ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Modelagem de Desempenho de Programas Paralelos Utilizando Redes de Petri Temporizadas

Trabalho de Conclusão de Curso

Engenharia da Computação

Nivia Cruz Quental
Orientador: Dra. Maria Lencastre Pinheiro de Menezes
Co-Orientador: Dr. Francisco Heron de Carvalho Junior

Recife, 30 de junho de 2006

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

Modelagem de Desempenho de Programas Paralelos Utilizando Redes de Petri Temporizadas

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Nivia Cruz Quental
Orientador: Dra. Maria Lencastre Pinheiro de Menezes
Co-Orientador: Dr. Francisco Heron de Carvalho Junior

Recife, 30 de junho de 2006

UNIVERSIDADE
DE PERNAMBUCO

Nivia Cruz Quental

Modelagem de Desempenho de Programas Paralelos Utilizando Redes de Petri Temporizadas

Resumo

Este trabalho trata do uso do formalismo das Redes de Petri Temporizadas para modelagem de desempenho de programas paralelos. Processamento paralelo é amplamente utilizado na indústria Petroquímica, Farmacêutica, Bioquímica, Medicina e Mecânica computacional, entre outras áreas. A sistematização da avaliação de desempenho destes programas é algo que vem sendo buscado há mais de vinte anos pela comunidade interessada em processamento de alto desempenho, uma vez que em suas áreas de aplicação a eficiência constitui um fator crítico.

Por outro lado, têm-se procurado meios de prover mecanismos de programação paralela que permitam oferecer um maior nível de abstração sem que haja maiores penalidades no desempenho. O modelo # (pronuncia-se *hash*) de componentes procura atender estes requisitos propondo observar uma aplicação como um conjunto de componentes sobrepostos, sob uma perspectiva orientada a interesses. Esta separação de interesses e o fato da especificação das comunicações no modelo # se basearem em expressões regulares sincronizadas permitem que a mesma seja traduzida para Redes de Petri lugar/transição, para a qual já existe um esquema de tradução. Redes de Petri são um formalismo matemático que permite que sistemas concorrentes sejam analisados, revelando propriedades importantes para a compreensão de seu funcionamento. As suas extensões temporizadas permitem análise de desempenho do sistema modelado, em nosso caso, dos programas paralelos.

Como estudo de caso, utilizamos um subconjunto dos NAS Parallel Benchmarks, um conjunto de aplicações de computação científica utilizado como referência para avaliação de desempenho em diversos trabalhos da área. A partir do comportamento do interesse de comunicação, foi traçada a Rede de Petri equivalente, baseada no esquema de tradução de programas # em Redes de Petri. Amostras dos tempos de comunicação foram retiradas e utilizadas como base na transformação para uma Rede de Petri Temporizada. Os resultados obtidos mostram que uso deste formalismo é bastante válido para a modelagem de desempenho de programas paralelos.

Abstract

This work is about the use of Timed Petri Nets for performance modeling of parallel programs. Parallel processing is largely applied in Oil and Pharmaceutics Industry, Biochemistry, Medicine, Computational Mechanics, etc. Ways for creating performance evaluation systematization for these programs has been researched for decades by the HPC (High Performance Computing) community, once, in HPC applications, efficiency is a critical factor.

At the same time, mechanisms of parallel programming capable to offer high level of abstraction without efficiency penalties have been searched. The # (read *hash*) model of components intends to match these requirements with its view of an application as a set of overlapped components, under a concern oriented perspective. This separation of concerns and the use of synchronized regular expressions for representing a communication concern allow the translation of this concern into place/transition Petri Nets, using a preexisting translation schema for this. Petri Net is a formalism for designing concurrent systems, allowing formal property analysis. Its timed extensions make performance analysis possible, in our case, parallel program performance evaluation.

As a case study, we used a subset of the NAS Parallel Benchmarks, a set of scientific computing applications, taken as reference for several works about performance evaluation. From the communication concern behavior, the equivalent Petri Net model was obtained. Communication time samples were taken to be the basis for generating a Timed Petri Net. The obtained results show the validity of Timed Petri Nets for performance modeling of parallel programs.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

(Dispostos por ordem de aparição no texto)
– Lê-se *hash*, nome dirigido ao modelo de componentes citado nesta monografia
SISD – *Single Instruction Single Data*
SIMD – *Single Instruction Multiple Data*
MIMD – *Multiple Instruction Multiple Data*
MPMD – *Multiple Program Multiple Data*
SPMD – *Single Program Multiple Data*
MPP – *Massive Parallel Processing*
NOW – *Network of Workstations*
PVM – *Paralell Virtual Machine*
MPI – *Message Passing Interface*
SPN – *Stochastic Petri Nets*
PRAM – *Parallel Random Access Memory*
CSP – *Communicating Sequential Processes*
CCS – *Calculus of Communicating Systems*
BSP – *Bulk-Synchronous Parallel*
LogP – *Latency Overhead Gap Processor*
NAS – *Numerical Aerodynamic Simulation*
LB – *Lower Is Better*
HB – *Higher Is Better*
NB – *Nominal Is Better*
TPS – *Transactions Per Second*
SPEC – *Standard Performance Evaluation Corporation*
NPB – *NAS Parallel Benchmarks*
RPC – *Remote Procedure Call*
CTMC – *Continuous Time Markov Chains*
PDF – *Probability Distributions Function*
GSPN – *Generalized Stochastic Petri Nets*
PAD – Processamento de Alto Desempenho
HPE – *Hash Programming Environment*
GEF – *Graphical Editing Framework*
HCL – *Hash Configuration Language*
CV – Coeficiente de variação
γ – Número de fases em uma distribuição PH
λ – Taxa de uma distribuição exponencial
r – Peso de uma transição de uma GSPN
μ_D – Média de uma amostra
σ_D – Desvio padrão de uma amostra
DTMC – *Discrete Time Markov Chain*

Em memória de minha avó e madrinha
Olga Cruz e Silva e meu primo, nosso
anjo Gabriel Vieira. Sei que vocês nos
observam e trazem alegria e serenidade
aos nossos corações sempre que
precisamos.

Agradecimentos

- A Deus, por permitir a minha existência e me dar oportunidade de conhecer tantas pessoas que foram e têm sido tão importantes para o meu amadurecimento em todos os sentidos, e por estar sempre ao meu lado, me guiando e protegendo.
- A toda minha família, em especial, meus pais Roberto e Ozita Quental e meus irmãos Roberto Junior e Jamily Quental. É minha fortaleza, me apóia e ama incondicionalmente, dando sempre palavras de força e incentivo.
- A meu amigo e orientador Dr. Francisco Heron de Carvalho Junior, cujo exemplo de inteligência e obstinação tento seguir. Mesmo diante de todas as dificuldades enfrentadas, sempre respeitou minhas limitações, mantendo apoio à minha jornada, que durou mais de três anos.
- A minha amiga e orientadora Dra. Maria Lencastre Pinheiro de Menezes, sempre atenciosa e disposta a me dar valiosas recomendações para este trabalho e durante a disciplina de Estágio Supervisionado. Jamais esquecerei a forma acolhedora como me recebeu como aluna.
- Ao professor Dr. Ricardo Massa Ferreira Lima, meu orientador de Iniciação Científica nos anos de 2004 e 2005, por me auxiliar no direcionamento de minha pesquisa, indicando-me os melhores caminhos a seguir. Agradeço também ao meu orientador de monitoria Dr. Carlos Alexandre Melo e todos os professores do DSC pelo grande esforço em conduzir o Departamento do qual tanto me orgulho de ter feito parte.
- Agradeço aos meus amigos da Poli, em especial, Adélia, César, Cleyton, Laura, Marcelo, Milena, Petrônio (“quarteto fantástico e os novos amiguinhos”) pelos momentos de descontração, pelo companheirismo e solidariedade durante projetos e momentos pessoalmente difíceis. Agradeço também a Fernando Aires, cuja preciosa amizade me ajudou em vários momentos, inclusive quando mais precisei.
- Aos professores e amigos do CEFET-PE Paula, Paulo, Washington, Alessandra, Anderson, Ewerson e Edson, responsáveis por uma fase maravilhosa da minha vida, quando conquistei grandes amizades.
- Agradeço aos amigos da E-Labore Soluções e do Centro de Informática da UFPE Glauco, Arnoldo, José Carlos, Orlando Campos, Leonardo Cole, Antônio Luis e amigos do projeto Samsung, pela experiência e amizade surgida durante a convivência diária. Agradeço também aos professores Rafael Lins, Paulo Maciel, Nelson Rosa, Sérgio Galdino, Marcília Campos, Renato Corrêa e ao mestrando Hilano Carvalho, da USP por toda experiência passada.

Capítulo 1

Introdução

Em diversas áreas da ciência, em especial em Ciência da Computação, podemos encontrar várias definições para o termo desempenho. De forma geral, pode ser visto como um tempo decorrido entre dois eventos, da mesma forma que pode estar associado a uma quantidade de operações processadas em uma dada unidade de tempo [27]. Por outro lado, na área de Sistemas Inteligentes, desempenho pode estar associado a métricas relacionadas à sua função objetivo, como velocidade de convergência, ou taxa de acertos. Nota-se, porém, um fator comum a todos estes casos: a necessidade de encontrar uma forma de caracterizar o comportamento do sistema, no qual serão baseadas as decisões de projeto. Uma escolha correta de métricas de aferição de desempenho é fundamental para uma interpretação coerente dos resultados.

Em particular, desempenho constitui um fator crítico no desenvolvimento de sistemas voltados ao Processamento de Alto Desempenho (PAD), escopo desta monografia, onde a maior preocupação encontra-se na questão da eficiência, medida em termos do tempo necessário a obter-se a solução de um problema. Nestes sistemas, a programação paralela emerge como uma técnica fundamental para atingir seus requisitos de eficiência. Nas próximas seções abordaremos momentos históricos na área de Processamento Paralelo, bem como o modelo # de componentes, utilizado como estudo de caso, as técnicas de avaliação de desempenho empregadas, principais objetivos e a estrutura geral da monografia.

1.1 Processamento Paralelo: Retrospecto e Conceitos

Por processamento paralelo entendemos a capacidade de sistemas computacionais executarem várias ações simultaneamente, com o objetivo de obter a solução de um problema de maneira mais rápida. Compreende um conjunto de técnicas, englobando *hardware* e *software* de um sistema computacional. Com respeito às arquiteturas paralelas de computadores, em 1966 foi publicada a conhecida Taxonomia de Flynn [20], propondo a seguinte classificação adequada para estas:

- **SISD** (*Single Instruction, Single Data*) – Nesta arquitetura, uma instrução é executada por vez e cada instrução manipula um dado por vez. Segue o modelo de Von Newmann (entrada – processamento – saída) com um único processador.

- **SIMD** (*Single Instruction, Multiple Data*) – Nesta arquitetura há um controle único das instruções, as quais são capazes de manipular conjuntos de dados. Como exemplo, temos as arquiteturas vetoriais, cujas instruções são capazes de realizar operações simultâneas sobre elementos de vetores e matrizes, com a vantagem de possuir paralelismo implícito, transparente ao programador. Tal arquitetura tem sido aplicada ou especializada para o processamento de alto desempenho.
- **MIMD** (*Multiple Instruction, Multiple Data*) – Nesta arquitetura, várias linhas de instruções operam independentemente sobre conjuntos de dados possivelmente disjuntos. É a arquitetura utilizada em processadores super-escalares e sistemas distribuídos em geral. Cada processador, ou unidade de processamento, executa suas instruções independentemente dos demais, exigindo que o programador aponte explicitamente como o paralelismo é gerenciado.

Outras classificações têm sido propostas para arquiteturas paralelas de computadores. Uma destas, bastante usual, considera a arquitetura de memória utilizada [19]. Na **arquitetura de memória compartilhada**, por exemplo, os processadores atuam independentemente, porém compartilhando um único espaço de endereçamento físico. Somente um processador tem acesso a essa memória por vez, podendo causar o problema de contenção de memória, conhecido em controle de concorrência. Tal fato limita o número de processadores nestas arquiteturas. A **arquitetura de memória distribuída** é a mais difundida atualmente e consiste em manter processadores operando independentemente, cada qual com sua própria memória física. O compartilhamento de dados nesse caso é feito através de sincronização por passagem de mensagens. Isso permite um acesso irrestrito à memória, sendo o usuário responsável pelo sincronismo das mensagens. Ao contrário do uso de memória compartilhada, não há limite teórico para o número de processadores, porém, o *overhead* de comunicação é um fator que degrada o desempenho, fator este que não está presente na arquitetura de memória compartilhada. A programação em passagem de mensagens ainda pode ser do tipo **MPMD** (*Multiple Program, Multiple Data*), onde cada processador executa um programa diferente, ou **SPMD** (*Single Program, Multiple Data*) onde todos os processadores executam o mesmo programa [22].

Nas décadas de 70 e 80, arquiteturas de processadores vetoriais dominaram o cenário da supercomputação, termo adotado para definir computadores cujo poder de processamento excedia em ordens de magnitude o poder computacional dos computadores pertencentes a sua classe diretamente inferior. Porém, ainda na década de 80, o uso de arquiteturas distribuídas emergiu como uma alternativa de maior escalabilidade, especialmente devido à evolução na tecnologia de interconexão de computadores em redes. Surgiram então arquiteturas distribuídas chamadas máquinas massivamente paralelas (MPPs), supercomputadores compostos de vários processadores independentes conectados em redes proprietárias de alta velocidade [49]. Disseminaram-se ainda as arquiteturas multiprocessadas, onde os processadores compartilhavam memória. Arquiteturas de supercomputação destacavam-se pelo alto custo de aquisição e manutenção. Nos anos 90, devido ao sucesso de alguns projetos acadêmicos, como os *Beowulfs* [5] e os NOWs [6], emergiram os *clusters*, redes de computadores de prateleira, como uma alternativa viável e mais barata aos supercomputadores, dando abertura às pesquisas sobre processamento paralelo em países em desenvolvimento. Embora o termo "supercomputador" tenha caído em desuso, o projeto de tais classes de arquiteturas não cessou devido aos clusters. Exemplos atuais desta classe de máquinas, segundo Dongarra [49] adequadas ao que ele chama de *capability computing*, são o Earth Simulator e IBM Blue Gene's, etc, máquinas que tem liderado a lista do Top 500 (*ranking* dos 500 computadores mais rápidos do mundo) nos últimos anos.

Em processamento paralelo, uma tendência nos últimos anos é a busca por um maior nível de abstração no nível de *software* e um maior desacoplamento no nível de *hardware*. A

tecnologia de grades computacionais [18] consiste em associar diversos computadores espalhados pelo mundo como um único supercomputador, oferecendo eficiência que não pode ser alcançada por arquiteturas de computador convencional, porém ainda limitada pela latência de redes de computadores geograficamente distribuídas. Grades computacionais oferecem algo mais que eficiência. O acesso a informações por uma maior quantidade de pessoas e a necessidade da oferta de *serviços pela web* constitui outras de suas motivações. Atualmente, o leque de interesses estende-se desde a questão militar a questões científicas como o avanço da Física, Biologia, Indústria, e-*Commerce*, entre outras áreas.

Os esforços conduzidos pela comunidade interessada em processamento paralelo em termos de *software*, porém, levaram esta evolução a ter um ritmo bem diferente da evolução em termos de *hardware*. Na década de 80 (trinta anos de defasagem com relação ao surgimento das primeiras máquinas paralelas), quando os maiores esforços estavam concentrados na questão da eficiência, as técnicas predominantes no desenvolvimento de aplicações eram as interfaces de passagem de mensagens, conhecidas pelo seu baixo nível e falta de portabilidade. A necessidade de portabilidade levou à adoção de modelos como o PVM [45] e o MPI [50] nos anos 90. A dificuldade de manutenção de programas paralelos deu abertura à pesquisa por novos paradigmas de programação, que buscassem oferecer um alto nível de abstração, com a adoção de técnicas de Engenharia de Software. Por ser uma questão bastante recente, ainda não existe um consenso em relação a este novo contexto.

Em termos de modelagem de desempenho para programação paralela, ao longo das décadas a comunidade científica vem procurando formas de sistematizar este processo. Nos anos 60, Amdhal [1] realizou estudos sobre os limites de desempenho de programas paralelos. Anos mais tarde surgiram modelos formais de desempenho, como o PRAM (*Parallel Random Access Machine*) [21], extensivamente utilizado na época. Mais tarde surgiram os modelos BSP [53], o qual propunha uma separação entre computação e comunicação, e o LogP [16] (baseado no BSP). Skillicorn em 1995 estabeleceu três critérios que um modelo de desempenho deveria possuir [24]: independência arquitetural, congruência (o custo de um programa deve refletir no seu modelo correspondente), e simplicidade de descrição.

1.2 O modelo # de componentes

O modelo # (leia-se "*hash*") de componentes teve origem na linguagem Haskell#, proposta no fim dos anos 90 por pesquisadores do Centro de Informática da Universidade Federal de Pernambuco [33]. Está voltado à programação paralela orientada a interesses (*concerns*) e se contrapõe à tradicional perspectiva orientada a processos [10]. Interesses relacionados à sincronização e computação são separados em dimensões ortogonais. Sob a perspectiva de modelos de coordenação, emprega conectores exógenos, onde o gerenciamento dos componentes é definido através de expressões de comportamento. Uma peculiaridade do modelo # é a não distinção entre componentes e conectores, sendo estes também tratados como componentes #. O modelo # permite conciliar alto nível de abstração, generalidade, portabilidade e eficiência. Esses requisitos têm sido discutidos e procurados pela comunidade interessada em programação paralela há mais de vinte anos, como podemos notar nesta citação de Skillicorn:

> "O grande problema com computação paralela hoje é encontrar o nível certo de abstração. Máquinas e arquiteturas mudam com freqüência. Há uma necessidade de se desenvolver software que possa ser executado em novas máquinas com alterações relativamente pequenas. Um bom nível de abstração deve ser embasado matematicamente. MPI facilita a construção de implementações eficientes mas não

ajuda muito com as propriedades que os programadores desejam." [1](Skillicorn, 1995 *apud* Grove,2003).

O modelo# foi projetado tendo em vista a representação comportamental de programas paralelos utilizando Redes de Petri, com vistas à análise de propriedades formais [15]. Adicionalmente, utilizando extensões temporizadas das mesmas, como as SPNs (*Stochastic Petri Nets*), podemos realizar análise de desempenho [46]. O ambiente # de programação encontra-se em fase de desenvolvimento, procurando ser um meio comum, onde várias tecnologias de programação paralela poderão ser integradas sob a perspectiva de componentes, sendo possível acoplar, futuramente, ferramentas de análise quantitativa.

1.3 Técnicas de avaliação de desempenho

Podemos destacar três tipos de abordagem em se tratando de avaliação de desempenho: medições [27], simulação [31] e modelagem analítica [7], à qual será dado maior foco neste trabalho.

A abordagem de medição, também conhecida como prototipação, consiste em retirar medições simples utilizando o sistema real como fonte direta de dados. Sua grande vantagem é a exatidão nos resultados, porém com o custo de obrigatoriamente já existir um sistema pronto e acessível para os seus analistas, o que nem sempre é possível.

A simulação procura utilizar técnicas computacionais para imitar o funcionamento sistemas do mundo real com o objetivo de compreendê-los, antecipando os efeitos produzidos por possíveis alterações. Isto permite que questões sobre o sistema sejam respondidas sem que o sistema real sofra qualquer tipo de perturbação. A principal motivação para simulação é a possibilidade de análise mesmo que o sistema ainda não exista, ou quando sua utilização é dispendiosa ou não apropriada (simulação de desastres, por exemplo). Os simuladores podem ser de propósito geral, como as ferramentas ARENA® e Microsaint [39], indicados para modelar sistemas industriais, de transporte e logística, ou de propósito específico, como o *Network Simulator* [51], uma ferramenta de simulação de redes, sobre a qual já existe um estudo sobre avaliação de desempenho de programas paralelos [31]. Recentemente têm sido propostos simuladores voltados a *Grids computacionais* [54].

A modelagem analítica, técnica enfatizada deste trabalho, abrange o uso da Teoria das Filas [30], Cadeias de Markov [36] e Redes de Petri Estocásticas (*Stochastic Petri Nets* – SPNs) [37]. Destacaremos o uso de SPNs, utilizando como estudo de caso central o modelo # de componentes. As Redes de Petri Estocásticas são extensões às Redes de Petri *lugar/transição*, associando variáveis aleatórias exponencialmente distribuídas às transições, permitindo a análise de desempenho, além das já conhecidas análises formais, presentes na maioria das ferramentas desenvolvidas para a confecção destas redes. As SPNs têm sido aplicadas na análise de diversos sistemas, desde sistemas de manufatura [34] a redes de computadores [52]. Maiores informações sobre modelagem analítica são encontradas na Seção 2.3 e no Apêndice B desta monografia.

1 "The big problem with parallel computation today is finding the right level of abstraction. Machines and architectures change frequently. There is a need to develop software that can run on new machines with relatively little change. A good level of abstraction should be mathematically based. MPI makes it easy to build efficient implementations but does not help much with properties that programmers want."

