation appears as an alternative solution to solve large Petri net models. This article presents a way of using many processors in parallel to obtain a solution to RP models.

*Parcialmente financiado pela FAPEMIG (contrato TEC-1113/90)

1 Introdução

Desde que foram propostas em 1962 por C.A. Petri [Petr62], redes de Petri (RP) vêm sendo usadas como uma ferramenta poderosa para modelagem e análise de sistemas reais. O grande poder expressivo, a simplicidade de seus conceitos básicos e a flexibilidade com que podem ser usadas para modelagem são seus principais atrativos. Em computação, os conceitos de concorrência e sincronização nelas embutidos são fundamentais, principalmente a partir da última década, quando começaram a surgir sistemas de processamento paralelo.

Tal como foram definidos inicialmente, modelos de RP não se aplicam a estudos quantitativos, como, por exemplo, os problemas de análise de desempenho de sistemas de computação. Para esses problemas, a noção de tempo é fundamental. Por essas e outras necessidades, várias extensões de RP foram propostas, sendo as RP estocásticas uma dessas extensões. RP estocásticas são redes que associam uma variável aleatória exponencialmente distribuída a cada transição, para representar o tempo de disparo da transição. Esta extensão foi inicialmente proposta por Molloy [Moll81].

Quanto à sua análise, RP estocásticas podem ser resolvidas analiticamente ou por simulação. Neste trabalho, apenas esta última técnica é enfocada. Simulação é uma técnica através da qual uma rede é avaliada quantitativamente, obtendo-se, ao fim do processo, estimativas das soluções exatas do problema. Apesar de não ser exata, a simulação permite obter resultados para casos onde a solução analítica é intratável computacionalmente, ou seja, sistemas onde o espaço de estados é muito grande. Entretanto, o uso de simulação também apresenta dificuldades. Seu principal problema reside na grande demanda por tempo de computação.

Com o objetivo de reduzir o tempo de computação de uma solução de um modelo de RP, este trabalho apresenta a possibilidade de se usar vários processadores paralelos na execução de um programa de simulação de RP. A implementação de um simulador paralelo em um sistema composto por vários processadores é também apresentado no artigo.

2 Objetivos

Este artigo apresenta os principais aspectos do desenvolvimento de uma simulação paralela de RP. Mais especificamente, este trabalho aborda os seguintes aspectos:

- **Técnica de simulação paralela de RP**
  Diversas são as estratégias de paralelizar um programa de simulação. Dentro deste objetivo específico, vamos focalizar a simulação paralela conservadora. Estaremos preocupados tanto com os aspectos gerais de uma simulação paralela conservadora qualquer, quanto com os aspectos particulares de uma simulação paralela conservadora de RP. Ao passar de uma simulação sequencial para uma paralela, vários cuidados devem ser tomados. Por exemplo, em vez de um único relógio de simulação controlando toda a execução do programa, teremos agora vários relógios, um para cada processo, exercendo controles locais.
- **Utilização do protocolo TFP (Protocolo de Disparo de Transição)**
  TFP é um protocolo de simulação paralela conservadora de RP que será usado na nossa simulação. Num primeiro momento, ele será implementado tal como foi definido. Entretanto, o protocolo é implementado para sistemas distribuídos ao invés da proposta original para multiprocessadores de memória compartilhada. Uma vez funcionando o protótipo, faremos algumas otimizações neste protocolo, de forma a melhorar o desempenho do programa de simulação. As otimizações serão feitas levando-se em consideração a topologia da rede.

O artigo está organizado da seguinte forma: Seção 3 apresenta uma breve introdução às RP básicas e descreve as RP estocásticas (RPE). Seção 4 apresenta os princípios básicos da simulação paralela conservadora

e as particularidades da simulação paralela conservadora de RP. Na seção 5, são apresentados o protocolo TFP de simulação paralela conservadora para RP e a nossa proposta de otimização.

