Caracterização de Desempenho de Programas SPMD Utilizando Modelos Probabilísticos

Nivia Cruz Quental
Departamento de Sistemas Computacionais
Universidade de Pernambuco
ncq@dsc.upe.br

Francisco Heron de Carvalho Junior
Departamento de Computação
Universidade Federal do Ceará
heron@lia.ufc.br

Ricardo Massa Ferreira Lima
Departamento de Sistemas Computacionais
Universidade de Pernambuco
ricardo@dsc.upe.br

Resumo

Em processamento de alto desempenho, o uso de instrumentos para a avaliação quantitativa pode afetar a metodologia e as técnicas de programação usadas no desenvolvimento de uma aplicação. Este artigo apresenta um estudo experimental que visa caracterizar probabilisticamente os tempos de computação e sincronização de programas paralelos. Os resultados serão aplicados na construção de instrumentos para avaliação de desempenho de programas paralelos baseados no modelo # de componentes, voltado à programação paralela distribuída, que possui a importante característica de separar os interesses de computação e coordenação, permitindo a modelagem da especificação em nível de coordenação; a ocorrência de eventos de computação e sincronização é capturada estaticamente, usando redes de Petri estocásticas.

1. Introdução

Instrumentos para avaliação de desempenho constituem importantes requisitos no desenvolvimento de programas paralelos, devido a sensibilidade destes em relação às características de potenciais ambientes de execução, considerados seus aspectos de *software* e *hardware*, sobretudo em arquiteturas distribuídas. Alguns especialistas consideram a atividade de avaliação de desempenho uma arte, uma vez que exige um estudo aprofundado das características do programa e do ambiente de execução em questão para escolha de uma metodologia adequada para sua implementação [22]. Algumas das principais características que modelos de desempenho devem possuir são [7]: (1) aplicabilidade do modelo para qualquer metodologia e arquitetura; (2) relevância do modelo, especialmente em *clusters*; (3) permitir que usuários comuns compreendam a influência exercida pela aplicação no desempenho em diversos aspectos; (4) auxiliar programadores a validar ou invalidar hipóteses.

O modelo # de componentes tem sido proposto como uma alternativa orientada a interesses às técnicas convencionais de programação paralela, orientadas a processos [12]. Estas não têm sido capazes de conciliar os requisitos de eficiência, portabilidade, generalidade e abstração, desejáveis às técnicas de programação paralela, especialmente devido ao surgimento recente de aplicações de larga escala, motivado pelo advento das grades computacionais como arquiteturas para processamento de alto desempenho. O modelo # separa em meios ortogonais os interesses relacionados a computação e coordenação

O modelo # é classificado como um modelo de coordenação de caráter exógeno, uma vez que a orquestração (coordenação) dos elementos computacionais, encapsulados em componentes, é realizada completamente no meio de coordenação, através de componentes que assumem o papel de *conectores* [20]. Tal característica torna possível a tradução de programas #, obviamente com respeito aos interesses em nível de coordenação, para o formalismo de redes de Petri, proposto na década de sessenta [5] e, desde então, utilizado em aplicações de diversas áreas, incluindo áreas não relacionadas a ciência da computação [21]. Viabiliza-se, por meio desta abordagem, a verificação/análise de propriedades formais e a avaliação de desempenho de programas paralelos especificados segundo o modelo #. Este trabalho está particularmente interessado neste último. Para isso, propõe-se a tradução de programas # para redes de Petri estocásticas (SPN), as quais têm sido empregadas largamente na avaliação de desempenho de sistemas. Em SPN's, à ocorrência de eventos, modelada pelo disparo de tran-

sições, podem ser atribuídas variáveis aleatórias, contínuas e independentes, as quais descrevem o tempo para ocorrência destes eventos. Para este tipo de rede de Petri, a teoria das cadeias de Markov de tempo contínuo pode ser aplicada na avaliação de desempenho. Em particular, programas # permitem modelar separadamente a ocorrência de eventos de computação e sinronização/comunicação, usando transições temporizadas. Tal informação é suficiente para caracterizar o desempenho de um programa paralelo. Deseja-se usufruir desta característica, para proposição de uma metodologia para avaliação de desempenho de programas # implementada no topo de ferramentas de suporte a SPN's, como TimeNET e GreatSPN.