1.4 Objetivos

Este trabalho tem o objetivo de realizar um estudo voltado para a caracterização probabilística de programas paralelos segundo o modelo #. Um mapeamento do interesse de comunicação de programas # para SPNs deverá ser proposto utilizando como estudo de caso um subconjunto dos NAS Parallel Benchmarks (discutido no Capítulo 5), estendendo a tradução original para Redes de Petri lugar/transição com vistas à análise e verificação formal de programas paralelos. Esta extensão será um avanço na tradução automática de programas # para SPNs, o que facilitará o trabalho do usuário do ambiente de programação na tarefa de realizar análise de desempenho.

Nesta monografia também discutiremos os principais aspectos relacionados a avaliação desempenho, destacando a técnica de modelagem analítica. Discutiremos também a importância de uma sistematização para o processo de análise de desempenho e os erros mais comumente cometidos.

1.5 Estrutura geral da monografia

Este capítulo abordou as principais motivações para o desenvolvimento do trabalho, destacando a evolução dos modelos de programação, arquiteturas paralelas e as principais abordagens utilizadas em avaliação de desempenho. A seguir, no Capítulo 2, serão discutidos conceitos e idéias relacionados à avaliação de desempenho de uma forma geral, destacando uso da modelagem analítica. A partir do Capítulo 3 as Redes de Petri são introduzidas e exploradas em seus conceitos, propriedades e extensões, com foco nas variantes estocásticas. O Capítulo 4 é inteiramente dedicado ao modelo # de componentes, expondo motivações, impactos, conceitos básicos, sintaxe, e seu esquema clássico de tradução para Redes de Petri lugar/transição. O mapeamento de programas # em SPNs é mostrado em forma de estudo de caso, descrevendo todo o processo experimental e de análise de resultados no Capítulo 5. Finalmente, no Capítulo 6, teremos as conclusões e trabalhos propostos a partir dos resultados obtidos.

Capítulo 2

Conceitos básicos de avaliação de desempenho

Em geral, no processo de desenvolvimento de um sistema computacional tem-se como objetivo obter do mesmo o maior benefício possível ao menor custo. Em Processamento de Alto Desempenho entende-se por benefício a eficiência na utilização de recursos durante processamento de uma computação, o que deve resultar na minimização do tempo de resposta da aplicação. Esta área da computação é muito valorizada em diversos campos da ciência e engenharia, como na indústria Petroquímica, Farmacêutica, Bioquímica, na Medicina, Mecânica Computacional, entre outros. Em todas estas áreas a avaliação do desempenho é crucial para que sejam tomadas decisões de projeto e/ou implementação necessárias para a sua melhoria. Nas próximas subseções, as métricas de desempenho mais utilizadas, os passos necessários para o sucesso no processo de avaliação dessas métricas, a abordagem de modelagem analítica e erros comuns cometidos durante o processo de avaliação serão discutidos. Finalmente, daremos destaque às questões referentes à avaliação de desempenho de programas paralelos.

2.1 Métricas comuns de desempenho

A maioria dos conceitos a seguir está descrita em [30]. As métricas de desempenho mais comumente encontradas são:

- *Tempo de resposta* – Definido como o intervalo entre uma requisição de um sistema (ou parte dele) e a sua resposta. Geralmente cresce em proporção com a carga computacional aplicada. É classificado como uma métrica *Lower is Better* (LB – menor é melhor). Em programação paralela é conhecido como *tempo de execução*, sendo o tempo que transcorre desde quando o primeiro processador começa a executar um problema até quando o último processador completa a execução.
- *Throughput ou Vazão* – Taxa em que as requisições são servidas pelo sistema. Em outras palavras, é a relação de atendimentos, pacotes, ou qualquer tipo de dado em uma unidade de tempo. Inicialmente cresce com a carga, tendendo a se estabilizar em certo valor de

carga computacional. É classificado como uma métrica *Higher is Better* (HB – maior é melhor).

- *Utilização* - Fração de tempo que um servidor está ocupado. Classificado como *Nominal is Best* (NB – nominal é o melhor).
- *Confiabilidade* – Medida como a probabilidade de erros ou tempo médio entre erros.
- *Disponibilidade* - Fração de tempo em que o sistema está disponível para receber novas requisições.
- *Taxa custo/desempenho* – Usada para comparar dois sistemas; um exemplo é a comparação entre dois sistemas de processamento de transações, comparados em termos de unidades monetárias por TPS (transações por segundo).

Ao longo do texto também será encontrado o termo carga ou *workload*, que se refere à quantidade de dados que trafegam no sistema, fator decisivo na obtenção das métricas de desempenho. O valor da carga varia de sistema para sistema. No caso de um programa paralelo, a carga pode ser determinada pela quantidade de iterações em um algoritmo, ou até mesmo uma matriz na memória.

2.2 Sistematização da avaliação de desempenho

Segundo Jain [30], a maioria dos problemas de desempenho são peculiares a certo contexto, e as métricas, cargas e resultados e técnicas não podem ser reaproveitados. Porém, alguns passos são comuns a qualquer situação encontrada:

- *Estabelecer objetivos e definir o sistema* – A definição dos limites do sistema afeta a escolha das métricas e cargas;
- *Listar serviços e saídas* – Algumas saídas são desejáveis e outras não. O analista deve listá-las de acordo com suas necessidades;
- *Selecionar métricas* – Selecionar o critério de avaliação de desempenho;
- *Listar parâmetros* – Incluindo parâmetros de *hardware* ou *software* que afetam o desempenho;
- *Selecionar fatores a estudar* – Os parâmetros a serem variados são denominados *fatores* e seus valores são os *níveis* ; em geral, a lista de fatores e níveis é maior que os recursos podem permitir;
- *Selecionar a técnica de avaliação* – A seleção da técnica correta depende do tempo e dos recursos disponíveis para resolver o problema e do nível de exatidão desejado;
- *Selecionar a carga* – Para produzir uma carga significativa é preciso medir e caracterizar a carga do sistema;
- *Analisar e interpretar os dados* – Os resultados provêm a base na qual os analistas e tomadores de decisão podem traçar conclusões;
- *Apresentar os resultados.*

2.3 Modelagem analítica

Modelagem analítica é uma abordagem para avaliação de desempenho de sistemas bastante recomendável quando é necessário obter resultados rapidamente e de maneira simples. Em geral, as métricas de desempenho são representadas por variáveis aleatórias que seguem uma distribuição de probabilidade escolhida pelo analista que represente mais fielmente o sistema real. Os sistemas são modelados por processos estocásticos, que são conjuntos de variáveis aleatórias

dentro de um espaço de estados indexadas no tempo (que pode ser discreto ou contínuo). A análise consiste em avaliar as probabilidades de transição de um estado para outro, seja em regime transiente, seja em regime estacionário. Detalhes sobre processos estocásticos e Cadeias de Markov (um tipo especial de processo estocástico) podem ser encontrados no Apêndice B. O nível de exatidão dos resultados, porém, é, em geral, inferior ao dos resultados obtidos por medições ou simulação. Uma vantagem é a simplicidade e baixo custo na obtenção, apresentação e interpretação dos resultados.

A seguir serão enumeradas algumas das distribuições mais encontradas em modelagem analítica. Muitos destes conceitos podem ser encontrados em [23], [30] e [48]. Uma descrição e as curvas que caracterizam as distribuições citadas a seguir são mostradas na Tabela 1. Entre as distribuições contínuas temos:

- **Distribuição Normal** – A distribuição normal constitui uma família de curvas determinadas por dois parâmetros: média e desvio padrão. A distribuição normal padrão possui média 0 e desvio igual a 1. É utilizada quando a aleatoriedade é causada por várias fontes independentes e aditivas (erros de medição, por exemplo). A justificativa mais comum para o seu uso em modelagem é a aplicação do Teorema do Limite Central que estabelece que a distribuição das médias de várias amostras que seguem uma distribuição qualquer, retiradas em um experimento, tende a ser normal com o aumento do número de amostras.
- **Distribuição Lognormal** – É a distribuição do logaritmo de uma variável que segue uma distribuição normal. Geralmente o produto de várias variáveis aleatórias positivas segue a distribuição lognormal, comumente empregada na modelagem de tempos de serviços.
- **Distribuição Weibull** – Inicialmente proposta para modelagem em problemas de confiabilidade de equipamentos. Possui formato semelhante à normal se seu parâmetro de formato C for igual a 3.602. Para valores acima deste, a curva apresenta uma cauda longa à esquerda. Para valores inferiores, possui uma longa cauda à direita. Se o parâmetro de formato é igual a 1, a curva possui formato de L. Para valores altos de C, possui um pico íngreme próximo à média.
- **Distribuição Exponencial** – Sua principal característica é a ausência de memória. O conhecimento prévio do tempo de ocorrência de um evento não auxilia na determinação do tempo de ocorrência do próximo evento. Muito utilizada na modelagem de tempos decorrentes entre falhas de sistemas, devido à sua alta variabilidade. A ausência de memória dá suporte à solução de Cadeias de Markov de Tempo Contínuo.
- **Distribuição Erlang** – Comumente usada em modelos de fila como uma extensão da distribuição exponencial. É, portanto, uma distribuição *Phase-Type*, como veremos adiante. É utilizada na modelagem de tempos de serviço em um modelo de fila, ou tempo entre falhas de um sistema.
- **Distribuição Gama** – A distribuição Gama é uma generalização das distribuições Exponencial e Erlang. É utilizada em modelos de filas, para modelar tempos de serviço e reparo de um recurso, similarmente à distribuição Erlang.
- **Distribuição Uniforme Contínua** – Utilizada quando se desconhece quase completamente o sistema e os únicos dados disponíveis são os limites mínimo e máximo.
- **Distribuição de Pareto** – Foi originalmente criada para representar distribuição de renda. É uma distribuição cuja *pdf* (função de distribuição de probabilidade, ver Apêndice A) é caracterizada por uma curva acentuada, o que indica que valores possuem uma alta variabilidade.
- **Distribuição Beta –** Representa uma variável cujos valores variam dentro de um limite superior e um inferior, cujo formato depende de dois parâmetros C e D. Em geral é usada

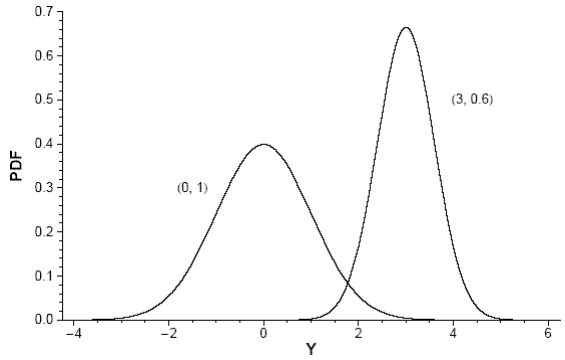
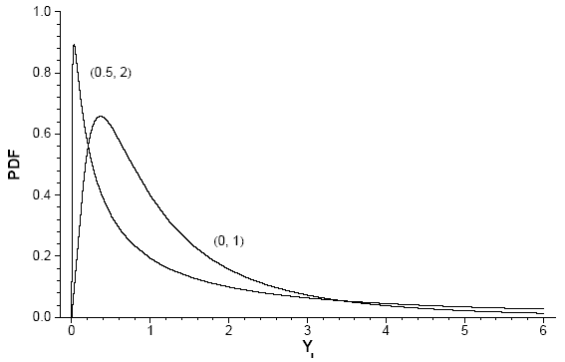
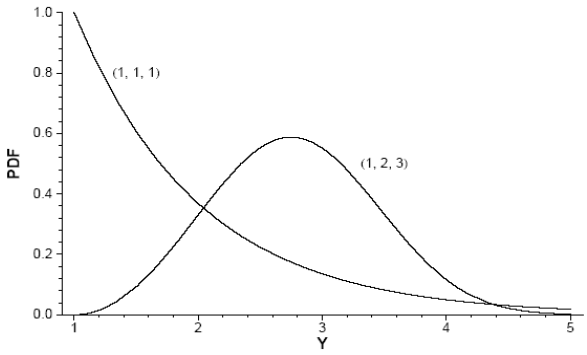
para modelar proporções aleatórias, como a fração de chamadas remotas (RPCs) que demoraram mais que um tempo especificado. Pode assumir um grande número de formas, dependendo de seus parâmetros C e D.

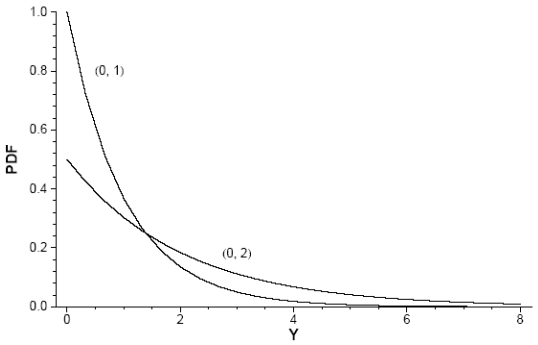
- **Distribuição Triangular** – Assim como a uniforme, é usada quando o sistema é pouco conhecido, porém, além dos limites máximo e mínimo, o valor mais provável também é conhecido.

Entre as distribuições discretas temos:

- **Distribuição Geométrica** – Distribuição discreta equivalente à exponencial, por não possuir memória. Usada na modelagem do número de tentativas entre falhas sucessivas. Sua ausência de memória auxilia na solução de cadeias de Markov de tempo discreto.
- **Distribuição de Poisson** – Pode ser utilizada para modelar o número de chegadas de um evento em um dado intervalo de tempo (número de requisições, falhas ou consultas).
- **Distribuição Uniforme Discreta** – Versão discreta da distribuição uniforme contínua, tomando um número finito de valores, cada qual com a mesma probabilidade.

Tabela 1. Distribuições de probabilidade

Curva	Descrição
	**Distribuição Normal** Tipo: contínua Parâmetros: A = μ - média (localização) B = σ - desvio padrão (escala) $\text{Pdf} = \frac{1}{B\sqrt{2\pi}}\exp\left(\frac{1}{2}\left(\frac{y-A}{B}\right)^2\right)$ $\text{Cdf} = \Phi\left(\frac{y-A}{B}\right)$ Φ é a distribuição normal padrão
	**Distribuição Lognormal** Tipo: contínua Parâmetros (no espaço logarítmico): A =ζ (localização) B = σ (escala) $\text{Pdf} = \frac{1}{By\sqrt{2\pi}}\exp\left(\frac{1}{2}\left(\frac{\log(y)-A}{B}\right)^2\right)$ $\text{Cdf} = \Phi\left(\frac{\log(y)-A}{B}\right)$, Φ é a distribuição normal padrão
	**Distribuição Weibull** Tipo: contínua Parâmetros: A = ξ (localização) B = α (escala) C = c (forma) $\text{Pdf} = \frac{C}{B}\left(\frac{y-A}{B}\right)^{C-1}\exp\left(-\left(\frac{y-A}{B}\right)^C\right)$ $\text{Cdf} = 1-\exp\left(-\left(\frac{y-A}{B}\right)^C\right)$

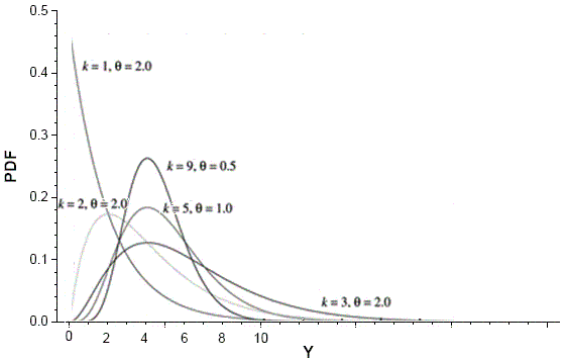


Distribuição Exponencial

Tipo: contínua
Parâmetros:
A = θ (localização)
B = λ (escala)

$$\text{Pdf} = \frac{1}{B}\exp\left(\frac{A-y}{B}\right)$$

$$\text{Cdf} = 1-\exp\left(\frac{A-y}{B}\right)$$

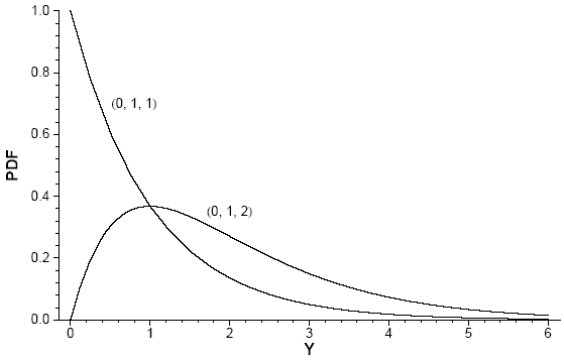


Distribuição Erlang

Tipo: contínua
Parâmetros:
A = θ – escala
B = k – número de fases (forma)

$$\text{Pdf} = \frac{y^{B-1}\exp\left(-\frac{B}{A}\right)}{(B-1)!A^{B}}$$

$$\text{Cdf} = 1-\exp\left(-\frac{B}{A}\right)\left[\sum_{i=0}^{B-1}\frac{(y/A)^{i}}{i!}\right]$$

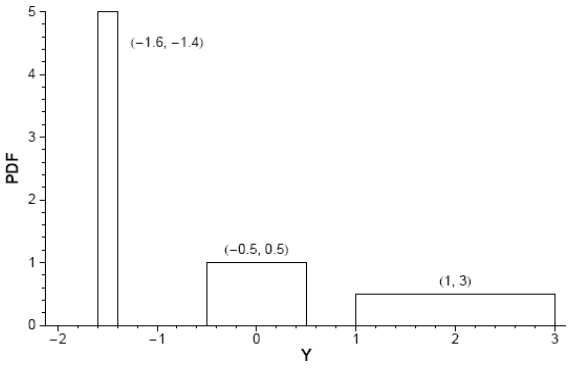


Distribuição Gama

Tipo: contínua
Parâmetros:
A = γ (localização)
B = β (escala)
C = α (forma)

$$\text{Pdf} = \frac{1}{B\Gamma(C)}\left(\frac{y-A}{B}\right)^{C-1}\exp\left(\frac{A-y}{B}\right)$$

$$\text{Cdf} = \Gamma\left(C,\frac{y-A}{B}\right)$$

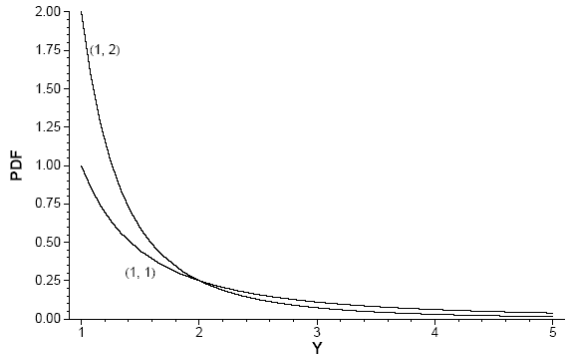


Distribuição Uniforme contínua

Tipo: contínua
Parâmetros:
A = limite mínimo (localização)
B = limite máximo (escala)

$$\text{Pdf} = \left(\frac{1}{B-A}\right)$$

$$\text{Cdf} = \left(\frac{y-A}{B-A}\right)$$

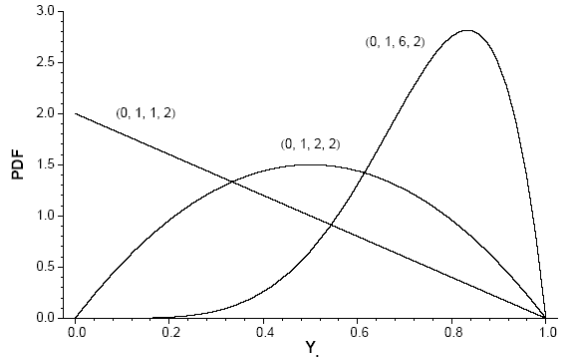


Distribuição de Pareto

Tipo: contínua
Parâmetros:
A = k (localização e escala)
B = a (forma)

Pdf = $\frac{BA^B}{y^{B+1}}$

Cdf = $1-\left(\frac{A}{y}\right)^B$

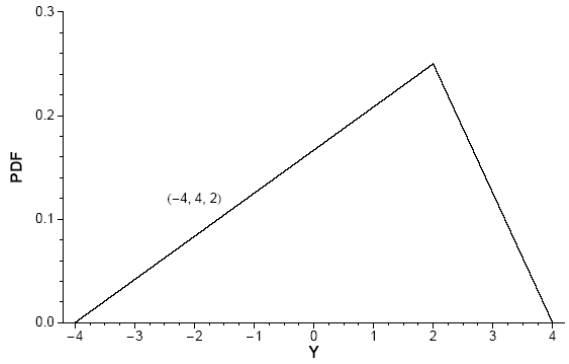


Distribuição Beta

Tipo: contínua
Parâmetros (no espaço logarítmico):
A (localização)
B (escala)
C,D (p, q) (forma)

Pdf = $\frac{\Gamma(C-D)}{\Gamma(C)\Gamma(D)(B-A)^{C+D-1}}(y-A)^{C-1}(B-y)^{D-1}$