Seção 6 refere-se à simulação em si. Serão apresentados o ambiente - hardware e software - de computação distribuída em que o trabalho foi desenvolvido, o mapeamento da RP em uma rede de processos físicos, a estrutura geral do programa, os testes realizados e os resultados numéricos obtidos.

Seção 7 apresenta uma avaliação do trabalho e discute a viabilidade de paralelizar simulações de RP num ambiente distribuído.

3 Redes de Petri

3.1 Estrutura de uma Rede de Petri

Uma RP é um grafo bipartido direcionado. Seus nodos podem ser de dois tipos: lugares e transições. Graficamente, os lugares são representados por círculos e as transições por barras. Os arcos se dirigem de um lugar a uma transição ou de uma transição a um lugar.

Um arco (p,t) é um arco que se dirige de um lugar p para uma transição t. Dizemos que p é um lugar de entrada de t e que t é uma transição de saída de p. Analogamente, (t,p) é um arco que se dirige de uma transição t para um lugar p. Neste caso, t é uma transição de entrada de p e p é um lugar de saída de t.

Uma Rede de Petri (RP) pode ser formalmente definida por uma tripla

$$RP = (P,T,A) \tag{1}$$

onde:

$$P = \{p_1, p_2, ..., p_n\}, T = \{t_1, t_2, ..., t_m\}, A_i \subset (P \times T), A_0 \subset (T \times P) A = (A_i \cup A_0)$$

3.2 Marcação

No interior de cada círculo da representação gráfica de uma RP, podem ser encontrados zero ou mais pontos pretos. São os *tokens*. O número de tokens num lugar dá origem à marcação do lugar. A marcação da rede é um vetor de inteiros, sendo que cada inteiro representa a marcação de um lugar na rede. A marcação pode ser interpretada como o estado da rede.

Uma definição formal de RP com marcação é dada por PN $= (P,T,A,M_0)$ onde P, T e A já foram definidos na subseção 3.1 e $M_0 = \{m_{01}, m_{02}, ..., m_{0n}\}$

3.3 Execução de RP

A execução de uma RP é dada pelos disparos das suas transições e controlada pela quantidade e distribuição de tokens nos seus lugares. Uma transição só pode disparar quando está habilitada. Isto acontece quando cada um dos seus lugares de entrada possuir, pelo menos, um token. Ao disparar, uma transição remove um token de cada um dos seus lugares de entrada e deposita um token em cada um dos seus lugares de saída. Assim, os disparos das transições levam a mudanças de marcação. O comportamento dinâmico de um sistema de computação pode ser naturalmente representado pelo movimento dos tokens em uma RP.

3.4 Lugares de Decisão

Muitas vezes, disparos de duas transições podem entrar em conflito, ou seja, o disparo de uma desabilita o da outra. Essa situação de conflito é originada pela existência de um lugar de decisão, que são lugares que possuem duas ou mais transições de saída.

3.5 Redes de Petri Estocásticas (RPE)

Como foi visto na definição original de RP, tempo é uma grandeza ausente na RP básica. Isto significa que o disparo de uma transição é instantâneo e pode ocorrer num tempo arbitrário, uma vez habilitada a transição.

A associação do tempo às redes básicas pode ser feita de várias formas. Em particular, pode-se associar o tempo às transições ou aos lugares. No que se segue, apenas a primeira abordagem será considerada.

Dependendo da natureza dos tempos associados, as RP se classificam em temporizadas ou estocásticas. RP temporizadas são aquelas em que os tempos associados são determinísticos. Quando os tempos são não determinísticos, as RP são chamadas de estocásticas. Redes de Petri Estocásticas (RPE) foram inicialmente propostas por Molloy.

Para caracterizar RPE, é suficiente acrescentar á definição básica da rede um conjunto finito de variáveis aleatórias, que representam taxas médias de disparo das transições. RPE com tempos exponenciais têm a vantagem de apresentar a propriedade Markoviana (*propriedade da falta de memória*).