Em [6], Duncan Grove confirma que valores médios não são suficientes para caracterizar comunicação em programas paralelos, e que tratá-los como variáveis aleatórias pode ser uma alternativa mais razoável. Grove propõe uma ferramenta de benchmarking chamada MPIBench, que promete caracterizar individual e estatisticamente operações de comunicação MPI. A-bordargem estatística também é usada em [10], onde o estudo de caso é feito em um *Call Center*. Em [17], encontra-se um trabalho extenso sobre aplicações de cadeias de Markov e SPNs, incluindo o processamento paralelo. A definição clássica de SPN supõe que a variável aleatória que descreve o tempo restante para o disparo de uma transição seja exponencialmente distribuída. Tal hipótese constitui uma aproximação satisfatória para certos problemas, como no caso dos estudos do desempenho de redes sem-fio descritos em [2]. Infelizmente, nem todos os sistemas reais são favorecidos por tal propriedade, com vários exemplos na literatura [19]. Baseado nesta compreensão, foram propostas diversas extensões de SPNs que utilizam distribuições não-exponenciais [1].

Este artigo apresenta um estudo experimental, de caráter exploratório, cujo objetivo é a caracterização das variáveis aleatórias usadas para modelagem do tempo de eventos de coordenação e computação em programas paralelos. Assume-se como hipótese inicial que a distribuição exponencial constitui uma aproximação aceitável. Porém, como serão mostrados, os resultados do experimento contradizem tal hipótese. Discutiremos então, baseados na análise dos resultados, uma caracterização aceitável para o tipo de distribuição, com vistas ao propósito de modelagem de desempenho de programas #. Os resultados apresentados neste artigo são fundamentais para a compreensão do comportamento de programas paralelos quando modelados usando redes de Petri temporizadas. O escopo de relevância dos resultados não é restrito ao modelo # de programação, podendo ser aplicável à caracterização estocástica do desempenho de programas paralelos em geral. Espera-se que, com base nas conclusões deste estudo, definir-se uma metodologia sistemática para avaliação de programas # usando SPN's, implementada no topo de ferramentas pré-existentes capazes de suportar este formalismo.

Na seção 2, é descrito o modelo de componentes # Na seção 3 introduz-se o *NAS Parallel Benchmarks* (NPB), usado como estudo de caso neste experimento. Em seguida, o experimento é discutido, bem como um maior detalhamento de cada *benchmark*, e, finalmente, na seção 5, apresentamos as principais conclusões.

2. O Modelo de Componentes

O modelo de componentes # tem sido proposto visando conciliar requisitos de alto nível de abstração, generalidade, portabilidade e eficiência na tarefa de programação paralela distribuída [12,15]. Seu principal atributo é mover a atividade de programação paralela de uma perspectiva baseada a processos para uma perspectiva orientada a interesses, os quais são encapsuláveis em componentes #. Este modelo é aplicável a qualquer tipo de projeto que envolva programação paralela, facilitando a tarefa do programador de codificar e analisar seu programa paralelo.

Um importante aspecto relativo ao modelo # é a separação dos interesses relativos à programação das computações e à coordenação de processos permitindo que *computação* e *coordenação* possam ser manipuladas separadamente (em níveis ortogonais) e integradas usando técnicas oriundas do paradigma de orientação a aspectos [11]. No nível de coordenação, são especificados todos os aspectos concernentes à instanciação e gerenciamento da execução paralela, usando a linguagem de configuração #, projetada para este propósito (*Hash Configuration Language*, HCL). No nível de computação, é implementada toda a funcionalidade computacional do programa, usando uma linguagem *host* capaz de descrever computações, como C, Fortran, Haskell, entre outras.

Esta separação em dois níveis permite provar propriedades formais e avaliar a eficiência do paralelismo de um programa # concentrando-se somente nas informações providas no meio de coordenação. Da mesma forma, é possível que ferramentas apropriadas sejam usadas para análise apenas no meio de computação. Atualmente, existe um esquema de tradução de configurações em HCL para redes de Petri [16], visando possibilitar a prova de propriedades formais de programas # usando ferramentas pré-existentes como INA e PEP [19]. Na Figura 1, é apresentada uma interface que define o comportamento das unidades que constituem a rede de processos da versão # de um programa paralelo (kernel IS dos *NAS Parallel Benchmarks*, descritos na próxima seção). Abaixo a tradução do

comportamento de um processo para redes de Petri. Deve-se notar que a coordenação é definida pela efetivação de uma seqüência de operações de comunicação.

Recentemente, a tradução de programas # para redes de Petri [16] tem sido proposta com a finalidade de permitir simulação e avaliação de seu desempenho. Para isso, propõe-se o emprego das redes de Petri estocásticas, cujos disparos das transições podem modelar a efetivação de operações de comunicação ou computação. Em outras abordagens de programação paralela, onde as primitivas de sincronização/comunicação e código das computações encontram-se misturadas ao código que descreve as computações, simulações desse tipo são difíceis de obter automaticamente, necessitando da análise do código fonte dos programas e a inferência da ordem em que eventos de sincronização se completam. Em configurações #, tais informações são explícitas e estáticas.