Cdf = $I\left(\frac{y-A}{B-A}, C, D\right)$

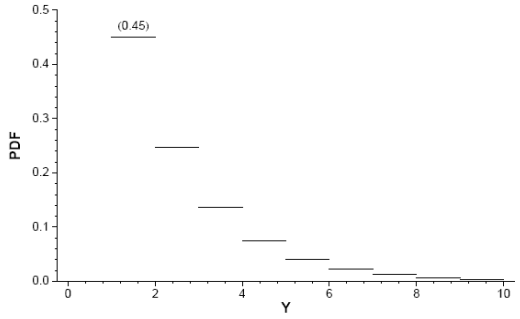


Distribuição Triangular

Tipo: contínua
Parâmetros:
A = (localização)
B = limite máximo (escala)
C = moda (forma)

Pdf = $2\frac{(y-A)}{(B-A)(C-A)}$, y<C $\quad 2\frac{(B-y)}{(B-A)(B-C)}$, y≥C

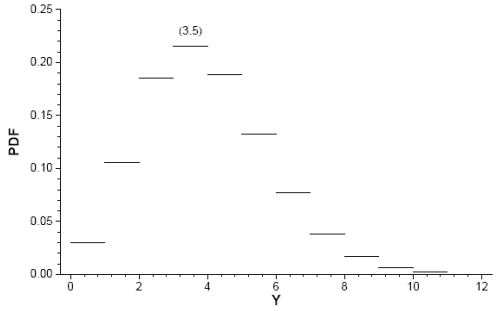
Cdf = $\frac{(y-A)^2}{(B-A)(C-A)}$, y<C $\frac{\left(A(B-C)+B(C-2y)+y^2\right)}{(A-B)(B-C)}$, y≥C



Distribuição Geométrica

Tipo: discreta
Parâmetros:
A = p (probabilidade de sucesso)

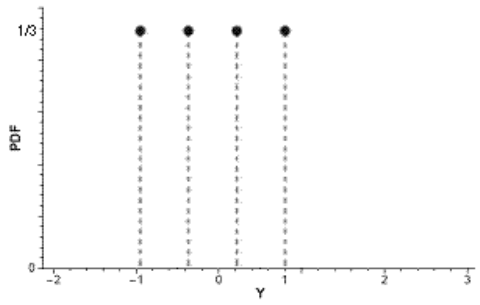
Pdf = $A(1-A)^{y-1}$



Distribuição de Poisson

Tipo: discreta
Parâmetros:
A = θ (esperança)

Pdf = $\frac{\exp(-A)A^y}{y!}$



Distribuição Uniforme (discreta)
Tipo: discreta
Parâmetros:
A = limite inferior
B = limite superior

$$\text{Pdf} = \frac{1}{B - A + 1}$$

$$\text{Cdf} = 0,\ y<A;\quad \frac{y - A + 1}{B - A + 1},\ A \le y < B\ ;\ 1,\ B \le y$$

Como afirmado anteriormente, existe uma família especial de curvas, derivadas de exponenciais, conhecidas como distribuições *Phase-type* ou distribuições PH. A seguir, alguns exemplos de distribuição PH (Figura 1):

- **Erlang** – Seqüência de *p* fases exponenciais.
- **Hiper-exponencial** – *p* fases exponenciais em paralelo.
- **Hipo-exponencial** – Seqüência de *p* fases, sendo permitido que cada uma possua uma taxa diferente; a distribuição Erlang é um caso particular.
- **Cox** – Generalização da distribuição Erlang onde o estado absorvente pode ser alcançado a partir de todos os estados da cadeia de Markov.

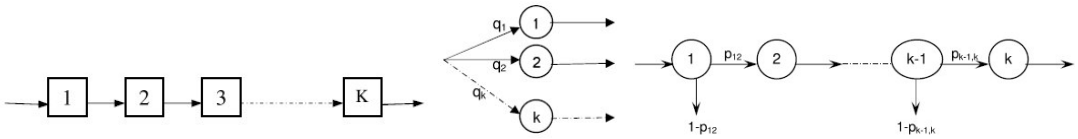


Figura 1. Representação de distribuições PH: a) Erlang b) Hiper-exponencial c) Cox

As principais propriedades encontradas nas distribuições PH são:

- Densa: toda distribuição não-negativa pode ser aproximada a uma distribuição PH.
- Modelagem markoviana: todas as distribuições PH mantêm as propriedades advindas do uso de distribuição exponencial.
- Informativo estruturalmente: distribuições PH são versáteis e tratáveis computacionalmente.

Em diversos problemas de modelagem, inclusive em nosso estudo de caso, encontraremos situações onde será conveniente a aproximação de distribuições para distribuições PH, por permitirem a modelagem markoviana, possibilitando-nos continuar trabalhando com SPNs. É importante lembrar também que nesta monografia estamos procurando caracterizar analiticamente o comportamento de programas paralelos, porém, isto não significa que este tipo de modelagem seja melhor que simulações e medições para este caso. A importância de se avaliar programas paralelos pela ótica analítica através das Redes de Petri Estocásticas, é que elas oferecem a possibilidade de utilizar um mesmo modelo para analisar tanto o aspecto quantitativo quanto as propriedades formais do programa.

2.4 Erros comuns em avaliação de desempenho

Abaixo são listados erros comumente encontrados no processo de avaliação de desempenho de sistemas [30], geralmente cometidos por conceitos mal construídos, descuidos simples, ou falta de conhecimento sobre as técnicas de avaliação existentes.

- *Falta de objetivos* – Todo modelo deve ser construído com um objetivo em mente;
- *Objetivos tendenciosos* – O analista deve atuar como um juiz. Não deve ter conclusões pré-concebidas a partir de suas próprias convicções;
- *Abordagem não sistemática* – Os objetivos devem ser identificados, bem como parâmetros, fatores, métricas e carga;
- *Análise sem compreender o problema* – Apesar de existirem avanços na procura por automatização no processo de avaliação de desempenho, o mesmo ainda é considerado uma arte, de modo que é importante conhecer bem o problema em questão;
- *Projeto de experimento inadequado* – Relacionado ao número adequado de execuções do experimento e de parâmetros variados ;
- *Nível de detalhe inapropriado* – Os objetivos têm impacto significativo sobre o que é analisado e como esta análise é feita;
- *Ausência de análise* – Muitos profissionais são ótimos coletores de dados, porém, não sabem o que fazer com estes após as medições;
- *Análises errôneas*;
- *Ausência de análise de sensibilidade* – Sem uma análise de sensibilidade, não é possível ter certeza sobre como os resultados podem variar com uma pequena mudança nos parâmetros;
- *Ignorar erros nas entradas*;
- *Tratamento inadequado dos Outliers* – *Outliers* são valores que são excessivamente altos ou baixos comparado à maioria dos resultados. É importante descobrir os motivos destes desvios e ignorá-los adequadamente;
- *Não considerar mudanças futuras*;
- *Ignorar variabilidade* - Como afirmado anteriormente, a carga ou *workload*, que se refere à quantidade de dados que trafegam no sistema, é fator decisivo na obtenção das métricas de desempenho;
- *Análise excessivamente complexa*;
- *Apresentação inapropriada de resultados*;
- *Ignorar aspectos sociais* – Os impactos devem ser identificados para que se compreenda a importância do sistema em questão;
- *Omitir considerações e limitações*.

2.5 Avaliação de desempenho de programas paralelos

Segundo Foster [22], desempenho em programação paralela é uma questão multifacetada. Além do tempo de execução e escalabilidade (capacidade que um programa tem de se adaptar a diferentes granularidades) é importante considerar os mecanismos pelos quais os dados são gerados, armazenados, transmitidos na rede, movidos de/para o disco e passado para diferentes estágios de computação. As principais métricas encontradas nesta área são: tempo de execução e *throughput* (citados anteriormente), requisitos de memória, requisitos de entrada/saída, latência (tempo que uma unidade de informação leva para chegar ao seu destino através de um meio de comunicação), custos de projeto, de implementação, verificação de *hardware*, de potencial de

reuso e de portabilidade. Uma métrica bastante conveniente em programação paralela é a *eficiência* (Equação 1) e o *speedup* (Equação 2), frações de tempo que os processadores passam realizando trabalho útil, caracterizando a efetividade de se utilizar paralelismo em relação a uma versão seqüencial. Neste caso, T_1 é o tempo de execução do programa em um processador e T_p é o tempo de execução do programa em P processadores.

$$E = \frac{T1}{PT_P} \qquad \text{(Eq. 1)}$$

$$S = \frac{T1}{T_P} \qquad \text{(Eq. 2)}$$

Durante a execução de um programa paralelo, um processador pode estar computando, comunicando ou ocioso. O *tempo total de execução* pode ser definido de duas formas: pode ser a soma da computação T^i_{comp}, da comunicação T^i_{comm} e do tempo ocioso T^i_{idle} de um processador arbitrário conforme a Equação 3, ou a soma destes tempos em todos os processadores dividido pelo número de processadores P (Equação 4).

$$T = T^i_{comp} + T^i_{comm} + T^i_{idle} \qquad \text{(Eq. 3)}$$

$$T = (T_{comp} + T_{comm} + T_{idle})/P \qquad \text{(Eq. 4)}$$

Entende-se o tempo de computação como o tempo gasto por um processador realizando computações. Esse tempo, geralmente, depende da carga do problema. O tempo de comunicação é o tempo total que um processo leva enviando ou recebendo mensagens. Depende, em geral, de um tempo de inicialização e do tamanho da mensagem. Um processador pode estar ocioso por falta de computação ou por falta de dados. No primeiro caso este problema é evitado utilizando-se uma política de balanceamento de carga; no segundo caso, quando o processo espera algum dado que ainda não foi calculado, o problema deve ser resolvido com uma melhor estruturação do código.

A realização de experimentos em programas paralelos requer uma coleta cautelosa dos dados mensurados. A obtenção de tempos de execução pode ser realizada coletando o tempo máximo entre todos os processadores, ou escolhendo um processador, onde o estudo será realizado, sempre repetindo os experimentos, para obter resultados mais precisos. Possíveis causas de variação incluem:

- Um algoritmo não determinístico;
- Cronômetro pouco preciso;
- Custos de inicialização e finalização;
- Interferência por outros processos em uma rede não dedicada;
- Contenção da rede;
- Alocação aleatória de recursos.

Entretanto, mesmo tomando todas as precauções, a natureza idealizada do modelo raramente será compatível com os resultados obtidos no sistema real. Maiores discrepâncias, porém, indicam que há erros no modelo ou problemas com o algoritmo. No segundo caso, é importante realizar um estudo de *profiling,* cujo conceito já foi visto anteriormente. No caso do primeiro, o modelo deve ser repensado.

Capítulo 3

Redes de Petri e extensões

Neste capítulo daremos destaque ao formalismo das Redes de Petri enquanto ferramenta de modelagem analítica utilizada tanto para realizar análise formal de propriedades quanto de desempenho, esta última através das extensões temporizadas. Nas próximas seções serão apresentados conceitos básicos relacionados às Redes de Petri, bem como propriedades formais que podem ser analisadas através das mesmas e, finalmente, extensões que introduzem o conceito de tempo, com as Redes de Petri Temporizadas. Estas últimas serão utilizadas para avaliação de desempenho de programas paralelos em um estudo de caso no Capítulo 5.

3.1 Conceitos básicos

Carl Adam Petri introduziu o conceito de Rede de Petri em sua tese para obtenção do título de doutor em 1962. Desde então, tal formalismo tornou-se amplamente conhecido, sendo posteriormente aprimorado e estendido. Em [34] podemos encontrar a definição de uma Rede de Petri lugar/transição como o tipo mais simples de Rede de Petri,sendo representado por uma 5-upla PN (P,T,F,W,M_0) composta por:

- **Um conjunto de lugares P**, onde cada lugar é representado graficamente por uma circunferência, simbolizando um depósito de recursos, chamados de marcas (simbolizadas por pequenos círculos pretos dentro de um lugar). A quantidade de marcas que cada lugar comporta modela o estado do sistema, ou uma condição que afeta a mudança de estado.
- **Um conjunto de transições T**, que simbolizam eventos capazes de governar as mudanças de estado do sistema. São representados graficamente por barras.
- **Um conjunto de arcos F**, que define o fluxo de relações; graficamente é uma seta que liga lugares a transições modelando o fluxo de marcas ao longo da rede;
- **Um conjunto de pesos W**, referente a cada arco; quando não indicado junto ao arco, seu valor é 1; o peso indica quantas marcas deverão ser consumidas caso o arco aponte para uma transição ou quantas marcas deverão ser produzidas em um lugar, caso o arco aponte para um lugar.
- **A marcação inicial M_0**, que determina o estado inicial do sistema modelado.

Uma representação alternativa é tomar uma rede de Petri por uma 5-upla (P,T,I,O,K). Neste caso, P e T são os conjuntos de lugares e transições respectivamente, e I e O são matrizes de mapeamento de lugares de entrada para transições e de transições para lugares de saída,

respectivamente. O conjunto K mapeia os lugares à sua capacidade, isto é, o numero máximo de marcas suportadas pelo lugar. Uma matriz de especial interesse é a matriz de incidência $C = O - I$, usada em diversas equações em análise formal. Ao longo do texto adotaremos também a notação N(R, M_0), onde N é a Rede de Petri em questão, e R é a 5-upla (P,T,I,O,K), com marcação inicial M_0. Também será encontrada a notação M(pi), que se refere ao número de marcas em um lugar p_1 quando a rede possui marcação M em determinado instante. Na Figura 2 temos um exemplo de modelo em Rede de Petri:

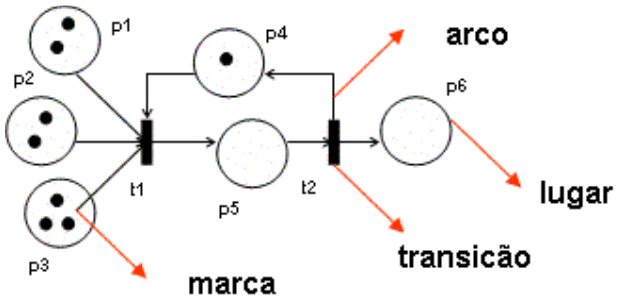


Figura 2. Representação de rede de Petri

Para este exemplo, temos as matrizes I e O (ver Figura 3) onde as colunas são as transições t_1 e t_2 e as linhas são os lugares de p_1 a p_6; o conteúdo de cada posição da matriz indica o peso relacionado ao arco que liga um lugar de entrada a uma transição, no caso da matriz I, ou então ao arco que liga a transição a um lugar de saída, no caso da matriz O. Em caso de ausência de arcos, considera-se o valor 0. Por exemplo, na posição (3,1) da matriz I temos o valor 1, pois há um arco de entrada para a transição t_1 a partir do lugar p_3; já a posição (3,1) da matriz O possui valor zero, pois não há arcos de saída da transição t_1 para o lugar p_3.

$$I = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \quad O = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{(Eq. 5)}$$

O disparo das transições (mudança de estado devido à execução de ações) é controlado pela marcação dos seus lugares de entrada. Uma transição está habilitada a disparar quando o número de marcas em cada um dos seus lugares de entrada é maior ou igual ao peso do arco que os liga. Por exemplo, na Figura 3a temos a transição t1 habilitada, pois os lugares p_1, p_2, p_3 e p_4 possuem no mínimo uma marca, que será "transportada" pelos arcos ligados a eles, cujo peso é 1; o disparo da transição irá consumir uma marca de p_1, p_2, p_3 e p_4 e produzirá uma marca em p_5 (pois o peso do arco que liga t_1 a p_5 é 1). Isto gera um novo estado na rede, como podemos ver na Figura 3b. Uma vez que não há mais marcas em p_4, t_1 não possui condições de ser habilitada neste estado. Porém, notamos que a transição t_2 passa a ser habilitada a disparar, consumindo a marca em p_5 e produzindo uma marca no lugar p_4 e outra no lugar p_6. Neste caso t_1 fica novamente habilitada a disparar e assim por diante.

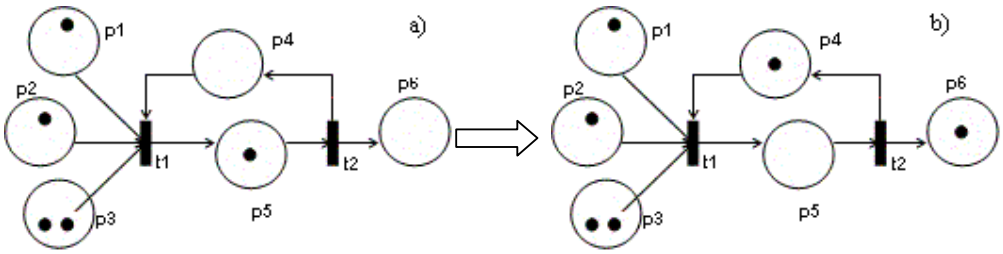


Figura 3. Disparo de uma transição

A estruturação de um sistema simples usando os conceitos convencionais de Redes de Petri pode ser bastante trivial. Porém, à medida que o problema torna-se mais complexo, a tarefa de gerar a Rede de Petri correspondente torna-se mais complexa. O uso de Rede de Petri Hierárquica é uma forma interessante de lidar com a complexidade da Rede de Petri, permitindo o encapsulamento de trechos da rede em transições refinadas. A Figura 4 ilustra uma Rede de Petri Hierárquica, onde as transições t_0 e t_1 são refinadas. A rede não-hierárquica correspondente pode ser obtida removendo a transição de alto nível e inserindo a sub-rede correspondente, ligando os lugares às transições.

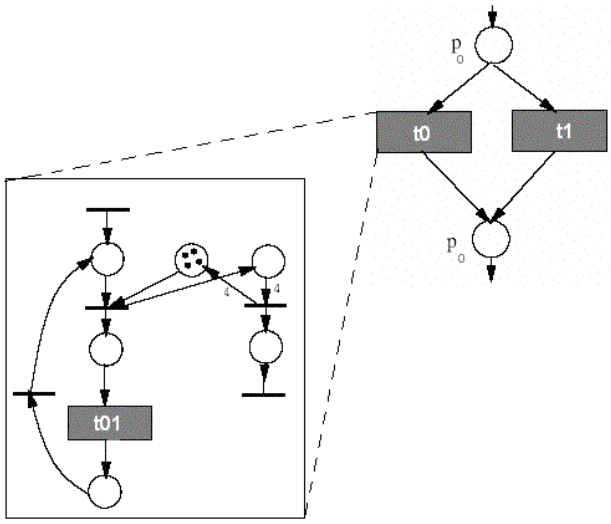


Figura 4. Rede de Petri Hierárquica

3.2 Propriedades formais

A análise de propriedades formais das Redes de Petri permite revelar diversas características do sistema modelado como, por exemplo, a possibilidade de assumir um estado final, ou a existência de estados inatingíveis ou irreversíveis. A seguir, temos propriedades comportamentais que podem ser encontradas nas Redes de Petri [34]:

- *Alcançabilidade* – Possibilidade de alcançar uma marcação a partir do disparo de um número finito de transições, dada uma marcação inicial; em geral, constrói-se grafos de alcançabilidade para facilitar o processo de análise, onde os nós são as marcações e os arcos simbolizam o disparo das transições;
- *Limitação* – Uma rede é tida como limitada se todos os seus lugares forem limitados, ou seja, quando, para cada lugar na rede, a marcação não ultrapassa um número *k* de marcas (lugar *k*-limitado).

- *Segurança* – É um caso particular de limitação. Uma rede é segura quando todos os seus lugares são 1-limitados.
- *Liveness* – Uma rede entra em uma situação de *deadlock* quando se encontra em um estado em que não é possível o disparo de qualquer transição. Uma transição t_i é *live* quando não há possibilidade de ocorrência de *deadlock* causada pelo seu disparo. A rede será *live* se todas as suas transições forem *live*.
- *Cobertura* – Uma marcação M' em uma rede N(R, M_0) é coberta se há uma marcação alcançável M'' a partir de M_0 tal que $M''(p_i) \geq M'(p_i)$ para todo lugar p_i da rede.
- *Persistência* – Para qualquer par de transições habilitadas da rede, o disparo de uma não desabilita a outra.
- *Reversibilidade* – Uma rede N(R, M_0) é reversível se há uma marcação M_k acessível a partir de Mi, para toda marcação Mi alcançável de M_0.
- *Justiça* – A questão da justiça pode ser dividida em duas etapas:
    - *Justiça limitada* – uma rede possui justiça limitada se todo par de transições (t_i, t_j) pertencente à rede possui justiça limitada, isto é, t_i e t_j possuem justiça limitada se é limitado o número de vezes que uma dispara enquanto a outra não dispara. A Figura 5a mostra uma rede onde, se a transição t_1 for disparada, a mesma ficará desabilitada e t_2 terá chances de disparar. O não determinismo de t_2 não garante que t_1 ficará habilitada com seu disparo, pois é possível que habilite a transição t_3 ao invés disto. Desta forma, sempre haverá chances de t_1 estar habilitada quando t_2 não está, porém , não há garantias.
    - *Justiça incondicional* – uma rede é incondicionalmente justa quando todas as transições da rede disparam um número infinito de vezes em seqüência. A Figura 5b mostra um exemplo de justiça incondicional. Nota-se que quando t_1 dispara, ela tornar-se-á desabilitada, enquanto t_2 ficará habilitada. Ambas possuem chances iguais de habilitação e se alternam nesta chance infinitamente.