4 Simulação Paralela Conservadora de Redes de Petri

Nesta seção, vamos apresentar os princípios básicos de uma simulação paralela conservadora e ressaltar as particularidades de uma simulação paralela conservadora de RP.

4.1 Simulação Paralela Conservadora

Neste contexto, simulação é um processo computacional que tem por fim avaliar numericamente o modelo, obtendo estimativas das soluções exatas dos problemas. No que se segue, apenas simulações discretas serão tratadas.

Como os programas de simulação requerem usualmente um grande tempo de processamento, pesquisas têm sido feitas no sentido de utilizar processadores paralelos na aceleração de sua execução. Desses estudos surgiram várias propostas de paralelização. Um levantamento bibliográfico a respeito dessas propostas pode ser encontrado em [Kaudel87]. A seguir, será apresentada a estratégia de paralelização adotada neste trabalho.

Dado um *sistema físico* a ser simulado, pode-se identificar os diversos componentes que o constituem. Tais componentes interagem entre si e recebem o nome de *processos físicos* (PF). Um programa de simulação para este sistema físico será composto de vários *processos lógicos* (PL), cada um deles mapeando um PF. Quanto às interações entre os PFs, elas serão simuladas através de troca de mensagens entre os PLs correspondentes. Cada mensagem vem acompanhada de um *timestamp*, que indica o instante do tempo da simulação em que a mensagem foi gerada. As mensagens trafegam por entidades lógicas, chamadas *canais*.

Num dado instante t do tempo de processamento, diferentes mensagens, com diferentes timestamps, podem estar sendo processadas por diferentes processos. A única restrição que se impõe é que , dentro de cada processo, as mensagens sejam processadas em ordem não decrescente de timestamp, independente da

ordem em que possam ter chegado, em tempo real. Na literatura, uma simulação paralela que obedece estes princípios recebe o nome de simulação paralela conservadora [Chandy & Misra 81].

4.2 Simulação Paralela Conservadora de RP

Ao se tentar construir uma simulação paralela conservadora de RP, a primeira preocupação que se deve ter é em como mapear a RP dada numa rede de processos, que constituirão o programa de simulação.

A estratégia de mapeamento mais natural parece ser aquela que mapeia cada nodo da RP em um processo da simulação e cada arco ligando dois nodos em um canal ligando processos correspondentes. Esta decomposição tem a vantagem de maximizar o paralelismo potencial da simulação e é particularmente adequada às máquinas altamente paralelas.

Existe, entretanto, uma dificuldade para se aplicar este mapeamento. Na definição de simulação paralela conservadora, viu-se que os processos interagem via troca explícita de mensagens. Por outro lado, existem nodos de RP cujo comportamento é determinado não só pelas informações que fluem através dos arcos, mas também pelo estado geral da rede.

Esse é precisamente o caso dos lugares de decisão, que precisam saber a marcação dos lugares de entrada das suas transições de saída para decidir a que transição mandar seus tokens disponíveis num dado instante.

Na próxima seção, será apresentado o protocolo de disparo de transição, denominado TFP, que contorna o problema apresentado na simulação paralela de RP.

5 TFP: Transition Firing Protocol

5.1 TFP - Definição

TFP - Transition Firing Protocol - é um protocolo de simulação paralela conservadora de RP que foi proposto originalmente por G. Thomas [Thomas91] para **máquinas de memória compartilhada**. Sua idéia principal é tornar as informações globais, que algum processo pode eventualmente precisar, acessíveis a todos os processos diretamente envolvidos, através de troca de mensagens entre eles. Mais precisamente, TFP vem resolver o problema criado pelos lugares de decisão. Neste trabalho, TFP foi adaptado para **arquiteturas de memória distribuída**.