Em nível de coordenação, programas # são descritos pela composição de uma coleção de *componentes*, cada qual interessada no atendimento de certo interesse (*concern*) de aplicação. São suportados interesses funcionais e não-funcionais, os quais podem estar entrelaçados. A Figura 2 resume a arquitetura do modelo #, do ponto de vista composicional e de processos. Componentes são descritos através de uma rede de *unidades* conectadas por *canais* de comunicação (ponto-a-ponto, unidirecionais e tipados) através de *portas* de comunicação que definem suas interfaces. Uma unidade pode ser enxergada como uma *fatia* de um processo, em seu sentido geralmente empregado em programação paralela, o qual especifica o papel do processo com respeito ao interesse endereçado pelo componente. É explícita a ordem de ativação das portas de cada unidade, através de um protocolo descrito usando um formalismo (expressões regulares enriquecidas com semáforos) que possui equivalência expressiva com redes de Petri rotuladas terminais. A uma unidade pode ser atribuída um componente (composição aninhada). Componentes aninhados podem ser sobrepostos através da unificação de unidades que fazem parte de sua composição.

Essa infra-estrutura básica de passagem de mensagens, modelada através dos conceitos de portas e canais de comunicação permite modelar virtualmente qualquer tecnologia de programação paralela que possa ser definida em termos de passagem de mensagens (a implementação atual de HCL basea-se em MPI).

A abstração no modelo # é atingida pelo emprego do conceito de programação orientada a esqueletos [14], componentes cuja preocupação endereçada encontra-se parametrizada.

Ao contrário de outros modelos de componentes, os quais suportam exclusivamente a composição aninhada de componentes, o modelo # suporta a composição de componentes por sobreposição. Para isso, unidades provenientes de componentes distintos podem ser unificadas. Em um componente de aplicação, um processo, sob a perspectiva de programação baseado em processos, é definido como um conjunto de unidades unificadas, as quais definem suas fatias. Cada unidade especifica o papel do processo em um dado interesse de aplicação.

Um ambiente para construção de aplicações baseado no modelo # tem sido proposto. Através de *plug-ins*, será possível adicionar ao ambiente módulos com fins específicos, como a análise de propriedades formais de programas e avaliação de desempenho usando redes de Petri, cujo desenvolvimento deverá ser baseado nos resultados desta pesquisa. Este trabalho exploratório visa, portanto, a análise do comportamento de programas paralelos sobre *clusters*, com a finalidade de prover informações úteis para a definição de uma metodologia de avaliação de desempenho de programas # usando redes de Petri estocásticas e sua incorporação ao ambiente de programação #, discutida mais adiante.

```
interface IIS where
ports: IAllReduce @ bs #
       IAllToAllv @ kb #
       IRShift    @ vf #
       IAllReduce @ pv
protocol: seq { do bs; do kb;
                repeat seq {do bs; do kb}
                   until <bs&kb>;
                do vf; do pv }
```

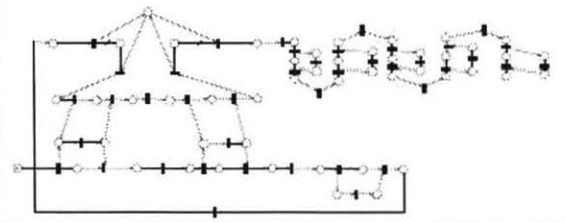

Figura1: Interface das Unidades de IS e sua Tradução para Redes de Petri

3. Os *NAS Parallel Benchmarks*

O *NAS Parallel Benchmarks* [8] foi um esforço de pesquisadores no *Nasa Ames Research Center* no programa NAS (*Numerical Aerodynamic Symulation*), com o objetivo de prover avanços na simulação de um veículo aeroespacial durante sua missão e durante todo o seu ciclo de vida. É composto por oito programas de computação científica, escritos em C e Fortran com a biblioteca de passagem de mensagens MPI, listados a seguir: EP (*Embarassingly Parallel*), IS (*Integer Sort*), CG (*Conjugate Gradient*), FT (*Fast Fourier Transform*), MG (*Multigrid*), LU (*LU solver*), SP (*Pentadiagonal Solver*), BT (*Block Tridiagonal*), cuja carga computacional 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, sendo exaustivamente estudados na literatura, nos incentiva a utilizá-los como estudo de caso deste artigo.

Os NPBs, já foram utilizados em pesquisas relacionadas ao modelo#. Versões # para um subconjunto do NPB já foram implementadas para realização de medições simples, para comparação de desempenho de programas # em relação a suas versões originais em trabalhos anteriores [13].