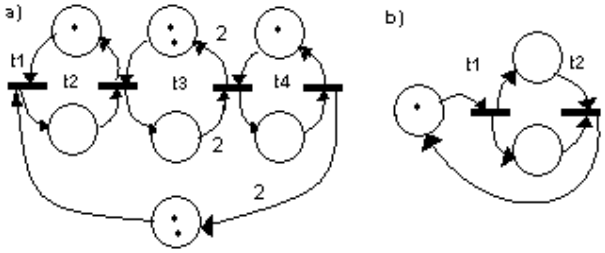


Figura 5. Rede com a) justiça limitada e b) justiça incondicional

Estas propriedades se caracterizam por serem condicionadas à marcação da rede. Um outro conjunto de propriedades, conhecidas como estruturais, independem da marcação e refletem questões relacionadas à estrutura dos modelos. Estas propriedades são apresentadas a seguir:

- *Limitação estrutural* – A rede será estruturalmente limitada se for limitada para qualquer marcação inicial. Em outras palavras, será limitada se, e somente se, existir um vetor de inteiros positivos *w* com dimensão #P (ou seja, um inteiro para cada lugar) tal que $wC \leq 0$ (sendo C a matriz de incidência).
- *Conservação* – O disparo de qualquer transição de uma rede conservativa não modifica o número total de marcas na rede.
- *Repetitividade* – A rede de Petri será repetitiva quando existir, a partir de uma marcação M_0, uma seqüência de transições *s* a partir da qual todas as transições de *s* disparam um número infinito de vezes.

- *Consistência* – A rede é consistente se uma marcação Mi é alcançada a partir de uma marcação M0, seguindo uma seqüência de transições *s*, onde todas as transições da rede disparam pelo menos uma vez em *s*.

3.3 Redes de Petri Temporizadas

Carl Adam Petri evitou introduzir o conceito de tempo em seu trabalho original, dadas as dificuldades que a temporização adicionaria na análise do comportamento das redes. Os primeiros trabalhos utilizando redes temporizadas são os de P.M. Merlin e D.J Farber [38] e J.D.Noe e G.J Nutt [41]. Podemos considerar três tipos de redes temporizadas (maiores detalhes podem ser obtidos em [37]):

1. redes com marcas temporizadas (onde cada marca em um lugar contém uma informação de tempo disponível para poder ser consumido);

2. redes com lugares temporizados (onde o lugar permite que suas marcas sejam consumidas após um dado tempo) ;

3. redes com transições temporizadas (onde o disparo ocorre depois de um tempo de atraso correspondente, a partir do momento em que ela se torna habilitada).

Ao longo deste trabalho daremos ênfase às redes com transições temporizadas. A representação gráfica de uma transição temporizada é, em geral, uma barra branca (o nome desta transição será iniciado com letra maiúscula), mais grossa que uma transição comum (Figura 6). A partir deste ponto chamaremos essa transição comum de *transição imediata*, por não depender de tempo, tendo precedência sobre as demais (o nome da transição será iniciado com letra minúscula).

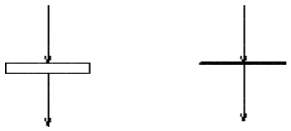


Figura 6. Tipos de transições: à esquerda, temporizada; à direita, imediata

A temporização torna a análise das Redes de Petri mais complexa, devido a mudanças nas regras de disparo das transições e à forma como a rede deve lidar com o não determinismo. A Figura 7 apresenta um exemplo de rede temporizada, onde há um conflito entre as transições *T1* e *T2,* pois o disparo de uma transição desabilita a outra. A transição que tiver um menor tempo associado vence a "disputa" pela marca. O problema, porém, consiste em decidir o que será feito do tempo restante na transição que foi desabilitada e nas demais transições temporizadas da rede que não estavam envolvidas no conflito. Três possíveis abordagens podem ser adotadas:

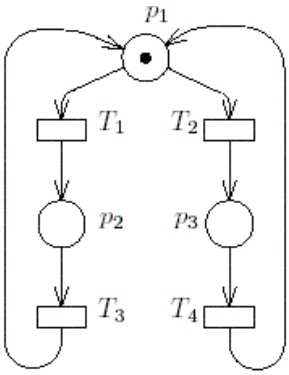


Figura 7. Uma Rede de Petri Temporizada

- *Resampling* – Após cada disparo todos os tempos, de TODAS as transições da rede são reiniciadas. Neste caso não há memória, e o "cronômetro" de cada transição será ligado assim que estiver habilitada.
- *Enabling memory* – Após cada disparo, os tempos das transições que foram desabilitadas são reiniciados. As outras transições mantêm seus valores. O fato de possuir memória torna a análise mais complexa, porém, mais próxima do sistema real modelado.
- *Age memory* – Após cada disparo, todos os tempos de TODAS as transições mantêm seus valores.

Um novo conceito que surge com as Redes Temporizadas é o de grau de habilitação, que determina o número de vezes que uma transição pode ser disparada após o tempo associado a ela se esgotar. Quando o grau de habilitação é maior que 1, podem ser destacadas três semânticas, que serão apresentadas a seguir e exemplificadas através da Figura 8, onde consideramos que a transição T1 está associada a uma duração de 3 unidades de tempo.

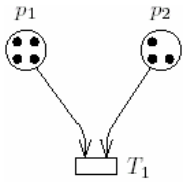


Figura 8. Grau de Habilitação

- *Single server* – Os conjuntos de marcas são consumidos sequencialmente, ou seja o tempo da transição apenas é reiniciado para um novo conjunto de marcas quando um disparo ocorre. O comportamento da transição neste caso independe do seu grau de habilitação. Por exemplo, para a Figura 8, temos a seqüência:
  - No tempo t = 0: T1 está habilitada e a primeira atividade é iniciada;
  - No tempo t = 3: a primeira atividade termina e a transição T1 é disparada, consumindo uma marca de p1 e uma de p2; o cronômetro é então reiniciado;
  - No tempo t = 6: T1 é dispara novamente e consome mais uma marca de p1 e de p2;
  - No tempo t = 9: T1 é disparada, consome uma marca de p1 e p2, e , por não mais haver marcas em p2, a transição torna-se desabilitada.
- *Infinite server* – O conjunto de marcas são consumidos assim que chegam aos seus respectivos lugares de entrada, ou seja, ao mesmo tempo. A especificação temporal associada à transição depende do grau de habilitação. Isso significa que se o grau de

habilitação de T1 for 2, significa que, após 3 unidades de tempo, a transição será disparada duas vezes, consumindo duas marcas de p1 e duas marcas de p2 de uma só vez.

- *Multiple server* – Semelhante ao anterior, porém tem um limite k de paralelismo. Se k tender ao infinito, então o disparo segue a semântica *infinite server*.

Nota-se um problema em relação a essas redes. A especificação determinística do tempo associado a uma transição em conflito impede que certos estados do sistema real sejam alcançados, uma vez que os conflitos são sempre resolvidos da mesma forma (considerando-se que a política de memória já foi definida). Para os sistemas que apresentam um maior número de estados e requerem avaliação mais detalhada e completa, a abordagem estocástica é uma alternativa mais interessante que as redes temporizadas determinísticas.

3.4 Redes de Petri Temporizadas Estocásticas

As Redes de Petri Estocásticas (*Stochastic Petri Nets* - SPNs) são extensões às Redes de Petri lugar/transição associando-se tempo às transições; este tempo é representado por uma variável aleatória com distribuição exponencial. Um método probabilístico transforma o não determinismo de uma rede não temporizada em probabilidade na rede temporizada, de modo que a teoria das Cadeias de Markov de Tempo Contínuo (CTMC, detalhes no Apêndice B) pode ser aplicada na avaliação de desempenho de sistemas descritos por uma SPN. Um estado em uma CTMC corresponde a uma marcação em uma SPN, e a transição de uma CTMC corresponde ao disparo de uma transição de uma SPN.

Formalmente, as SPNs são Redes de Petri com um conjunto adicional de *taxas* (uma para cada transição) que definem as *pdfs* das exponenciais. As SPNs seguem uma política de corrida para resolução de conflitos : quando uma transição temporizada torna-se habilitada, é atribuída uma amostra da distribuição correspondente ao seu cronômetro; a transição é disparada quando este cronômetro tem seu tempo esgotado; a solução de conflitos entre transições se dá com o disparo daquela que tiver o tempo mais curto para ser disparada. A característica de ausência de memória da distribuição exponencial permite a total imprevisibilidade no disparo de transições conflitantes, tornando o modelo mais fiel ao sistema real. Podemos definir uma SPN como uma 6-upla SPN=(P,T,I,O,W,M0) onde temos: P como o conjunto de lugares; T é o conjunto de transições; I é a matriz de mapeamento de lugares de entrada à transições; O é a matriz de mapeamento de transições à lugares de saída; M_0 é a marcação inicial da rede; e W é a função de mapeamento de taxas exponenciais às transições. Para compreender melhor os conceitos introduzidos, podemos analisar um exemplo de sistema paralelo modelado por uma SPN apresentado na Figura 9, onde existem as seguintes transições:

- TnewData – define a transição que modela a leitura de dados; se considerarmos que esta operação leva em média 10 unidades de tempo para ser executada, podemos dizer que sua taxa é 0.1 (pois para distribuições exponenciais a taxa λ é o inverso da média);
- Tstart –representa a ativação de dois processos paralelos;
- Tpar1 – define o primeiro processo, cuja taxa determina sua duração;
- Tpar2 – define o segundo processo, cuja taxa determina sua duração;
- Tsyn – realiza a sincronização entre processos;
- Tcheck – faz o controle dos resultados;
- Tio – determina saída dos dados para um dispositivo de I/O;
- Tok e Tko informam se a operação foi bem sucedida.

Entre Tok e Tko há um conflito, onde a vencedora será definida pela duração delas e pela probabilidade de falha do sistema.

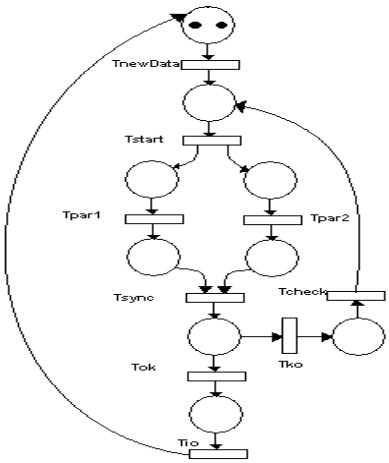


Figura 9. Sistema paralelo simples modelado em SPN

Se considerarmos que em 90% dos casos o sistema funciona corretamente (probabilidade de Tok disparar é de 90%) e que a duração média da verificação é de 2 unidades de tempo, a taxa é calculada da seguinte forma:

- $\lambda_{Tok} = 0.9/2 = 0.45$
- $\lambda_{Tko} = 0.1/2 = 0.05$

Uma vez que as SPNs conservam as propriedades formais da rede em questão, ainda é possível realizar análise de propriedades comportamentais e estruturais.

Uma nova classe de SPN pode ser obtida se acrescentarmos transições imediatas ao modelo: são as Redes de Petri Generalizadas Estocásticas (*Generalized Stochastic Petri Nets* - GSPNs). Se no exemplo anterior considerarmos instantâneos o tempo de disparo dos processos paralelos e o tempo de sincronização, temos a GSPN da Figura 10. As marcações do conjunto de alcançabilidade (conjunto das marcações alcançáveis a partir de uma inicial em uma rede) podem ser organizadas em dois subconjuntos: as marcações *vanishing*, onde há pelo menos uma transição imediata habilitada, e as *tangible*, que habilitam apenas transições temporizadas. Obviamente as marcações *vanishing* permanecem na rede num tempo infinitesimal, pelo fato de possuir transições que disparam instantaneamente (transições imediatas sempre têm prioridade sobre transições temporizadas).

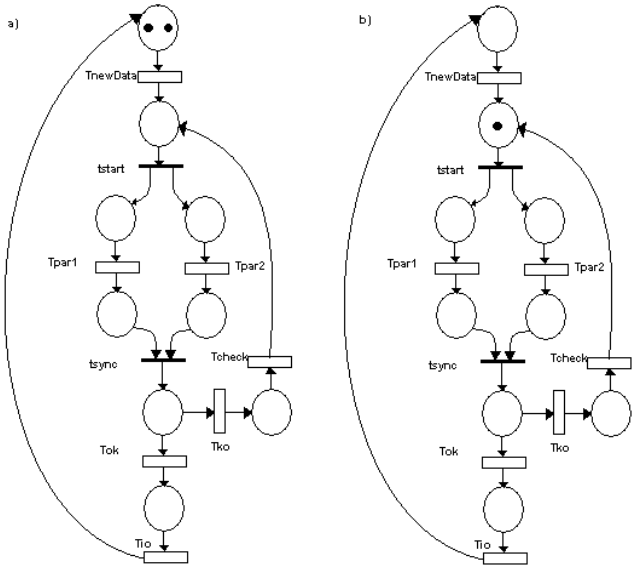


Figura 10. Exemplo de GSPN. a) marcação *tangible* b) marcação *vanishing*

Assim como as SPNs, as GSPNs possuem uma correspondência com as CTMCs, desde que a GSPN tenha um conjunto de alcançabilidade finito (detalhes sobre CTMCs no Apêndice B). A análise de desempenho de uma GSPN como a da Figura 11a pode ser dividida em três etapas [7]:

1. Geração de um grafo de alcançabilidade estendido, que contém as possíveis marcações alcançáveis a partir da inicial como nós do grafo e a informação estocástica nos arcos do grafo, gerando um processo semi-markoviano (na Figura 11b, as marcações *vanishing* estão em cinza). Temos na Figura 10b por exemplo, a marcação inicial m_0, onde há apenas uma marca em P_1 e nenhuma nos demais lugares. A partir dela, com taxa μ, a marcação *vanishing* m1 é alcançável (uma marca em P_2).
2. Transformação do processo semi-markoviano em CTMC, através da retirada das marcações *vanishing* e transições imediatas correspondentes, uma vez que são instantâneas. (ver Figura 11c)
3. Análise estacionária ou transiente da CTMC obtida, onde, em geral, são computadas [17]:
    a. Probabilidades estacionárias para cada marcação alcançável a partir da qual a matriz que indica o próximo estado pode ser obtida. Esse cálculo permite descobrir a probabilidade de uma transição estar habilitada.
    b. A *pdf* de cada lugar, calculada a partir de *a*. Isto representa a probabilidade estacionária de haver um numero *n* de marcas naquele lugar.
    c. O número esperado de marcas em um lugar, calculado a partir de *b*.
    d. *Throughput* de cada transição temporizada, indicando o fluxo médio de marcas através de uma transição. É calculado como o produto da taxa associada à transição pela probabilidade da mesma estar habilitada.
    e. Tempo médio de retorno a uma marcação, obtido através da Lei de Little: o número médio de clientes em uma fila é igual ao produto da taxa de sua chegada pelo tempo de espera. Para um conjunto de lugares, o número de marcas deve ser conservado e mudanças internas que alteram o número de marcas devem ser levadas em consideração. Podemos utilizar essa medida para calcular o tempo total de execução de um programa paralelo, como veremos no Capítulo 5.

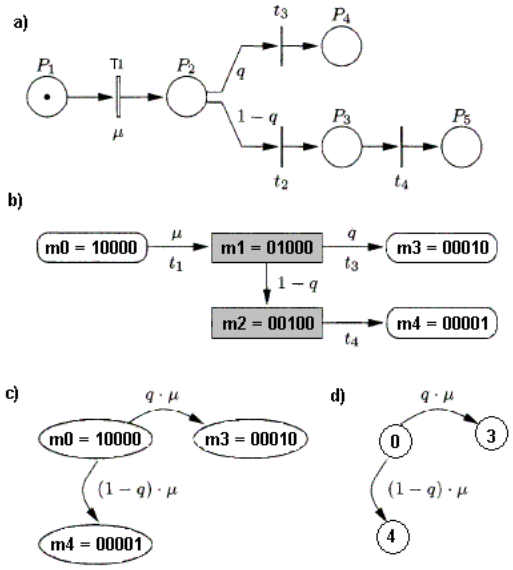


Figura 11. Transformação de uma GSPN em CTMC

As GSPNs são amplamente utilizadas na modelagem de diversos tipos de sistemas, desde sistemas de manufatura à redes sem fio [26]. Em [47] discute-se o uso de SPNs na análise de desempenho da composição de *Web Services*. No Capítulo 5, apresentaremos como estudo de caso a modelagem de programas paralelos em GSPNs, sob a perspectiva do modelo#, apresentado no capítulo a seguir.

Capítulo 4

O modelo # de componentes

O modelo # (leia-se *hash*) de componentes, o qual procura aliar os recursos disponíveis nas linguagens de alto nível à eficiência contida nas bibliotecas paralelas. É capaz de separar os interesses relacionados à especificação da computação dos interesses relacionados à coordenação e composição dos processos. Apresentaremos neste capítulo os principais conceitos relacionados ao modelo #, incluindo o esquema de tradução proposto em [15] para Redes de Petri, formalismo descrito no capítulo anterior, que permite analisar as propriedades formais de um programa paralelo #.

4.1 Modelo # : premissas e conceitos

A maioria dos programas paralelos são escritos utilizando linguagens seqüenciais, geralmente C ou FORTRAN, aliados a alguma biblioteca, como MPI, PVM, OpenMP e PThreads. Estas bibliotecas incluem rotinas de baixo nível para criação e gerenciamento de processos, sua comunicação e sincronização. As principais motivações para usá-las são a familiaridade que os programadores têm com estas linguagens e a garantia de obter eficiência. Além disso, a implementação dessas bibliotecas está disponível em diversas plataformas. Em compensação, esta abordagem oferece crescente dificuldade à medida que o projeto aumenta de tamanho e complexidade. Isso se dá ao fato de não fornecerem recursos adequados para a aplicação de artefatos de Engenharia de *Software*, tais como modularidade e *refactoring*. A vantagem do uso de linguagens de alto nível que possuam mecanismos de programação seqüencial e paralela é a garantia de um maior nível de abstração, com verificação de tipos e maior robustez. O maior desafio consiste em conciliar o desenvolvimento de abstrações de mais alto nível com o desenvolvimento de implementações eficientes.

O modelo de componentes #, voltado para programação paralela, busca conciliar alto nível de abstração, eficiência, generalidade e portabilidade. Estes requisitos têm sido procurados pela comunidade de processamento de alto desempenho (PAD) há mais de vinte anos. Tem suas origens na linguagem Haskell#, desenvolvida entre os anos de 1998 e 2003. Em 1999 [33] a linguagem Haskell# foi proposta como uma extensão paralela à linguagem funcional Haskell, onde as primitivas de comunicação eram implementadas sobre mônadas (recurso de programação imperativa que permite a existência de conceitos como E/S e estado em um programa funcional). As computações eram descritas por módulos funcionais. No ano seguinte, a noção de hierarquia de processos foi reforçada e foram publicados os primeiros resultados de pesquisas sobre mapeamento de programas Haskell# para Redes de Petri lugar/transição [32]. Um esquema de

tradução mais expressivo foi proposto em [15] e estendido em [13]. O modelo # foi proposto em [13], a partir do modelo de coordenação e composição de componentes de Haskell#. O modelo # atualmente busca atender às seguintes premissas [9]:

- *Ser um modelo de programação explícito e estático* – Sabe-se que o paralelismo implícito, contido em algumas linguagens, notadamente as do paradigma funcional (por possuírem transparência referencial), acarreta perdas na eficiência. Modelos explícitos e estáticos tendem a ser mais eficientes, além de constituírem uma suposição razoável para aplicações de PAD.
- *Possibilidade de Tradução para Redes de Petri* – Como visto no capítulo anterior, as Redes de Petri são uma poderosa ferramenta para análise formal e de desempenho. É desejável que programas # possam ser mapeados para Redes de Petri.
- *Hierarquia de processos* - A descrição das computações é tratada separadamente, de forma ortogonal, dos aspectos que determinam a coordenação do paralelismo e composição de componentes.

O modelo # é orientado a componentes. Um programa # é um componente # constituído pela sobreposição de outros componentes #, compostos hierarquicamente, cada qual representando um interesse de aplicação. O interesse "coordenação", por exemplo refere-se a coordenação do paralelismo da aplicação. Podemos ter ainda interesses "depuração", "entrada/saída", "mapeamento de processos a processadores" e "computações". A seguir, as principais característica da programação #:

- Componentes podem ser compostos a partir de outros componentes, pela unificação de suas unidades formando novas unidades.
- Uma unidade possui uma *interface* que descreve o comportamento dinâmico do sistema, através da ativação de *portas*, conectadas através de componentes *canais* [9].
- O modelo # também oferece suporte a esqueletos topológicos, que são componentes compostos parametrizáveis e reutilizáveis. Sua implementação pode variar de acordo com as características da arquitetura alvo [14]. Por exemplo, podemos ter um esqueleto "broadcast", onde um possível parâmetro é o número de processadores para os quais a mensagem em questão será transmitida (exemplos de chamada a esqueletos serão encontrados no estudo de caso, no Capítulo 5).