Para a sua implementação, TFP exige que se acrescentem canais adicionais ao modelo original de simulação. Esse acréscimo se traduz na adição de um canal de controle (t, p) para cada par de processos já ligados por um canal (p, t).

No que se segue, os termos *lugar* e *transição* são usados indistintamente para designarem tanto as entidades de uma RP quanto os processos da simulação que os mapeiam. A acepção com que são empregados, em cada caso, pode ser inferida do próprio contexto.

TFP é um *handshake* duplo entre transições e seus lugares de entrada. O início de um ciclo do protocolo se dá quando tokens são recebidos por um lugar de decisão[1] Quando isto acontece, ele deve "saber" para qual transição de saída deverão ser enviados os tokens recebidos. Tipicamente, os tokens são enviados às transições que se tornam habilitadas com a presença de tokens no lugar em questão. Assim, o objetivo do primeiro handshake é detectar quais transições estão habilitadas. Uma vez detectadas as transições habilitadas, o lugar "decide"[2] quais transição(ões) fará(ão) disparo e a(s) comunica do número de tokens disponíveis a ela(s). Isso é feito na primeira fase do segundo handshake. Como uma transição pode ter vários lugares de

[1] O protocolo é definido para todos os lugares. Mas o caso mais ilustrativo acontece com os lugares de decisão.
[2] Essa decisão é feita baseando-se no tempo de disparo de cada transição habilitada.

decisão entre seus lugares de entrada, e cada lugar de decisão pode tomar decisões diferentes, ela só dispara quando todos os seus lugares de entrada tiverem tokens reservados para ela. Na segunda fase do segundo handshake, a transição comunica aos seus lugares de entrada se ela irá disparar e quantas vezes irá disparar.

Concretamente, as mensagens do TFP são as seguintes:

- **Activate**(n)@$T0$
  Quando um lugar $p1$ dispõe de n tokens (quer da marcação inicial, quer de depósitos recebidos das suas transições de entrada) no tempo de simulação $T0$, ele envia uma mensagem *Activate*(n)@$T0$ para cada uma das suas transições de saída não ativadas[3], avisando que ele possui n tokens no tempo $T0$.
- **Request**$(m, TD_1, TD_2, ..., TD_m)$@$T1$
  Quando uma transição t_1 recebe *Activate* de todos os seus lugares de entrada, ela responde (a todos eles) com a mensagem *Request*$(m, TD_1, TD_2, ..., TD_m)$@$T1$. m ($m \leq n$) é o valor mínimo entre os parâmetros n de todos os *Activates*; $T1$ ($T1 \geq T0$) é o maior entre todos os *timestamps* de *Activates* recebidos. Com esta mensagem, a transição avisa que ela pode disparar m vezes, no tempo $T1$, com tempos de disparo $TD_1, TD_2, ..., TD_m$, respectivamente, se seus lugares de entrada ainda dispuserem de m tokens.
- **Grant**(k)@$T2$
  Depois de enviado *Activate*, $p1$ fica esperando *Request* de todas as transições para as quais ele enviou *Activate*. Seja $T2$ o menor valor entre os $T1$s que acompanham estes *Requests*. Quando $p1$ receber todos os *Requests* de tempo $T2$, ele efetua uma resolução de conflito, "decidindo" quantos tokens reservar para cada um desses *Requests*. Se k tokens são reservados para uma transição t_i, então ele envia Grant(k)@$T2$, $0 \leq k \leq m$, a ela, indicando que k tokens estão disponíveis para que ela possa disparar.
- **Confirm**(j)@$T2$
  Depois de enviado *Request*, $t1$ fica esperando *Grant* de cada um dos seus lugares de entrada. Seja j o menor valor entre os parâmetros k desses *Grants*. O protocolo termina com o envio de *Confirm*(j)@$T2$, de t_1 para cada um dos seus lugares de entrada, indicando que t_1 disparou (ou irá disparar) j vezes.
- **Deposit**(1)@$T3$
  Terminado um ciclo do protocolo, j tokens são subtraídos de cada um dos lugares de entrada de t_1 e somados a cada um dos seus lugares de saída. A soma é feita progressivamente, quando mensagens *Deposit*(1)@$T3$, enviadas por t_1, são recebidas pelos seus lugares de saída.

  Essas operações são feitas, observando-se os tempos de disparo. Assim, $T3$ é a soma de $T2$ (instante a partir do qual t_1 fica habilitada) com TD_i (tempo de um dos disparos de t_1).

  Ou seja, quando a transição chega ao tempo T3, uma mensagem de *Deposit* é enviada para todos os seus lugares de saída. À medida que os lugares vão recebendo os *Deposits*, um token é adicionado aos já existentes no lugar.

5.2 TFP - Otimizado

Como foi visto, TFP é definido para todas as transições e seus lugares de entrada. Dependendo da topologia, entretanto, uma ou mais etapas do protocolo podem ser dispensadas. Considerando-se uma transição t qualquer, a versão otimizada preveria os seguintes quatro casos:

- A transição possui apenas um lugar de entrada e o único lugar de entrada da transição tem a mesma como sua única saída:
  Neste caso, apenas *Grant* é necessário.
- A transição possui mais de um lugar de entrada, mas todos eles tem esta transição como única saída:
  Neste caso, apenas *Grant* e *Confirm* são necessários.

[3] Transições de saída não ativadas são aquelas que não estão num ciclo de TFP.

- O único lugar de entrada da transição é um lugar de decisão. E, além disto, ele é o único lugar de entrada de todas as suas transições de saída:
  Neste caso, são necessários *Activate*, *Request* e *Grant*.
- A transição possui mais de um lugar de entrada. E, pelo menos, um deles é um lugar de decisão:
  Neste caso, são necessárias todas as etapas do protocolo.

6 Simulação Paralela

6.1 Ambiente

Esta seção descreve o ambiente de computação distribuída usado para executar a simulação paralela de RP. O ambiente utilizado na implementação do simulador de RP foi a rede de *workstations* do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (DCC-UFMG) e o software PVM - Parallel Virtual Machine, que simula um ambiente paralelo a partir de um ambiente distribuído.

O PVM pode ser visto como uma biblioteca de funções que facilita a criação, sincronização e escalonamento dos processos que fazem parte da aplicação paralela. Integra várias *workstations* como se fossem um único recurso, podendo o usuário se abstrair de problemas como, por exemplo, comunicação entre máquinas que possuem representação de dados diferentes.

Os processos da aplicação paralela que necessitam se comunicar são conectados aos pares por uma rede de comunicação. Os canais de comunicação implementam uma conexão lógica ponto a ponto entre dois processos. A comunicação se dá portanto através de troca de mensagens. Quando os processos que desejam se comunicar estão em máquinas distintas, a troca de mensagens ocorre através da rede local.

O PVM também provê um consistente e coerente paradigma de programação, sendo de fácil utilização como extensão de linguagens de programação amplamente difundidas como "C" e "Fortran". A linguagem utilizada neste trabalho foi "C".

6.2 Implementação

A estratégia de mapeamento utilizada foi a que parecia ser mais natural, ou seja, aquela que mapeia cada nodo da RP em um processo da simulação e cada arco ligando dois nodos em um canal ligando processos correspondentes. Assim, cada lugar e cada transição foram mapeados em um processo.

Existe um processo servidor que cria todos os processos (lugares e transições), determina a marcação inicial e as taxas de disparo de cada transição e fica suspenso até que a simulação termine.

Os processos criados obedecem ao protocolo FTP (descrito na seção 5) e assim simulam a movimentação dos *tokens* na rede, até que algum dos processos atinja o tempo máximo de simulação. Ao término da simulação, resultados são coletados e estatísticas são feitas para se determinar o tempo médio de ocupação de um lugar.