Nas próximas subseções estes conceitos serão aprofundados mais adequadamente. Uma semântica formal baseada em Teoria das Categorias [8] e um ambiente de programação vem sendo implementado, para dar apoio à análise e ao desenvolvimento de aplicações. A ferramenta HPE (# Programming Environment) vem sendo construída como uma extensão ao *framework* de edição gráfica GEF da plataforma Eclipse [9]. Este ambiente já vem sendo projetado para permitir a sua independência de Haskell e a possibilidade da adoção de linguagens imperativas como linguagens *host*, como C e Java.

4.1.1 O componente Canal

Sob a perspectiva de processos, um canal é uma abstração da rede de comunicação que interliga os processadores nela envolvidos, provendo um meio de comunicação entre os processos que nestes encontram-se hospedados [2]. A ligação entre canais e processos, pode ser feita através de *portas,* cuja ativação efetiva a operação de envio ou recebimento de mensagens. Um exemplo pode ser visto na Figura 12, onde as unidades U1 e U2 correspondem a dois processos.

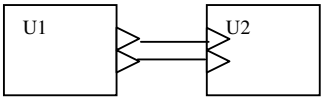


Figura 12. Comunicação entre processos através de portas e canais

Pela perspectiva de componentes, as unidades são compostas por *fatias*, que são unidades menores associadas a outros componentes em um nível abaixo da hierarquia de componentes (unidades observáveis de componentes aninhados). Temos, como exemplo, o componente canal Ch1 na Figura 13, com suas unidades S1, encapsulando operações de envio e observável pela unidade U1; já a unidade R1, que encapsula operações de recebimento, é observável pela unidade U2. Analogamente temos o componente Ch2, cujas unidades S2 e R2 são observáveis pela unidade U1 e U2, respectivamente. O conceito de *ativação de porta* dá lugar ao de *ativação de fatia*, por possuir um significado mais amplo que simplesmente um envio ou recebimento. Desta forma, as fatias de uma unidade devem ser ativadas para execução do comportamento descrito pela sua interface. Na base da hierarquia de componentes, admite-se ativação simples, como o uso de primitivas de envio e recebimento, assim como as portas. A unidade em seu nível mais alto da hierarquia (aquela que não é fatia de nenhuma outra unidade) pode ser considerada um processo (neste exemplo, as unidades U1 e U2 são consideradas processos, enquanto S1, R1,S2 e R2 são apenas fatias de processos)**.**

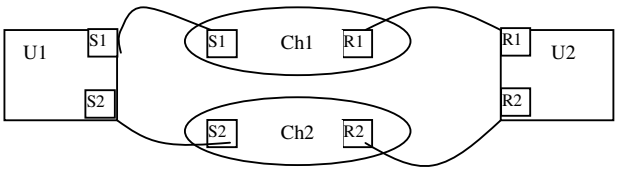


Figura 13. Componente canal Ch1 e Ch2

4.1.2 Compondo fatias

Se desejarmos observar o modelo # por uma perspectiva de processos, devemos considerar que um interesse de aplicação é definido por um conjunto de fatias, cada uma descrevendo o papel do processo com respeito a este interesse. Na Figura 14a, por exemplo, as fatias em branco simbolizam o interesse de E/S. Já as fatias cinza-claras, cinza-escuras e pretas simbolizam interesses de coordenação de processos. Elas descrevem operações de comunicação com topologias diferentes, das quais nem todos os processadores participarão. Outros exemplos de possíveis interesses que podem envolver dois ou mais processos são: computações, estruturas de dados, alocação de processos, operações de comunicação coletiva, entre outros. Os interesses sobrepostos formam os processos (Figura 14b). Vejamos o processo que se encontra no meio da figura: ele é composto por uma fatia branca de E/S e por uma fatia cinza-escura e por uma cinza-clara, correspondentes às duas operações de comunicação que este processo deve realizar. Devemos notar que este processo não possui a fatia preta, uma vez que o mesmo não faz parte de sua topologia.

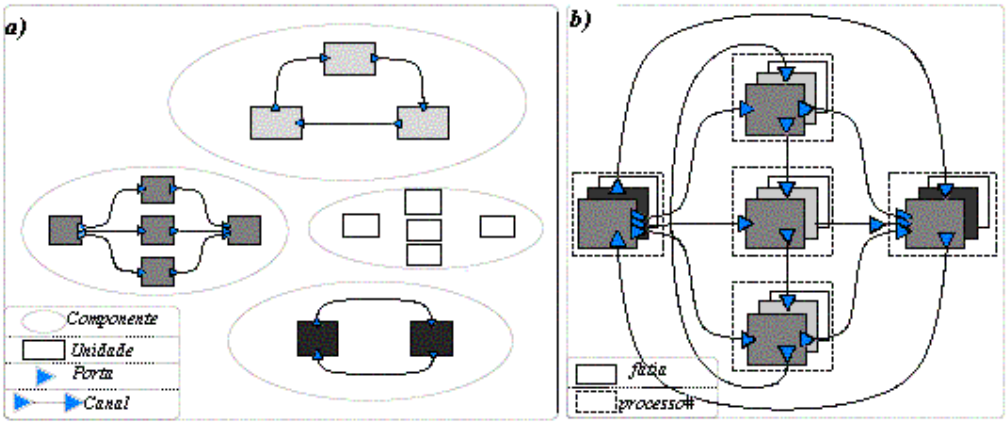


Figura 14. Programação # sob a perspectiva de processos: a) Programa separado por interesses b) Interesses se associam formando processos

Sob a perspectiva de componentes, temos que um interesse é materializado por um componente # que o implementa. O componente # forma-se a partir de unidades (fatias do processo), e/ou de outros componentes, hierarquicamente. Obviamente, para o componente mais externo, (o mais alto na hierarquia) o seu interesse é a aplicação em si. Temos na Figura 15 a programação # sob a perspectiva de componentes. Cada elipse corresponde a um interesse. Abaixo das elipses temos o interesse "aplicação" (elipse maior), que faz uso dos demais componentes através de suas unidades.

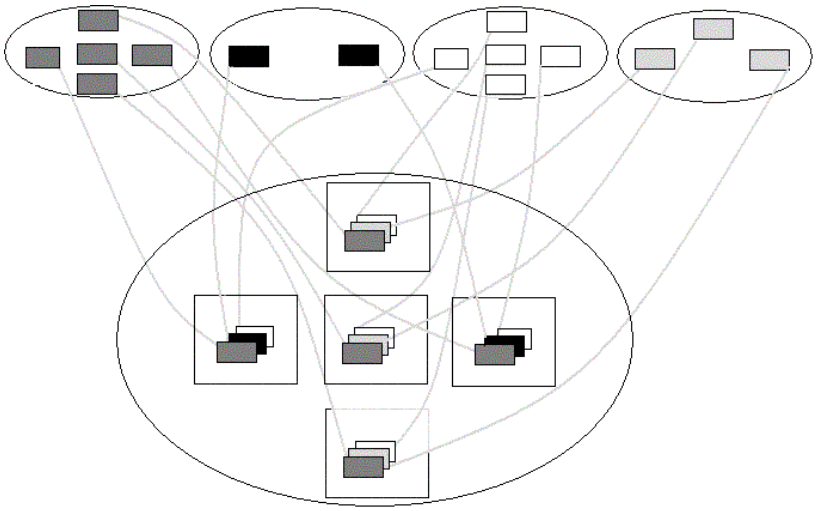


Figura 15. Programação # sob a perspectiva de componentes

4.2 Programação no modelo

A ferramenta HPE é um *framework* de programação orientado a componentes baseado no modelo #, sendo, portanto, um *framework #*. A idéia é que o usuário monte componentes utilizando a interface gráfica do ambiente, através de seu módulo *Front-End*, sem necessidade de conhecer qualquer tipo de linguagem intermediária. O *Front-End* de um *framework #* é capaz de gerar um modelo de componentes baseado no modelo #. Sob o ponto de vista orientado a objetos, este

modelo é representado por um conjunto de classes que implementam o conjunto de interfaces especificadas na arquitetura do modelo #. O módulo *Back-End* deverá sintetizar um programa paralelo a partir deste modelo de componentes, o qual incorpora informações sobre todos os interesses relacionados à aplicação, incluindo aqueles que caracterizam o ambiente de execução, notadamente a tecnologia de programação paralela sobre a qual o *Back-End* está definido. Isso permite que várias tecnologias possam ser vistas sob a perspectiva de componentes, desde que haja um *Back-End* desenvolvido para seu suporte (Figura 16).

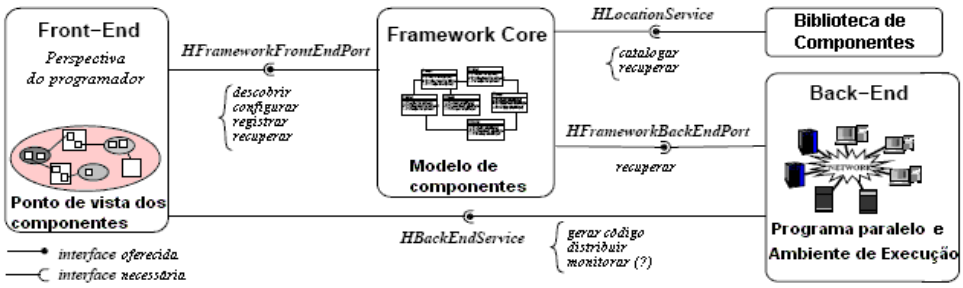


Figura 16. *Front-end* e *back-end* do *framework* #. Fonte: [9].

Um diagrama de classes simplificado da arquitetura de *frameworks* do modelo # pode ser visto na Figura 17. A arquitetura completa do framework # pode ser vista no Apêndice C. Uma instância da classe *Component* pode possuir outras instâncias de *Component*, composta por sobreposição. Possui ainda uma ou mais unidades, ou seja, instâncias da classe *Unit*, as quais representam fatias de processos que descrevem o papel desta com respeito ao interesse do componente. Cada unidade é tipada por uma instância da classe *Interface,* a qual define um protocolo de coordenação da unidade (classe *Protocol*). Em particular, o protocolo define em que ordem e como as ações serão executadas. Cada *framework* pode adotar um formalismo para descrever protocolos. HPE, por exemplo, adota expressões regulares sincronizadas, formalismo que possui equivalência com Redes de Petri rotuladas. Neste caso, o protocolo pode possuir um conjunto de semáforos (classe *Semaphore*) e uma instância da classe *Action*, denotando uma ação.

Uma ação pode possuir um dos seguintes tipos

- Repetitivas – Modela ações repetitivas, de modo que a repetição pode ser limitada por um contador (primitiva **repeat ... counter**) ou por uma condição (primitiva **repeat ... until**).
- Combinadores – Ações são combinadas de forma seqüencial (primitiva **seq**), paralela (primitiva **par**) ou por escolha (primitiva **alt**).
- Primitivas – Denotam ações básicas. Podem ser operações sobre semáforos, como as primitivas **signal** e **wait**, que equivalem às classes *ActionPrimitiveSignal* e *ActionPrimitiveWait*. Podem também ser a ativação de uma fatia de unidade (primitiva **do <unit>**). Por exemplo, a ativação de uma fatia que representa uma unidade de um componente canal denota uma ativação simples de porta de comunicação, especificando o tipo de envio, com as subclasses *BufferedChannelSender*, *BufferedChannelReceiver*, *SynchronousChannelSender*, *SynchronousChannelReceiver*, *ReadyChannelSender* e *ReadyChannelReceiver*.

Este *framework* se encontra em fase de desenvolvimento, de modo que representaremos os exemplos com a sintaxe textual da linguagem de configuração HCL (# Configuration Language), proposta para configuração de processos na linguagem Haskell#.

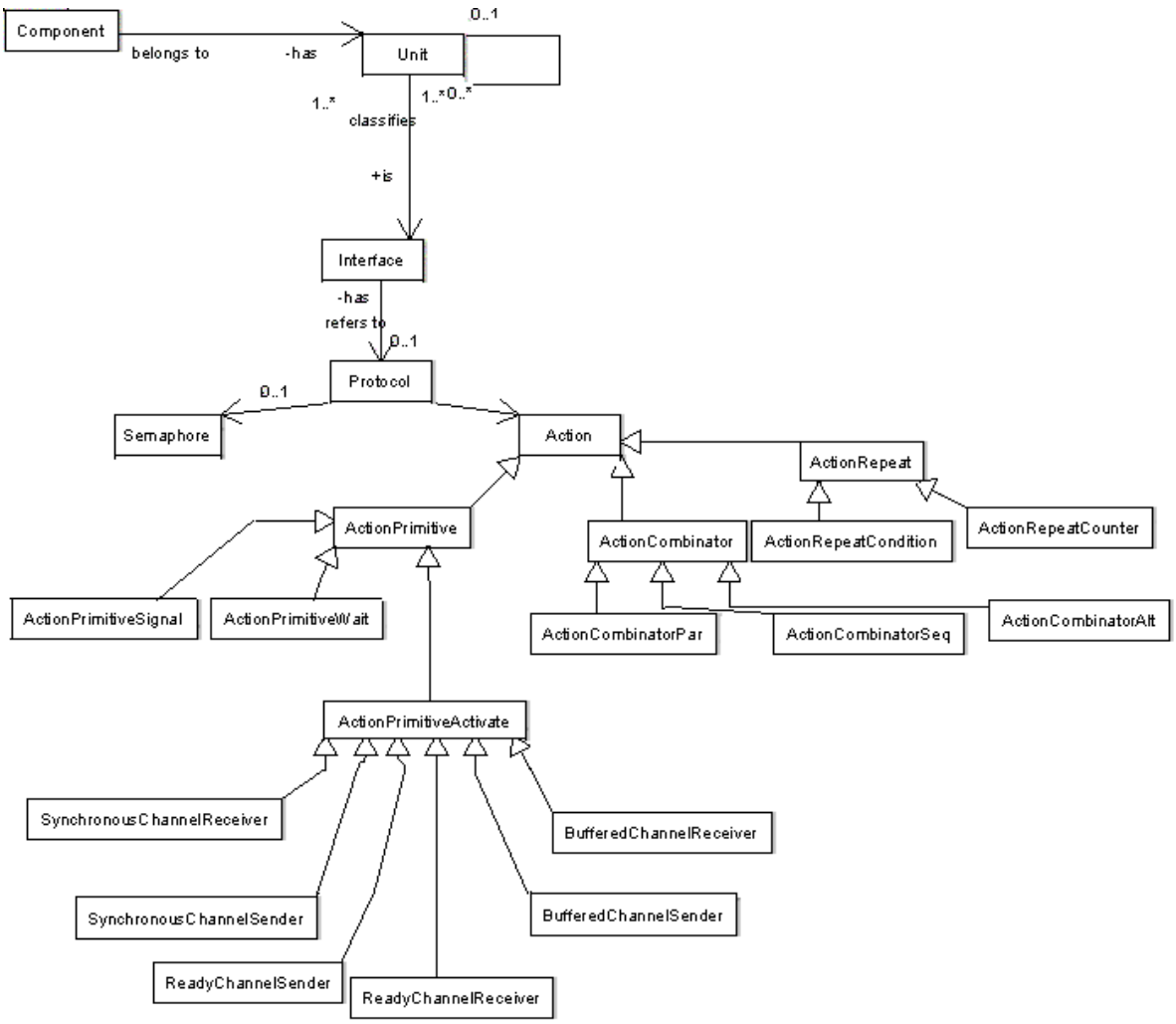


Figura 17. Diagrama de classes simplificado de um componente #

4.3 Programas # e Redes de Petri

Em [15] é descrito um esquema de tradução de especificações HCL para Redes de Petri lugar/transição. Cada unidade deve ser traduzida para uma Rede de Petri distinta, a partir do protocolo que a descreve. Posteriormente, as redes geradas são ligadas pelas transições que modelam os canais. As unidades mais externas são vistas como processos que executam em paralelo (Figura 18).

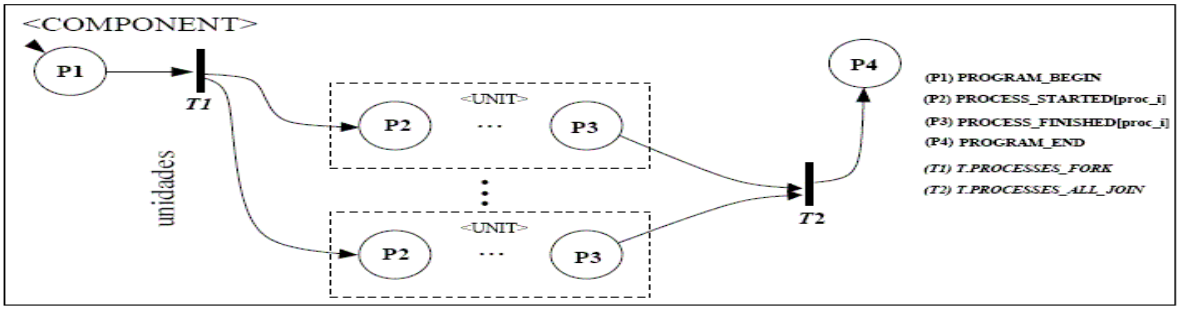


Figura 18. Rede de Petri modelando unidades. Fonte: [15].

A seguir, teremos as Redes de Petri equivalentes às ações combinadoras (seqüenciais, paralelas e alternativas), repetitivas e primitivas (ativação de portas ou unidades). Também são descritas a equivalência para, canais e semáforos

- Ações seqüenciais: são modeladas simplesmente por redes ligadas por transições em seqüência (Figura 19).

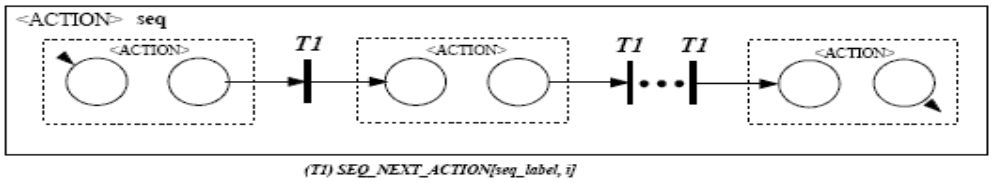


Figura 19. Rede de Petri modelando ações seqüenciais. Fonte: [15].

- Ações paralelas: transições de *fork* e *join* são inseridas permitindo que ações sejam executadas simultaneamente.

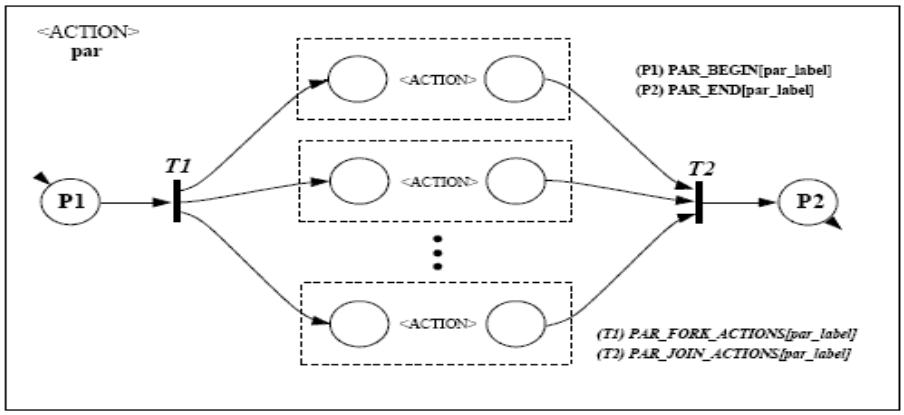


Figura 20. Rede de Petri modelando ações paralelas. Fonte: [15].

- Ações alternativas: a escolha é modelada com uso de um lugar ligado a várias transições, formando uma situação de conflito na Rede de Petri, onde o disparo de uma transição desabilita o disparo das demais (Figura 21).

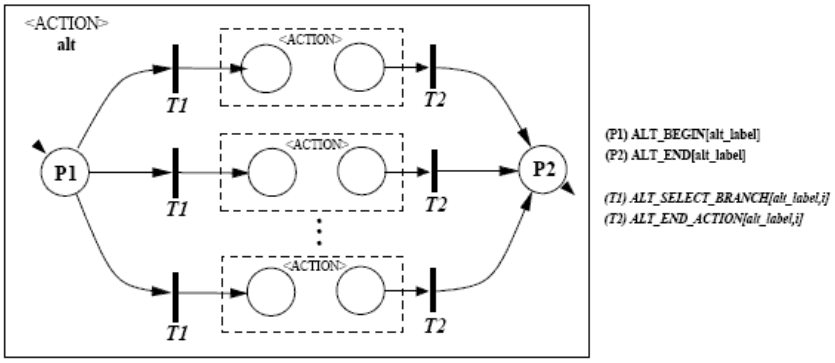


Figura 21. Rede de Petri modelando ações não determinísticas. Fonte: [15].

- Repetição limitada por contador: na Figura 22, o peso do arco que liga T1 a P2 possui peso *n*, indicando que em P2 surgirão *n* marcas assim que T1 disparar. P2 armazena a contagem de iterações restantes na execução da ação. P3 controla o disparo de T2, permitindo que, depois de iniciado o laço, a mesma só seja disparada quando uma iteração for completada, ou seja, quando T3 for habilitada. P4 armazena as iterações que já foram executadas, de modo que a transição T4, que marca o fim do laço, só fica habilitada quando P4 tiver as *n* marcas.

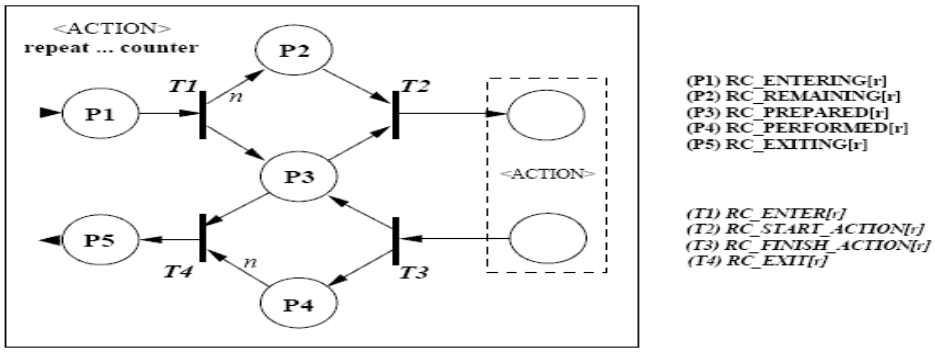


Figura 22. Rede de Petri modelando repetição com contador. Fonte: [15].

- Repetição condicional: a repetição ocorre com certa condição booleana ou por guardas associadas a alguma unidade. A verificação da condição é feita por uma situação de não-determinismo (Figura 23).

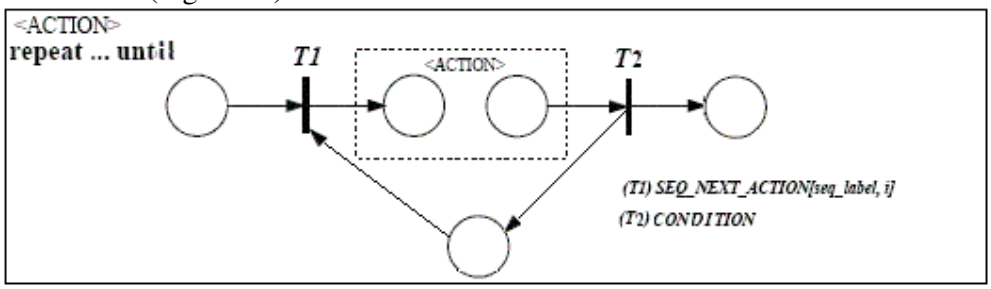


Figura 23. Rede de Petri modelando repetição condicional

- Ativação de portas/unidades: na Figura 24, a região cercada pela elipse indica a unidade que será ativada pelo disparo da transição T1; o fim desta ativação habilita a transição T2,

devolvendo o controle da rede ao programa que a ativou. No caso mais simples a região cercada pela elipse é simplesmente um canal.

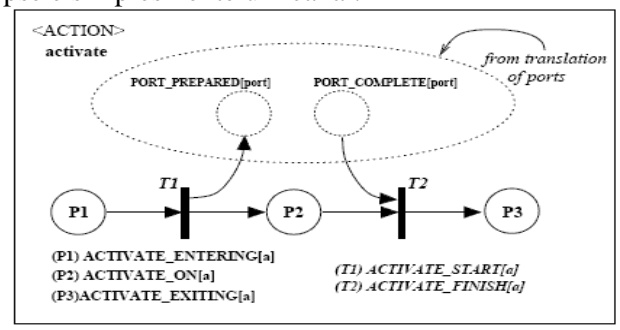


Figura 24. Rede de Petri modelando ativação de portas/unidades. Fonte: [15].

- Canais: são modelados como transições simples, que sincronizam as unidades. Na Figura 25 temos P1 como um lugar pertencente a certa unidade (no caso temos duas unidades ligadas por canais). No modo síncrono, basta a transição T1 para simbolizar o compartilhamento do canal, pois para sua habilitação as duas unidades deve ter marcas em P1. No modo assíncrono bufferizado, tem-se os lugares P3 e P4 que modelam o *buffer*, permitindo que uma das unidades inicie sua operação de comunicação antes da outra. No modo *ready*, a comunicação é modelada de forma assíncrona, porém sem garantias de funcionamento, por não possuir *buffer*; uma análise nesta rede mostra que ela é passível de *deadlock*.

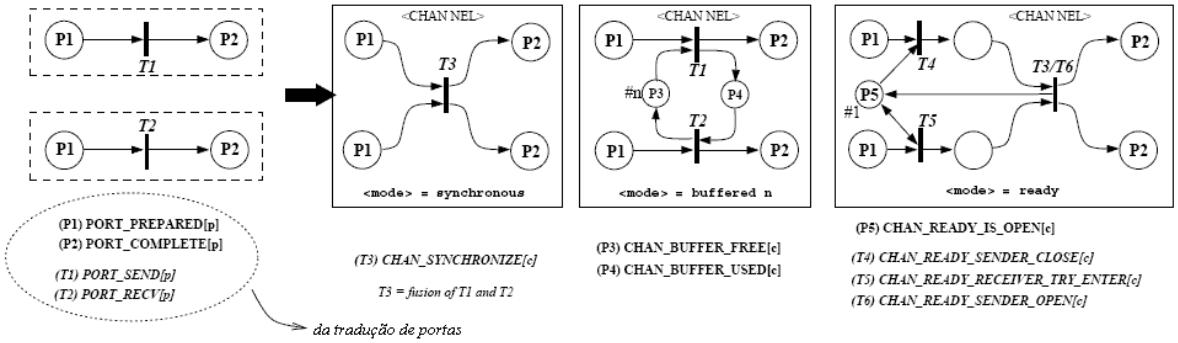


Figura 25. Rede de Petri modelando canais síncronos, bufferizados e canais *ready*. Fonte: [15].

- Semáforos: o lugar P3 da Figura 26 modela um semáforo, sendo o número de marcas correspondentes ao valor do semáforo.

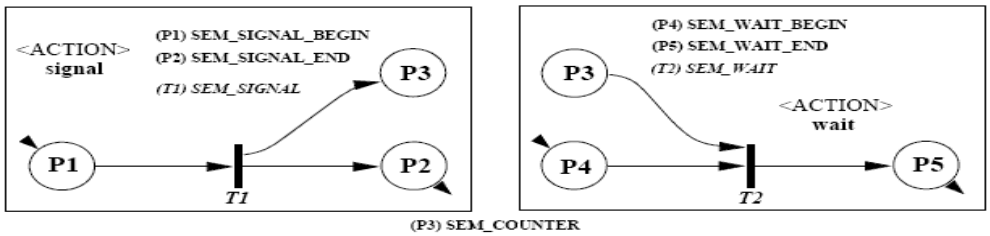


Figura 26. Rede de Petri modelando primitivas de semáforos. Fonte: [15].

Esta possibilidade de tradução para Redes de Petri permite que propriedades formais como aquelas descritas no Capítulo 3 sobre os programas # sejam analisadas.

Capítulo 5

Estudo de caso: Modelando e analisando programas

Neste capítulo apresentaremos uma metodologia para avaliação de desempenho de programas paralelos sob a perspectiva do modelo # de componentes utilizando as Redes de Petri Temporizadas, estendendo o esquema de tradução para Redes de Petri lugar/transição, proposto em [15]. Utilizaremos como estudo de caso um subconjunto dos NAS Parallel Benchmarks, descritos no Capítulo 2 caracterizando as variáveis aleatórias que modelam o comportamento dos programas, obtendo assim dados necessários para a modelagem e análise em GSPN. Este estudo motiva a construção de ferramentas de apoio à análise de desempenho de programas paralelos, inclusive a construção de *back-ends* que traduzam configurações # em especificações de GSPNs de forma semi-automática.

5.1 Metodologia de avaliação de desempenho

A metodologia proposta consiste na retirada de medições reais da aplicação escolhida como estudo de caso para traçar um perfil de desempenho de suas operações de comunicação. A partir das amostras obtidas são tiradas médias, desvios-padrão e coeficientes de variação, com auxílio da ferramenta Microsoft® Office Excel 2003. A partir deste momento, podemos inferir as variáveis aleatórias correspondentes a essas amostras e mapeá-las à transição ou seqüência de transições (caso se trate de uma distribuição *Phase-type*) da GSPN equivalente ao programa. Segue-se a metodologia de tradução de especificações HCL em Redes de Petri lugar/transição descrita em [15], adicionalmente atribuindo taxas exponenciais às transições temporizadas. Esta GSPN deve ser *live* e limitada para possibilitar a análise estacionária [17]. Portanto, sempre que necessário, será acrescentada uma transição ao modelo para que seja permitido o retorno da rede à sua marcação inicial [47] como temos na Figura 27.

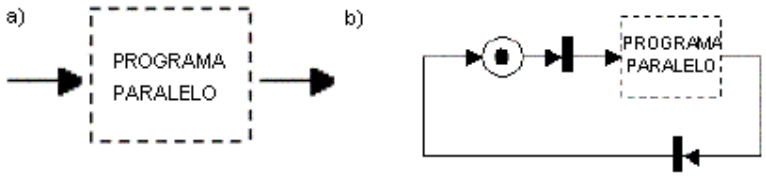


Figura 27. Acréscimo de um modelo de *loop* ao modelo do programa paralelo

Uma vez obtida a GSPN desejada, é possível realizar avaliação de desempenho, utilizando para tal ferramentas de modelagem e análise de Redes de Petri. Optamos por utilizar a ferramenta GreatSPN em sua versão 2.0 [44]. É uma ferramenta visual e textual para fins acadêmicos que oferece suporte à análise de Redes de Petri Temporizadas e de algumas das suas extensões. Para nosso experimento, temos interesse em medir o tempo total de execução do programa que será obtido a partir do *throughput* medido em uma transição a ser escolhida. Esta transição não deve estar envolvida em ramificações ou *loops* da rede para evitar interpretações errôneas. O tempo médio de execução é tido como o inverso do *throughput* encontrado. Desta forma, podemos calcular medidas importantes em processamento de alto desempenho, como *speedup*, uma vez que já temos os modelos seqüenciais e paralelos do programa, com a grande vantagem de não depender de recursos de hardware, como um supercomputador real, nem sempre disponível.

Para realizar as medições, utilizamos o *cluster* do Departamento de Computação da Universidade Federal do Ceará, um Infocluster® constituído por 28 nós Intel Xeon, cada qual com 4 processadores (2GHz de freqüência e 1GB de memória RAM) e interligados por meio de uma rede *Fast Ethernet.* O sistema operacional é o Linux, na distribuição *Red Hat 7.3*. Para compilação do código do NPB foram utilizados os compiladores gcc 2.96 (C) e g77 versão 0.5.26 (Fortran). A implementação da biblioteca MPI adotada foi a MPICH versão 1.2.4 e a versão do NPB utilizada foi a 2.3. Por questões de simplicidade apenas dois tipos de carga foram experimentadas e dois kernels foram escolhidos.

5.2 Os NAS Parallel Benchmarks

O pacote NPB (*NAS Parallel Benchmarks*) é um *benchmark* composto por um conjunto de kernels e aplicações simuladas, proposto no início dos anos 90 pelos cientistas do programa NAS (*Numerical Aerodynamic Simulation*) da NASA [3]. A versão do NPB, utilizada é composta de oito programas paralelos escritos em C e FORTRAN com a biblioteca de passagem de mensagens MPI: EP (*Embarassingly Parallel*), IS (*Integer Sort*), CG (*Conjugate Gradient*), FT (*Fast Fourier Transform*), MG (*Multigrid*), LU (*LU solver*), SP (*Pentadiagonal Solver*), BT (*Block Tridiagonal*). A carga computacional utilizada é variável de acordo com uma classe escolhida em tempo de compilação (níveis de A a D). O fato de serem muito bem documentados e de terem o respaldo da comunidade científica, nos incentivou a utilizá-los como estudo de caso deste trabalho.

Os NPB já foram anteriormente utilizados em pesquisas relacionadas ao modelo #. Versões Haskell# foram implementadas para realização de medições simples, para comparação de desempenho em relação a suas versões originais [12]. Estes programas também tiveram seus tempos de comunicação e computação caracterizados probabilisticamente em [46] para oferecer suporte ao desenvolvimento de uma metodologia de avaliação de desempenho de programas #. Notou-se que a distribuição dos tempos de comunicação e computação não se aproximam exatamente de uma exponencial, como seria o desejado (pois facilitaria a modelagem do programa em GSPN). Porém, foram encontradas boas aproximações para distribuições *Phase-type*, que, por sua vez, são representadas como uma combinação de transições exponenciais, viabilizando a modelagem destes programas através das GSPNs. Na maioria dos casos os tempos de computação e comunicação encontravam melhores aproximações com as distribuições Gama, Erlang, Lognormal e Weibull. Neste trabalho, trataremos os *kernels* IS e EP, variando nas classes A e B, destacando os principais trechos dos seus códigos para que possamos realizar uma análise mais detalhada e precisa dos mesmos.

5.2.1 O kernel EP

O *kernel* EP é o *benchmark* mais simples do NPB em termos de comunicação. É tido como "embaraçosamente paralelo" por possuir um paralelismo inerente e que não exige a sincronização de processos durante a computação em si. É um programa que gera pares de desvios gaussianos, explorando os limites superiores de desempenho de arquitetura em operações de ponto flutuante. Utiliza apenas três chamadas a *MPI_AllReduce* ao fim da computação, com os vetores `sx`, `sy` e `q`, respectivamente. Uma operação *MPI_AllReduce* faz com que todos os processos agreguem certo valor contido em cada um deles através de uma operação de redução, como uma soma ou função de máximo ou de mínimo. Na Figura 28 temos a configuração HCL correspondente ao EP.

```
interface IEP where          -- declaração da interface
ports: IAllReduce @ sx #  -- esqueleto IAllReduce é instanciado em sx
       IAllReduce @ sy #  -- esqueleto IAllReduce é instanciado em sy
       IAllReduce @ q     -- esqueleto IAllReduce é instanciado em q
protocol: seq {do sx; do sy; do q} -- descrição das ações (seqüência)
```

Figura 28. Trecho de código HCL descrevendo a interface do interesse de coordenação de processos para o programa EP

A Rede de Petri lugar/transição é extremamente simples, pois se trata apenas de três ativações de unidades (Figura 29).

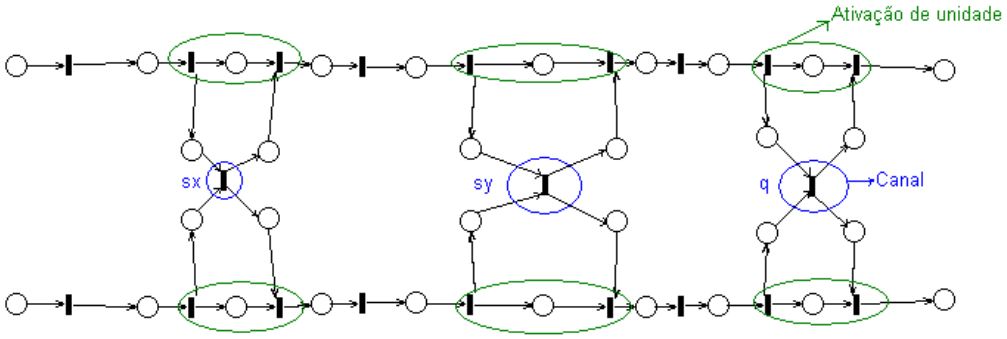


Figura 29. Rede de Petri do interesse de coordenação do kernel EP (2 processos)

5.2.2 O kernel IS

O kernel IS é um programa de ordenação de inteiros que utiliza o algoritmo de *bucket sort* paralelo. A classe do *benchmark* define o tamanho do *array* a ser ordenado. Ao longo do seu *loop* principal, o IS realiza as seguintes operações de comunicação:

- Um *array* chamado `bucket_size` é passado como parâmetro para uma rotina *MPI_AllReduce.*
- Os *arrays* `send_count` e `key_buf` são utilizados em chamadas *MPI_AllToAll,* onde todos os processos realizam *broadcast* de um determinado valor.
- Operações *MPI_Send* e *MPI_Recv* enviam informações da ordenação, cada processador para seu vizinho.

A seguir, o interesse "coordenação" do IS é representado em HCL (Figura 30). Estes esqueletos são instanciados na interface e chamados no protocolo. O esqueleto *IRShift* simboliza a operação *send/receive* que ocorre no IS. A cláusula `repeat … until` denota um loop cuja condição é a presença de dados a serem enviados/recebidos nas operações de `bs`, `kb` e `sc` (referente às operações envolvendo o array `key_buf`, o array `bucket_size` e `send_count`). As unidades `vf` e `pv` servem para verificação dos resultados finais do algoritmo.

```
interface IIS where          -- declaração da interface
ports: IAllReduce @ bs #  -- IAllReduce é instanciado em bs(bucket_size)
       IAllToAllv @ kb #  -- IAllReduce é instanciado em kb(key_buf)
       IRShift    @ vf #  -- esqueleto IRShift é instanciado em vf
       IAllToAll  @ sc #  -- esqueleto IAllToAll é instanciado em sc
       IReduce    @ pv    -- esqueleto IReduce é instanciado em pv
-- descrição das ações
protocol: seq { do bs; do kb;
 -- repete até que bs e kb não possuam mais dados para enviar
                repeat seq {do bs;
                            do sc;
                            do kb}
                until <bs&kb>;
                do vf; do pv }
```

Figura 30. Trecho de código HCL descrevendo a interface do interesse "coordenação" do programa IS

A seguir teremos a Rede de Petri equivalente ao programa IS com dois processos, de acordo com o esquema de tradução descrito em [15] e apresentado no Capítulo 4. Podemos identificar o laço e a ativação das unidades bs e kb, compartilhadas entre os processos dentro do loop. A ativação das unidades sc, vf e pv foram omitidas deste modelo para melhor compreensão do leitor.

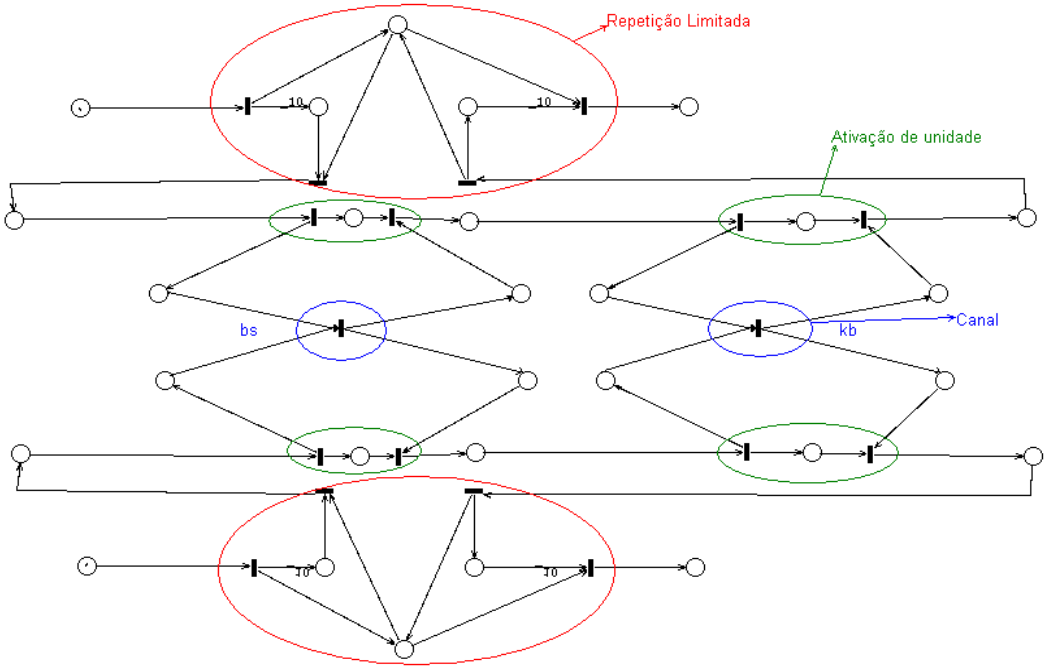


Figura 31. Rede de Petri do interesse de coordenação do kernel IS (2 processos)

5.3 Caracterização de Desempenho

Neste trabalho seguimos duas abordagens para escolher a distribuição de probabilidade mais adequada para caracterizar as variáveis aleatórias que correspondem aos tempos de comunicação: Análise do Coeficiente de Variação e Ajuste da Distribuição Encontrada (ver Figura 32a).

a) **Análise do coeficiente de variação** – O valor do coeficiente de variação (CV, detalhado no Apêndice A) pode nos levar a tirar algumas conclusões a respeito dos dados medidos [35]. Cada caso a seguir e seus respectivos parâmetros estão resumidos na Tabela 2:
   - Caso CV=1 – Aproximação da variável para distribuição exponencial. O inverso da média é usado como taxa da transição exponencial equivalente (Figura 32b).
   - Caso CV≤1 – Se o inverso do CV for um número inteiro, a aproximação para a distribuição Erlang é a mais adequada. Ela possuirá γ fases em seqüência de transições com taxa λ cada (Figura 32c). Um caso generalizado onde 0.5≤CV≤1 que permite que mais de uma taxa seja utilizada é a distribuição hipo-exponencial. Para efeitos de simplificação, consideraremos duas taxas diferentes: λ1 para a primeira fase e λ2 para as demais fases Figura 32e).
   - Caso CV>1 – Aproximação para distribuição hiper-exponencial, que corresponde a transições exponenciais em paralelo. Um modelo simplificado possui como parâmetros λh, como a taxa da transição exponencial e r1 e r2 como peso de duas transições imediatas. A transição associada a r1 fica paralela à transição associada a r2 e a transição exponencial, que, por sua vez, estão em seqüência (Figura 32d).