Apresentamos agora as Redes de Petri (RP) utilizadas nos testes, os resultados exatos obtidos pela solução analítica e os resultados obtidos através da simulação paralela.

6.2.1 Caso 1

O primeiro caso é um exemplo de uma RPE modelando um problema de *fork/join*, amplamente encontrado em sistemas de processamento paralelo e distribuído. O modelo de RPE é apresentado na figura 1.

A RPE em questão possui cinco lugares e cinco transições e apenas um *token*, com marcação inicial $M_0 = (1,0,0,0,0)$. As transições possuem as seguintes taxas de disparo:

$$\lambda_1 = 2.0$$
$$\lambda_2 = 1.0$$
$$\lambda_3 = 1.0$$
$$\lambda_4 = 3.0$$
$$\lambda_5 = 2.0$$

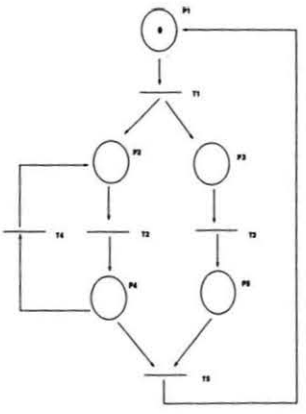

Figura 1: RPE modelando fork/join

A RPE modela operações seqüenciais (T_1 e T_2), operações paralelas (T_2 e T_3), *forking* (T_1), *joining* (T_5) e contenção (T_4 e T_5).

A solução analítica de RPE consiste em enumerar o conjunto de alcance do modelo, gerar a cadeia de Markov correspondente e calcular a probabilidade de presença de cada estado no conjunto de alcance, através da solução das equações de balanço global. O método para a obtenção de cada passo se encontra descrito em [MaCB84].

Lugares	Solução Exata	Simulação
P_1	0.11627	0.100982 ± 0.009009
P_2	0.72092	0.722366 ± 0.013206
P_3	0.23256	0.203403 ± 0.020494
P_4	0.16279	0.174246 ± 0.012122
P_5	0.65116	0.693290 ± 0.022693

Tabela 1: Utilização dos lugares para o caso 1

A tabela 1 fornece as utilizações dos lugares encontrados pela solução exata e pela simulação da RPE da figura 1. Os resultados da simulação foram obtidos a partir de 10 execuções do simulador, com tempo de simulação 100.0 e intervalo de confiança de 90%.

6.2.2 Caso 2

A RPE apresentada na figura 2 modela o comportamento de uma unidade de processamento que necessita da posse de certo recurso para executar alguma ação.

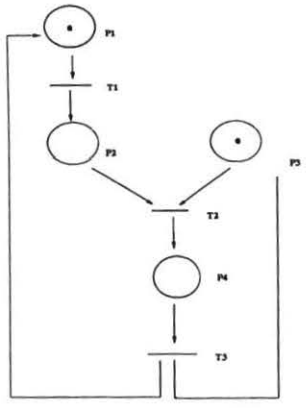

Figura 2: RPE modelando uma unidade de processamento

A RPE possui quatro lugares e três transições e marcação inicial $M_0 = (1, 0, 1, 0)$. As transições possuem as seguintes taxas de disparo:

$$\lambda_1 = 1.0$$

$$\lambda_2 = 1.0$$

$$\lambda_3 = 2.0$$

Um token em P_1 representa a unidade de processamento quando não está utilizando o recurso, um token em P_3 significa que o recurso está disponível, o tempo de disparo de T_2 é o tempo gasto para obtenção do recurso e o token em P_4 representa a utilização do recurso.

Lugares	Solução Exata	Simulação
P_1	0.40000	0.399786 ± 0.006453
P_2	0.40000	0.399588 ± 0.007274
P_3	0.80000	0.804357 ± 0.004545
P_4	0.20000	0.193570 ± 0.004497

Tabela 2: Utilização dos lugares para o caso 2

A tabela 2 fornece as utilizações dos lugares encontrados pela solução exata e pela simulação da RPE da figura 2. Os resultados da simulação foram obtidos a partir de 20 execuções do simulador, com tempo de simulação 100.0 e intervalo de confiança de 90%.