Tabela 2. Inferência de distribuição pelo coeficiente de variação

Distribuição	Coeficiente de variação	Parâmetros
Exponencial	CV=1	$\lambda = \frac{1}{\mu_D}$, μ_D σ_D são a média e desvio da amostra
Erlang	CV≤1 , e 1/CV diferente de 1 e inteiro	$\gamma = \left(\frac{\mu_D}{\sigma_D}\right)^2$ $\lambda = \frac{\gamma}{\mu_D}$
Hipo-exponencial	0.5≤CV≤1	$\lambda_1 = \frac{(\gamma+1)}{\mu_D \pm \sqrt{\gamma(\gamma+1)\sigma_D^2 - \gamma\mu_D^2}}$ $\lambda_2 = \frac{(\gamma+1)}{\gamma\mu_D \mp \sqrt{\gamma(\gamma+1)\sigma_D^2 - \gamma\mu_D^2}}$ $\left(\frac{\mu_D}{\sigma_D}\right)^2 - 1 \le \gamma < \left(\frac{\mu_D}{\sigma_D}\right)^2$
Hiper-exponencial	CV>1	$\lambda_h = \frac{2\mu_D}{(\mu_D^2 + \sigma_D^2)}$ $r_1 = \frac{2\mu_D^2}{(\mu_D^2 + \sigma_D^2)}$ $r_2 = 1 - r_1$

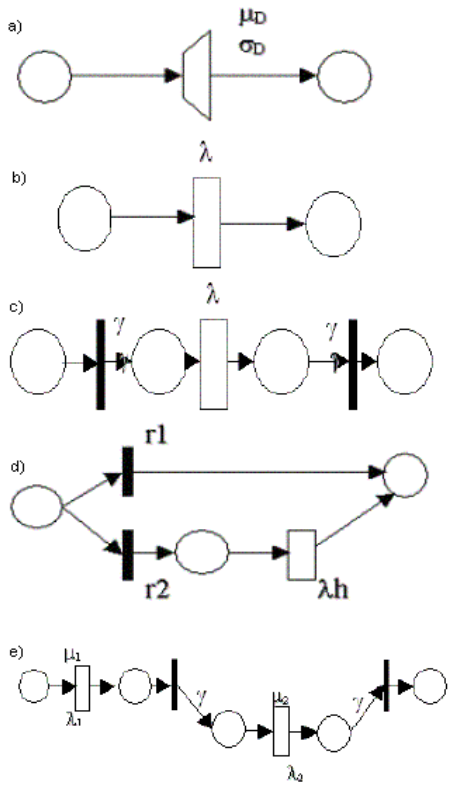


Figura 32. Representação de distribuições *Phase-type* em GSPN

b) Ajuste da distribuição encontrada – A amostra é passada a uma ferramenta capaz de fazer ajustes para distribuições. Em nosso caso, escolhemos o *Input Analyzer* da ferramenta ARENA. O *Input Analyzer* traça um histograma, podendo ajustar para as seguintes distribuições: Beta, Empirical, Erlang, Exponencial, Gama, Johnson, Lognormal, Normal, Poisson, Triangular, Uniforme e Weibull. Possui também uma opção "Fit all" que indica a distribuição que melhor se enquadra a massa de dados, ou seja, a que possuir menor erro quadrado. Em nosso caso, temos maior interesse em fazer ajustes para as distribuições Exponencial e Erlang por ser uma distribuição PH, permitindo a modelagem em GSPN.

5.4 Análise da Rede de Petri obtida

Os valores dos tempos de comunicação do kernel EP apresentam uma dispersão favorável à modelagem com GSPNs. Cremos que, por se tratarem operações simples, de granularidade fina e por não estarem associadas a um laço, as execuções do programa mostraram-se sensíveis às variações da rede de comunicação. Consequentemente o coeficiente de variação aproximou-se de 1.0, como podemos ver na Tabela 3. Devido a dificuldades no acesso ao cluster, foi possível apenas obter o tempo de comunicação total do EP. Para fins de modelagem consideraremos o tempo de execução de cada uma das três operações de comunicação (sx, sy e q) como o tempo total de comunicação dividido por três. Utilizaremos como exemplo o caso de 8 processadores executando uma carga de classe A.

Tabela 3. Medições obtidas no EP (Classe A, para 8 processadores)

Medições/operações de comunicação	**Operações sx, sy ou q**
Média	0.190797233
Desvio Padrão	0.155624114
Variância	0.024218865
Coeficiente de Variação	0.815651837
limite inferior do γ	0.503108612
limite superior do γ	1.503108612
γ escolhido	1
λ1	6.655629734
λ2	24.66183048
Ajuste via ARENA	ERLA(0.0954, 2) EXPO(0.191)
Erro quadrado de ajuste obtido no ARENA	0.098532 0.154796

É importante notar que o fato do CV estar entre 0.5 e 1.0, indica que podemos utilizar a distribuição hipo-exponencial como aproximação. Temos o modelo GSPN equivalente e o resultado da análise estacionária na Figura 33. Neste caso, o uso de GSPN foi fundamental para obter valores próximos à medição real.

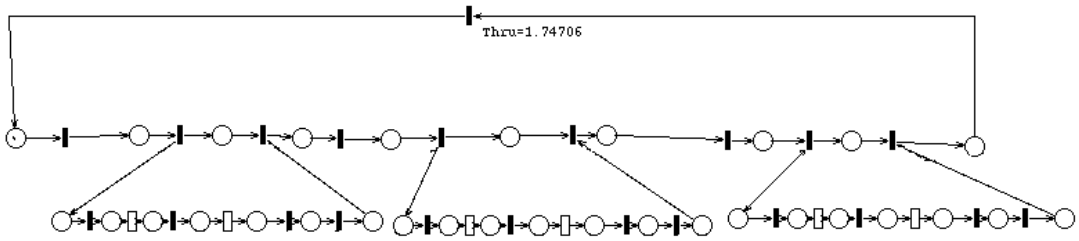


Figura 33. Representação do interesse de coordenação do EP com GSPN com aproximação para hipo-exponencial (1 processo de 8)

Os resultados do EP referentes às aproximações através de ajustes para as distribuições Erlang (Figura 34) e Exponencial (Figura 35) com o auxílio da ferramenta ARENA mostraram-se bastante positivos, com um *throughput* bastante semelhante ao caso hipo-exponencial. Maiores detalhes na Tabela 4.

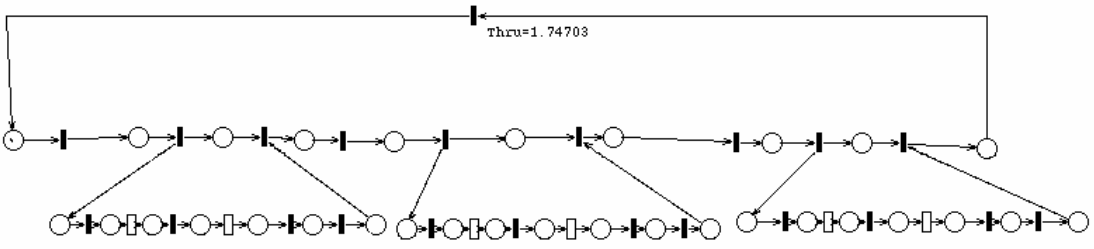


Figura 34. Representação do interesse de coordenação do EP com GSPN com aproximação para erlang de acordo com a ferramenta ARENA (1 processo de 8)

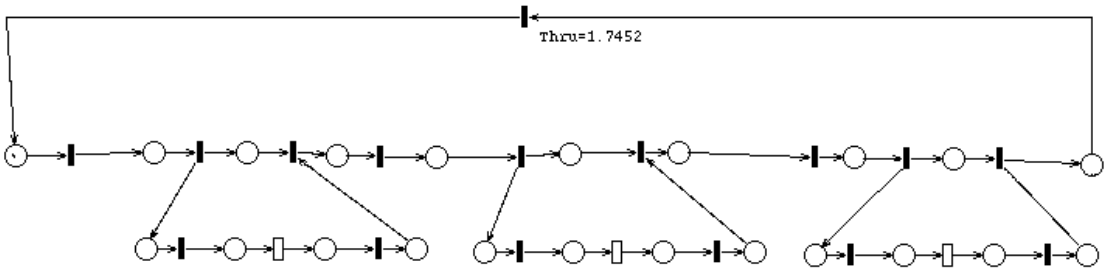


Figura 35. Representação do interesse de coordenação do EP com GSPN com aproximação para exponencial de acordo com a ferramenta ARENA (1 processo de 8)

Tabela 4. Comparação dos tempos obtidos no EP

	Medição	Hipo-exponencial	Erlang	Exponencial
Tempo de execução	0.572391699	1/1.74706=0.572390186	1/1.74703=0.572400016	1/1.7452=0.573000229

Com estes experimentos podemos constatar que, para este estudo de caso, o uso da metodologia adotada foi válido.

No kernel IS, destacamos como principais operações de comunicação as operações identificadas por bs e kb, descritas na Seção 5.2.2. Procedeu-se à retirada de uma amostra significativa dos tempos de execução de cada uma dessas operações, levando-nos à Tabela 5. Por questões de simplicidade, mostraremos apenas os dados referentes ao caso que utiliza 8 processadores com a carga de classe A.

Tabela 5. Medições obtidas no IS (Classe A, para 8 processadores)

Medições/operações de comunicação	**Operação bs**	**Operação kb**
Média	0.21384	0.372238
Desvio Padrão	0.065339	0.058972
Variância	0.046674	0.047842
Coeficiente de Variação	0.305552	0.158427
limite inferior do γ	9.710986	38.84214
limite superior do γ	10.71099	39.84214
γ escolhido	10	39
$\lambda 1$	33.85452	77.13647
$\lambda 2$	5.425878	2.783392
Ajuste via ARENA	0.72 + EXPO(1.42)	2.61 + EXPO(1.11)
Erro quadrado de ajuste obtido no ARENA	0.070280	0.186826

Apesar de termos obtido valores válidos para os parâmetros γ, $\lambda 1$ e $\lambda 2$, notamos um CV muito baixo, menor que 0.5 em todos os casos. Isto nos indica que uma aproximação da amostra para a distribuição hipo-exponencial é possível, porém, inadequada, uma vez que os dados apresentaram uma variação muito pequena. Uma possível solução é a modelagem com Redes de Petri Temporizadas Determinísticas, que neste caso específico apresenta resultados mais exatos que o uso de GSPN uma vez que a baixa dispersão dos dados nos leva a aproximação para valores constantes. Na Figura 36 temos a Rede de Petri Temporizada equivalente, com o

resultado do tempo total. Temos o *throughput* igual à 0.170626, o que nos dá um tempo médio de aproximadamente 5.86 segundos. É importante notar que o resultado obtido é muito semelhante à soma dos tempos médios de bs e kb, multiplicados por 10 (pois bs e kb são executados em um laço de tamanho 10 no IS).

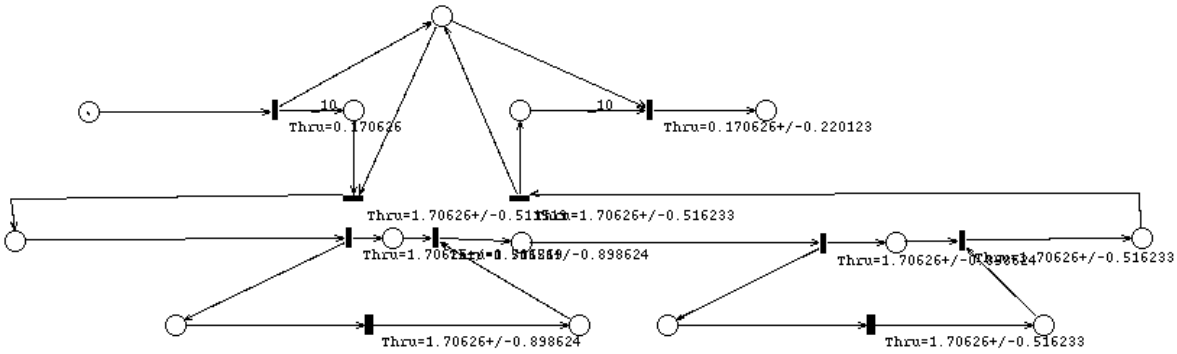


Figura 36. Representação do interesse de coordenação do IS com Rede de Petri Temporizada Determinística (1 processo de 8)

No caso do IS, notamos que não seria conveniente utilizar a abordagem de ajuste de distribuição, pois as expressões encontradas pelo ARENA na maioria dos casos incluíam um *offset* à exponencial. Isso impossibilita nossa modelagem na ferramenta GreatSPN, que não permite que transições determinísticas e com tempos exponenciais sejam incluídos no mesmo modelo.

Obviamente, os resultados variam de acordo com a natureza da aplicação (carga computacional, uso da rede de comunicação), além da arquitetura a qual a mesma está sendo executada. Em casos onde a dispersão dos dados é pequena, podendo ser caracterizados por um valor constante, soluções mais simples como o uso de Redes de Petri Temporizadas Determinísticas são mais adequadas, uma vez que a modelagem com variáveis aleatórias exponenciais não caracterizaria estes dados corretamente. É possível crer que em ambientes mais desacoplados, em g*rids* , onde há maior variação em tempos de comunicação, a solução de modelagem com GSPNs seja mais adequada que em ambientes mais fortemente acoplados e homogêneos, como os *clusters*.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste capítulo discutiremos a importância dos resultados obtidos para o surgimento de novos questionamentos e soluções para avaliação de desempenho de programas paralelos.

6.1 Conclusões

Avaliação de desempenho de programas paralelos não é uma tarefa simples: os resultados a serem obtidos dependem fortemente de fatores e níveis relacionados à carga de trabalho, à arquitetura à qual o programa está associado e à topologia dos processadores. Desta forma, faz-se necessário um estudo do problema em questão de modo a verificar qual abordagem de avaliação de desempenho é a mais adequada. Medições podem ser utilizadas apenas quando o sistema a ser avaliado está fisicamente disponível, porém, oferece resultados mais exatos. O uso de simulações é recomendado quando se deseja obter resultados mais detalhados de sistemas que não estejam disponíveis ou cuja execução real é proibitiva, possuindo, entretanto, a desvantagem de oferecer maiores dificuldades na interpretação dos resultados. A modelagem analítica, por sua vez, simplifica a tarefa de avaliação dos resultados, uma vez que trata o sistema como um conjunto de variáveis aleatórias cuja distribuição fornece informações ao analista a respeito do sistema. Sua principal desvantagem é a complexidade dos cálculos matemáticos envolvidos (podendo não ser matematicamente tratável) e da quantidade de estados a serem observados.

Neste trabalho demos destaque à modelagem analítica utilizando Redes de Petri Estocásticas, que se mostrou um formalismo válido para modelagem do interesse de comunicação de programas #. Porém, à medida que o coeficiente de variação torna-se muito pequeno, deve-se verificar se o uso de Redes de Petri Temporizadas Determinísticas não apresenta resultados melhores que a abordagem com GSPN. Uma vez que constatamos a validade do uso da metodologia na modelagem de programas paralelos, temos a extensão do esquema de tradução de especificações HCL para Redes de Petri lugar/transição de [15] para GSPNs com fins de avaliação de desempenho, com o acréscimo de transições ou combinações de transições, cujo tempo associado a cada uma delas é uma variável aleatória com distribuição exponencial.

Uma dificuldade encontrada no desenvolvimento deste trabalho (inclusive prevista como risco no projeto da monografia) foi a utilização do *cluster* remoto, que em certas ocasiões teve seu acesso impedido por problemas técnicos. Felizmente, porém, isto não impediu a coleta da maioria dos dados, mas exigiu que fossem realizadas adaptações, como no caso do EP, onde não foi possível obter os tempos individuais das operações de comunicação, sendo necessário realizar

inferências. Este fato evidencia ainda mais a vantagem do uso de modelagem analítica e simulação sobre medições simples.

6.2 Trabalhos futuros

A validação dos modelos em Redes de Petri Temporizadas nos levou a dar mais um passo no desenvolvimento de programas paralelos com o ambiente de programação # citado na Seção 4.2. Essa metodologia pode ser incorporada ao HPE com a construção de um *back-end* que traduza as classes HPE em um tipo abstrato que descreva a Rede de Petri equivalente. Uma vez que o usuário possa oferecer dados de desempenho para a modelagem de um interesse, como tempo médio e desvio-padrão, será possível realizar uma análise do coeficiente de variação pelo próprio ambiente, transformando a Rede de Petri em uma rede temporizada, com a aproximação mais adequada ao interesse modelado.

Uma abordagem não utilizada neste trabalho, mas que pode ser estudada em outros é a integração do HPE com ferramentas como o EMpht [43], que procura realizar ajustes de amostras para distribuições *Phase-type*. Outra ferramenta que poderia ser incorporada ao HPE é o MPIBench [25], para realizar medições detalhadas de operações MPI, uma vez que o HPE possuirá um *back-end* para passagem de mensagens.

Bibliografia

[1] AMDAHL, G.M. *Validity of Single-processor Approach to Achieving Large Scale Computing Capabilities*. Proceeding of the American Federation of Information Processing Societies,30:483-485, 1967.

[2] ANDREWS,G. Foundations *of Multithreaded, Parallel, and Distributed Programming.* Reading, MA: Addison-Wesley, 1999.

[3] BAYLEY, D. H., HARRIS, T., SHAPIR, W., WIJNGAART, R. van der, WOO, A., YARROW, M. *The NAS Parallel Benchmarks 2.0*, Tech. Rep. NAS-95-020, NASA Ames Research Center, 1995.

[4] BAXTER, R. *A Complete History of the Grid (abridged).* Universidade de Endiburgo, 2003.

[5] Beowulf.org - http://www.beowulf.org/ Acessado em Maio de 2006.

[6] Berkeley NOW Project - http://now.cs.berkeley.edu/ Acessado em Maio de 2006.

[7] BOLCH, G., GREINER, S., MEER, H., TRIVEDI, K.S. *Queuing Networks and Markov Chains – Modeling and Performance Evaluation with Computer Science Applications.* John Wiley and Sons inc, 1998.

[8] CARVALHO JUNIOR, F. H., LINS, R. D. *A Categorical Characterization for the Compositional Features of the Hash Component Model*. ACM Software Engineering Notes, New York, NY, USA, v. 31, n. 2, 2006.

[9] CARVALHO JUNIOR, F. H. ; LINS, R. D., CORRÊA, R. C., ARAÚJO, G. A., SANTIAGO, C. F. *Design and Implementation of an Environment for Component-Based Parallel Programming.* In: Proceedings of VECPAR'2006. Rio de Janeiro, 2006.

[10] CARVALHO JUNIOR, F.H., LINS, R.D. *Using Aspects for Supporting Procedural Modules in # Programming*. In Proceedings of the EURO-PAR 2005. Lisboa, Portugal, 2005.

[11] CARVALHO JUNIOR, F.H., LINS, R.D. *The # Model for Parallel Programming: From Processes to Components with Insignificant Performance Overheads.* In: Workshop on Components and Frameworks for High Performance Computing (CompFrame), 2005.

[12] CARVALHO JUNIOR, F. H., LINS, R. D., QUENTAL, N. C. *On the Implementation of SPMD Applications Using Haskell #.* In: The 15th Symposium on Computer Architecture and High Performance Computing, São Paulo, 2003.

[13] CARVALHO JUNIOR, F.H. *Programação paralela eficiente e de alto nível sobre arquiteturas distribuídas.* 291 f. Tese (Doutorado em Ciência da Computação) - Centro de Informática, Universidade Federal de Pernambuco, Recife,2003.

[14] CARVALHO JUNIOR, F. H., LINS, R. D. *Topological Skeletons in Haskell #.* In: Proceedings of the 2003 IEEE International Distributed and Parallel Symposium, Nice, França., 2003.

[15] CARVALHO JUNIOR, F.H., LINS, R.D., LIMA, R.M.F. *Translating Haskell# into Petri Nets*. Proceedings of the 5th International Meeting on High Performance Computing for Computational Science (VECPAR'2002), Porto, Portugal, 2002.

[16] CULLER,D. ,KARP, R., PATTERSON,D. ,SAHAY,A. ,SCHAUSER,K.E. ,SANTOS,E., SUBRUAMONIAN,R., EICKEN,T. *LogP: Towards a realistic model of parallel computation*. *In* Proceedings of the 5th ACM SIGPLAN Symposium on the Principles and Practices of Parallel Programming p.1-12, 1993.

[17] DESROCHERS, A., AL-JAAR, R.Y. *Performance evaluation of automated manufacturing systems using Generalized Stochastic Petri Nets*. IEEE Transactions on Robotics and automation vol. 6 dec, 1990.

[18] DONGARRA, J. *Trends In High Performance Computing and the Grid.* Talk at University of Utah, SCI Institute seminar. Outubro, 2003.

[19] DUNCAN, R. *A Survey of Parallel Computer Architectures.* IEEE Computer, 21(9):5–16, Fevereiro 1990.

[20] FLYNN, M.J. *Some Computer Organizations and Their Effectiveness.* IEEE Transactions on Computing, v. 21, n. 9, p. 948-960. 1972.

[21] FORTUNE, S. WYLLIE,J. *Parallelism in random access machines*. *In* Proceedings of the 10th ACM Symposium of Theory of Computing, p. 114-118, 1978.

[22] FOSTER, I. *Designing and building parallel programs –Concepts and tools for parallel software engineering* . Addison Wesley , 1994.

[23] FREITAS, P.J. *Introdução à Modelagem e Simulação de Sistemas*. Visual Books- Santa Catarina, 2001.

[24] GROVE, D.A. *Performance Modeling of Messaging Passing Parallel Programs*, PhD Thesis, University of Adelaide, 2003.

[25] GROVE, D., CODDINGTON, P. P*recise MPI Performance Measurement Using MPIBench.* Technical Report DHCP-104. Department of Computer Science. Adelaide University, Australia, 2001.

[26] HEINDL, A. R. GERMAN. *Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets.* Performance evaluation : an international journal. Elsevier, 2001.

[27] HENNESSY, J.L., PATTERSON, D.A. *Organização e projeto de computadores – A interface Hardware/Software*. 2ª. Edição LTC. Rio de Janeiro, 2000.

[28] HOARE, C.A.R. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs., NJ, 1985.

[29] ISHAY, E. *Fitting Phase-Type Distributions to Data from a Telephone Call Center*, tese de Mestrado, Israel Institute of Technology, 2002.

[30] JAIN, R. *The art of computer systems – Performance Analysis: Techniques for experimental design measurement, simulation and modeling.* Wiley Computer Publishing. John Wiley & Sons Inc, New York, N.Y, 1991.

[31] LINS, F.A.A. *Análise do Custo de Comunicação de Programas Paralelos sobre Redes de Computadores* Trabalho de Conclusão de Curso. Departamento de Sistemas Computacionais - Universidade de Pernambuco – Recife PE, 2004.

[32] LIMA, R. M. F., LINS, R. D. *Translating HCL Programs into Petri Nets.* In: 13th Brazilian Symposium on Software Engineering, 2000, p. 147-162. João Pessoa, 2000.

[33] LIMA, R. M. F., CARVALHO JÚNIOR, F. H., LINS, R. D. *Haskell#: A Message Passing Extension to Haskell.* In: 3rd Latin American Conference on Functional Programming. Recife, 1999.

[34] MACIEL, P.R.M., LINS, R.D., CUNHA, P.F. *Introdução às Redes de Petri e aplicações*. X Escola de Computação - Campinas – SP, 1996.

[35] MACIEL, P.R.M. *Modelagem para Avaliação de Desempenho e Confiabilidade* Centro de Informática – UFPE. Disponível em www.cin.ufpe.br/~prmm/slides/SPN.pps. Acessado em maio de 2006.

[36] MARKOV, A. *Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain*. Reimpresso no Appendix B of: R. Howard. Dynamic Probabilistic Systems, volume 1: Markov Chains. John Wiley and Sons, 1971.

[37] MARSAN, M.A., BALBO, G., CONTE, G., DONATELLI, S., FRANCESCHINIS,G. *Modeling with generalized stochastic Petri Nets.* Università degli studi di Torino – Dipartimento di informática, 1995.

[38] MERLIN, P. M., FARBER, D. J. *Recoverability of communication protocols: Implications of a theoretical study.* IEEE Transactions on Communications, 24(9):1036–1043. Setembro, 1976.

[39] Microsaint - http://www.adeptscience.co.uk/products/mathsim/microsaint/. Acessado em Maio de 2006.

[40] MILNER,R.. *A calculus of communicating systems*. In: Lecture Notes in Computer Science(80) Springer-Verlag, Nova Iorque, 1980.

[41] NOE, J. D., NUTT, G. J. *Macro e-nets representation of parallel systems.* IEEE Transactions on Computers, 31(9):718–727, August 1973.

[42] ÓLAFSSON, S. *IE313-Introduction to stochastic analysis.* Iowa State University, 2000.

[43] OLSSON, M. *The EMpht-programme*. Chalmers University of Technology e Göteborg University. Junho, 1998.

[44] Performance Evaluation Group. *GreatSPN User Manual (version 2.0.2).* Dipartimento di Informatica, Università di Torino.

[45] PVM: Parallel Virtual Machine - http://www.csm.ornl.gov/pvm/ Acessado em Maio de 2006.

[46] QUENTAL, N.C., CARVALHO JUNIOR, F. H., LIMA, R.M.F. *Caracterização de Desempenho de Programas SPMD Utilizando Modelos Probabilísticos*. In: XV Workshop em Sistemas de Computação de Alto Desempenho – WSCAD 2005, Rio de Janeiro, 2005.

[47] SILVA, A. N., LINS, F. A. A., JÚNIOR, J. C., ROSA, N. S. ; QUENTAL, N. C., MACIEL, P. R. M. *Avaliação de Desempenho da Composição de Web Services Usando Redes de Petri.* In: 24o. Simpósio Brasileiro de Redes de Computadores - SBRC 2006, Curitiba, 2006.

[48] SINCICH, T., LEVINE, D.M. e STEPHAN, D. *Practical Statistics by example using Microsoft Excel.* Prentice-Hall inc. Nova Jersey, 1999.

[49] STROHMAIER, E.DONGARRA, J., MEUER H.W., SIMON, H.D, *Recent Trends in the Marketplace of High Performance Computing.* In Parallel Computing, 2005.

[50] The Message Passing Interface Standard - http://www-unix.mcs.anl.gov/mpi/ Acessado em Maio de 2006.

[51] THE VINT GROUP. *The Network Simulator - ns-2*. Homepage. Disponível na Internet Http://www.isi.edu/nsnam/ns. Acesso em Maio de 2006.

[52] TRIVEDI, K., SUNH, H. *Stochastic Petri Nets and Their Applications to Performance Analysis of Computer Networks*. Invited paper at International Conference on "Operational Research for a Better Tomorrow", Nova Delhi, 1998

[53] VALIANT, L.G. *A Bridging Model for Parallel Computation* Communications of the ACM, 33(8):103-111. 1990.

[54] WATROUS, R. L. *GRADSIM: a connectionist network simulator using gradient optimization techniques.* Report, Siemens Corporate Research, Inc., Princeton,New Jersey, 1993

Apêndice A

Variáveis aleatórias

Uma variável aleatória é uma entidade que representa uma ocorrência cuja probabilidade pode ser representada por uma função de mapeamento [42]. Um exemplo é a probabilidade de um time vencer um jogo em um campeonato: os eventos podem ser a vitória ou a derrota do mesmo. Se for necessário saber a respeito da duração de um evento, a variável deve admitir uma aproximação. Caso a aproximação não seja desejável, é possível considerar a variável em um intervalo contínuo, e neste caso não é possível atribuir uma probabilidade a cada valor, sendo necessário determinar sua função de distribuição acumulada (*cumulative distribution function - cdf*). A cdf de uma variável aleatória X é

$$Fx(a) = P\{X \leq a\} \qquad \text{(Eq. 6)}$$

É considerada cumulativa por considerar valores de X inferiores a *a*, acumulando as probabilidades anteriores a *a.* Se X é uma variável discreta em um intervalo x1<x2<... a cdf é

$$Fx(a) = \sum_{i:xi \leq a} Px(xi) \qquad \text{(Eq. 7)}$$

onde $Px(Xi)$ é a função de distribuição de massa de X (*mass probability function* - mpf). Para uma variável contínua Y (tempo, por exemplo) teremos como cdf

$$Fy(a) = \int_{-\infty}^{a} fY(y)dy \qquad \text{(Eq. 8)}$$

Onde $f_Y(y)$ é a função de densidade de probabilidade (*probability density function* - pdf) of Y.

Um valor de suma importância em análise estatística é a esperança ou valor esperado de uma variável aleatória. A esperança é a medida da localização da distribuição desta variável e é denotada por:

$$E[Y] = \int_{-\infty}^{\infty} fY(y)dy \qquad \text{(Eq. 9)}$$

Sendo *Y* uma variável contínua.

$$E[X] = \sum_{i=1}^{n} xi \,.\, pX(xi) \qquad \text{(Eq. 10)}$$

Para *X* discreta.

A variância determina o grau de dispersão em uma distribuição. Na Equação 11 temos a variância para uma variável aleatória e na Equação 12 o desvio padrão.

$$Var[X] = E[(X-E[X])^2] \quad \text{(Eq. 11)}$$

$$\delta = \sqrt{Var\,[X]} \quad \text{(Eq. 12)}$$

Em diversas situações, porém, a análise da dispersão não é possível de ser feita observando apenas o desvio-padrão, uma vez que o mesmo depende da ordem de grandeza da variável, sendo difícil afirmar se a dispersão é grande ou pequena. Uma forma de eliminar essa ordem de grandeza é calculando o coeficiente de variação (CV), sendo um valor adimensional, relacionando o desvio-padrão e a média, como temos na Equação 13. Em geral, quando o CV é menor que 0,25 , significa que os dados estão homogêneos. Porém, este tipo de interpretação varia de aplicação para aplicação.

$$CV = \frac{\delta}{E[X]} \quad \text{(Eq. 13)}$$

Em certas amostras é possível encontrar observações inconsistentes. Uma observação é tida como um *outlier* quando seu valor é muito maior ou menor que os outros valores da amostra em questão [48]. Em geral são causados por erros na geração de dados, pela ocorrência de um evento raro, ou pela própria natureza do evento. O método mais óbvio de determinar se uma observação é um *outlier* é pelo cálculo do *z-score* (Equação 14).

$$z = \frac{x - E[X]}{\delta} \quad \text{(Eq. 14)}$$

De acordo com a Regra Empírica e o Teorema de Tchebysheff, a maioria das observações devem possuir um valor z menor que 3. Desta forma, caso seja encontrado um z maior que 3, significa que a observação deve ser considerada um *outlier*. A necessidade de eliminação dos *outliers* é uma questão que deve ser cuidadosamente analisada, de acordo com o tipo de aplicação em estudo e com as necessidades do analista.

Apêndice B

Processos Estocásticos e Cadeias de Markov

Um processo estocástico é um conjunto de variáveis X(t) indexadas por um parâmetro *t* (*t* pertencendo a um conjunto *T*). X(t) representa uma característica mensurável de interesse no tempo *t*, por exemplo, um nível de estoque em um tempo *t*:

- X(t) representa o estado do sistema no tempo *t*
- X(t) é definido pelo espaço de estados

Processos estocásticos podem ser divididos em duas categorias:

Por estado { Discretos (cadeias) – X(t) é definido em um conjunto enumerável ou finito
Contínuo - caso contrário

Por tempo { Discreto – *t* é definido em um conjunto enumerável ou finito
Contínuo – caso contrário

Um processo Markoviano é um processo estocástico onde:

$$P\{X(t_{k+1}) \leq x_{k+1} | X(t_k)=x_k, X(t_{k-1})=x_{k-1}, \ldots, X(t_1)=x_1, X(t_0)=x_0\} = \quad \text{(Eq. 15)}$$
$$P\{X(t_{k+1}) \leq x_{k+1} | X(t_k)=x_k\} \text{ para } t_0 \leq t_1 \leq \ldots \leq t_k = 0,1,\ldots \text{ e toda seqüência } k_0, k_1, \ldots k_{t-1}, k_t, k_{t+1}$$

Neste caso, o estado futuro depende apenas do estado presente (ausência de memória). Este processo será uma Cadeia de Markov quando as variáveis aleatórias X(t) são definidas em um espaço de estados discreto. A probabilidade $P\{X(t_{k+1}) = x_{k+1} | X(t_k)=x_k\}$ é conhecida como probabilidade de transição do estado no tempo t_k para um estado no tempo t_{k+1}. A matriz de transição *P* é de grande valia nos cálculos de probabilidades estacionárias.

Quando o tempo é considerado discreto, a cadeia é conhecida como Cadeia de Markov de Tempo Discreto (*Discrete Time Markov Chain* - DTMC).

$$P\{X_{k+1} = x_{k+1} | X_k=x_k, X_{k-1}=x_{k-1}, \ldots, X_1=x_1, X_0=x_0\} = \quad \text{(Eq. 16)}$$

$P\{X_{k+1} = x_{k+1} | X_k=x_k\}$,*para toda seqüência 0,1,…,k,k+1*

Se $P\{X_{k+1} = x_{k+1} | X_k=x_k\} = P\{X_1 = x_1 | X0=x_0\} \ldots P\{X_{k+1} \leq x_{k+1} | X_k=x_k\}$ então as probabilidades de transição são estacionárias (não mudam com o tempo).

Considerando $p_{ij}^{(n)}$ a probabilidade de transição de um estado *i* para um estado *j* em um tempo *n*, podemos classificar os estados de uma cadeia de Markov em:

- **Acessível –** Um estado *j* é acessível a partir de um estado *i* se $p_{ij}^{(n)} > 0$, para qualquer *n*;
- **Comunicante –** Dois estados *j* e *i* são comunicantes se *j* é acessível a partir de *i* e vice-versa; se *i* e *k* são comunicantes e *k* e *j* também o são, então, *i* e *j* são comunicantes. Dois estados comunicantes pertencem a mesma classe. Se todos os estados pertencem à mesma classe, a cadeia é irredutível.
- **Transiente –** Um estado é transiente (temporário) se, uma vez neste estado, o processo não pode voltar novamente à ele. É transiente se, e somente se, há um estado *j* ($j \neq i$) acessível por *i*, mas não vice-versa, sendo visitado um número finito de vezes.
- **Recorrente –** Um estado é recorrente se, entrando neste estado, o processo retornará a ele novamente; é recorrente se não for transiente.
- **Absorvente –** Um estado é absorvente se, uma vez alcançado, o processo não irá deixá-lo; é um caso especial de estado recorrente, onde *pii*=1.
- **Periódico** – Um estado é periódico com período *t* se o retorno a este estado é possível apenas em *t, 2t, 3t*... passos, para t>1 (sendo t o máximo divisor comum entre eles). Se um estado não é periódico, é referido como aperiódico.
- **Ergódico** – Estado recorrente e aperiódico e comunicante com os outros estados. Se todos os estados forem ergódicos, então, a cadeia é ergódica.

Uma cadeia de Markov de tempo contínuo (*Continuous Time Markov Chain* - CTMC) é um processo estocástico cujo tempo varia continuamente e, em geral, procura responder às seguintes questões: por quanto tempo um sistema se mantém num dado estado e qual será o próximo estado. Temos na Figura 37 um exemplo de CTMC:

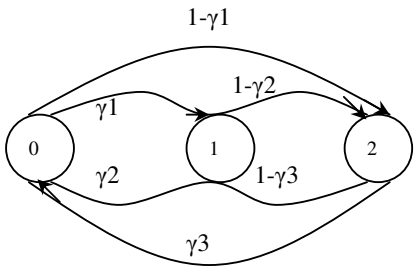


Figura 37. Cadeia de Markov de Tempo Contínuo

Um fato importante a ser lembrado é que não há transições de um estado para ele mesmo, uma vez que o tempo é contínuo, e não determinado em passos. Porém, antes de discutir mais aprofundadamente as CTMCs, devemos compreender o conceito de processos semi-Markovianos. Processos semi-Markovianos são processos estocásticos onde as transições podem ser descritas por uma cadeia de Markov, onde os tempos de duração são variáveis aleatórias. Um processo semi-Markoviano será uma CTMC quando:

1. Ausência de memória ($P\{Y(t)=j | Y(s)=i, Y(u)=k, 0 \leq u < s\} = P\{Y(t)=j | Y(s)=i\}$)

2. Processo estacionário no tempo ($P\{Y(t)=j|\ Y(s)=i\} = P\{Y(t-s)= |Y(0) = i\}$)

Isto significa que o processo deve ser uma cadeia de Markov e o tempo de duração é exponencialmente distribuído (seu parâmetro λ é conhecido como *taxa*). No caso das DTMCs, o tempo é geometricamente distribuído.

Uma CTMC com espaço de estados S = {1,2,...} é caracterizada por um vetor de estados iniciais $\pi^{(0)} = [\pi 1^{(0)}\ \pi 2^{(0)}\ \cdots\ \pi m^{(0)}]$ onde $\pi i^{(0)} = P\{Y_0=i\}$ tendo um matriz geradora *Q*:

$$Q = \begin{pmatrix} -q11 & q12... & q1m \\ q21 & -q22... & q2m \\ ... & ... & ... \\ qm1 & qm2... & -qmm \end{pmatrix} \qquad \text{(Eq. 17)}$$

Onde *qij* é a taxa que corresponde à mudança de estado de *i* para *j* sendo $qii = -\Sigma_{j\neq i}$ qij. Como exemplo, podemos considerar uma CTMC onde:

- Y(t) é o número de clientes de um sistema em um tempo t ≥0 e com espaço de estados S= {0,1,2,..};
- Os dois tipos de eventos que podem ocorrer são Ei {cliente chega, cliente deixa o sistema};
- O tempo até um cliente chegar é exponencialmente distribuído com taxa λ;
- O tempo que um cliente leva para ser atendido e deixar o sistema é exponencialmente distribuído com taxa μ.

Neste caso, teremos a seguinte matriz geradora (colunas e linhas representam o espaço de estados):

$$Q = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & 0 & ... \\ \mu & -(\lambda+\mu) & \lambda & 0 & 0 & ... \\ 0 & \mu & -(\lambda+\mu) & \lambda & 0 & ... \\ .. & .. & .. & .. & .. & .. \end{pmatrix} \qquad \text{(Eq. 18)}$$

Para obter a matriz de transição, basta "normalizar" a matriz *Q*:

$$P\{xi \leq min_j\ Xj\} = \frac{\lambda i}{\sum_{j=1}^{n} \lambda j} \qquad \text{(Eq. 19)}$$

Se $qii \neq 0$, então $pii=0$ e $pij = qij/qii$ para todo $j \neq i$
Se $qii = 0$, então $pii=1$ e $pij = 0$ para todo $j \neq i$

Neste exemplo teríamos a matriz *P* a seguir:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & \ldots \\ \frac{\mu}{\lambda+\mu} & 0 & \frac{\lambda}{\lambda+\mu} & 0 & 0 & \ldots \\ 0 & \frac{\mu}{\lambda+\mu} & 0 & \frac{\lambda}{\lambda+\mu} & 0 & \ldots \\ .. & .. & .. & .. & .. & .. \end{pmatrix} \quad \text{(Eq. 20)}$$

A matriz geradora Q para CTMCs tem um papel similar à matriz de transição P para as DTMCs. É importante lembrar que a soma dos elementos de uma linha em Q é igual a zero, enquanto a soma dos elementos de uma linha em P é igual a 1.

O tempo de absorção de uma cadeia de Markov ($\{Yt\},\ t>0$) com espaço de estados $E=\{1..p\}\cup\Delta$, onde Δ é o estado absorvente, pode ser modelado por uma variável aleatória com uma distribuição *Phase-Type* (distribuição PH) [29]. Para caracterizar esta distribuição é preciso determinar o vetor de probabilidades $\pi=(\pi 1,\ \pi 2,\ldots,\ \pi p)$ onde $\pi i = P\{Y_0 = i\}$. A probabilidade π_Δ é considerada zero. A matriz Q é representada da seguinte forma:

$$Q = \begin{pmatrix} T & t \\ 0\ldots & 0{,}0 \end{pmatrix} \quad \text{(Eq. 21)}$$

Onde T é o gerador *Phase-Type* e t é o vetor de taxas de saída (seu $i$ésimo termo é a intensidade condicional de absorção em Δ a partir do estado i). Desde que a soma de ms linha em Q deve ser zero, temos $t = Te$, com $e = (1,\ldots,1)'$, um vetor coluna com p 1s. Basicamente, uma distribuição PH é representada por um par (π,T) e sua ordem p. Sua cdf é dada por:

$F(y) = 1 - \pi\, exp(Ty)\, e$, com pdf (Eq. 22)

$f(y) = \pi\ exp(Ty\ )t$ (Eq. 23)

Apêndice C

Arquitetura do modelo # de componentes