6.2.3 Caso 3

Outro exemplo de RPE é apresentado na figura 3 e modela sincronização de dois processos do tipo produtor/consumidor.

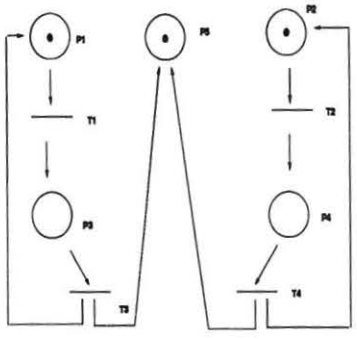

Figura 3: RPE modelando sincronização de processos

A RPE possui cinco lugares e quatro transições e marcação inicial $M_0 = (1, 1, 0, 0, 1)$. As transições possuem as seguintes taxas de disparo:

$$\lambda_1 = 1.0$$

$$\lambda_2 = 1.0$$

$$\lambda_3 = 2.0$$

$$\lambda_4 = 2.0$$

Lugares	Solução Exata	Simulação
P_1	0.7500	0.735496 ± 0.020875
P_2	0.7500	0.779024 ± 0.016567
P_3	0.2500	0.261643 ± 0.020980
P_4	0.2500	0.219308 ± 0.016750
P_5	0.5000	0.514520 ± 0.013793

Tabela 3: Utilização dos lugares para o caso3

A tabela 3 fornece as utilizações dos lugares encontra dos pela solução exata e pela simulação da RPE da figura 3. Os resultados da simulação foram obtidos a partir de 10 execuções do simulador, com tempo de simulação 100.0 e intervalo de confiança de 90%.

7 Conclusões

Este trabalho descreve a implementação de um simulador paralelo em uma rede de *workstations*. Foi feita adaptação de um protocolo para simulação paralela conservadora para o ambiente distribuído e fizemos uma proposta de otimização do mesmo. Simulações paralelas foram executadas em uma rede de *workstations* usando o PVM. Os resultados foram validados através da comparação dos mesmos com soluções analíticas exatas de Redes de Petri. Pesquisas futuras visam determinar métodos formais para a difícil tarefa de depuração de simuladores paralelos.

Referências

[1] M. A. Marsan *Performance Models of Mutiprocessor Systems*, MIT Press Series in Computer Systems, 1986.

[2] M. K. Molloy *Fundamentals of Performance Modeling*, Macmillan Publishing Company, New York, 1989.

[3] M. K. Molloy *On the Integration of Delay and Throughput Measures in Distributed Processing Models*, Ph.D. Thesis, University of California, Los Angeles, 1981.

[4] J. L. Peterson *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, N. J., 1981.

[5] J. L. Peterson *Petri Nets* , Computin Surveys, vol.9, No. 3, September 1977.

[6] M. M. Oliveira *Um ambiente de simulação para solução de modelos de redes de Petri estocásticas e generalizadas*, Relatório Técnico 90, Departamento de Ciência da Computação, UFMG, 1990.

[7] R. M. Fujimoto *Parallel discrete event simulation* , Communications of the *ACM* , vol.33, No. 10, October 1990.

[8] K. M. Chandy and J. Misra *Asynchronous distributed simulation via a sequence of parallel computations*, Communications of the *ACM*, vol.24, No. 4, April 1981.

[9] M. A. Marsan, G. Conte and G. Balbo *A Class of Generalized Stochastic Petri Net for the Performance Evaluation of Multiprocessor*, ACM transactions on Computer Systems, vol.2, No. 2, May 1984.

[10] G. Thomas *Parallel Simulation of Petri Nets* , Departament of Computer Science, TR 91-05-05, University of Washington.