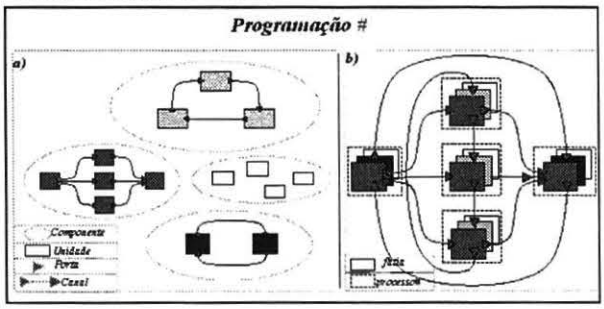

Figura2: Visão do modelo#: Componentes(1a) e Processos(1b)

4. Metodologia e Resultados

O experimento foi realizado no *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 versão empregada do MPI (*Message Passing Interface*) é o MPICH versão 1.2.4. A versão dos NPBs utilizada foi a 2.3. Por questões de simplicidade apenas as cargas do tipo A e B foram experimentadas.

O experimento consistiu na coleta de amostras dos tempos de sincronização (chamadas a rotinas MPI) e computação para uma seqüência de N(tamanho da amostra) execuções de cada caso de medição, onde N é calculado para cada caso utilizando medições preliminares de forma a permitir maior segurança no tratamento estatístico dos dados. Definimos como casos de medição as triplas obtidas do produto cartesiano P × N × C, onde P = {EP, IS, CG, MG, LU}, N = {8,16,32,64} e C = {A,B}. Dessa forma, um caso de medição (p, n ,c) indica a execução de uma certa classe de carga padrão c de programa p do NPB sob um certo número de processadores n. Ressaltamos ainda que o experimento foi realizado com o *cluster* em modo exclusivo de uso.

Para cada caso de medição as N medidas de comunicação e computação foram providas como entrada para a ferramenta de modelagem ARENA (*Input Analyzer*). Esta permite a construção de histogramas e o ajuste de curvas de distribuição probabilística aos dados. Atualmente o *Input Analizer* suporta o ajuste segundo os seguintes tipos de distribuições: Beta, Empirical, Erlang, Exponencial, Gamma, Johson, Lognormal, Normal, Poisson, Triangular, Uniform e Weibull. Podemos então avaliar que tipo de distribuição, dentre estas, possui melhor ajuste para os dados obtidos pelo experimento. Por questões práticas, apenas os histogramas mais relevantes estarão presentes neste artigo, os quais usaremos na discussão. Os gráficos a seguir apresentam uma relação de duração (do tempo de computação ou comunicação) *versus* freqüência. Esse experimento permitirá uma caracterização mais fiel de rotinas de comunicação e computação fazendo com que, futuramente, o processo de avaliação torne-se mais simples e rápido para o usuário, independente da aplicação em questão.

4.1. O kernel EP

O *kernel* EP, procura estimar os limites superiores de desempenho de operações de ponto flutuante, sendo o benchmark mais simples em termos de comunicação interprocessador, utilizando apenas MPI_AllReduce ao fim das computações.

Foi possível constatar neste experimento que, pelo fato de ser um *kernel* "embaraçosamente paralelo", o EP assume um comportamento relativamente previsível. Segundo o ARENA, na maioria dos casos as distribuições lognormal e erlang constituem uma boa aproximação (Figuras 3 e 4). Nota-se, para as comunicações, a proximidade do desvio padrão da média (típica de uma erlang), enquanto as computações apresentam uma variabilidade menor em relação à média. É importante lembrar que erlang é uma distribuição *Phase Type*, ou seja, pode ser representada como uma mistura de exponenciais. Isto reforça a hipótese de existir um relacionamento entre o desacoplamento da arquitetura à qual o programa é submetido e seu distanciamento de um caso ideal (distribuição exponencial). O kernel EP aproxima-se dessa situação ideal, por possuir alto grau de paralelismo, de forma que o estado da rede tem menor influência sobre seus resultados. Notamos que a partir do caso 32 o programa como um todo se comporta melhor (tanto o tempo de computação quanto o de comunicação diminuem, como pode ser visto nas Figuras 5 e 6).

Figura3: Comunicação (EP,8,A)

Figura4: Comunicação (EP.8,B)

4.2. O kernel CG

Este benchmark usa o método da potência inversa para encontrar o maior auto-valor de uma matriz *A* simétrica, positiva esparsa, com um padrão aleatório de zeros. Para tal, um sistema linear é resolvido utilizando o método do gradiente conjugado. A comunicação no kernel CG, diferentemente do EP, que usa exclusivamente operações coletivas, é dada por rotinas ponto a ponto, e faz uso mais intenso dos canais de comunicação. A ferramenta ARENA sugere como melhor aproximação, em sua maioria, distribuição beta e lognormal para as comunicações e weibull e lognormal para as computações. Nota-se o baixo desvio padrão em torno da média para todos os casos, conforme podemos notar nas figuras de 7 a 10.

Figura5: Comunicação (EP,32, A)

Figura6: Computação (EP,32, A)

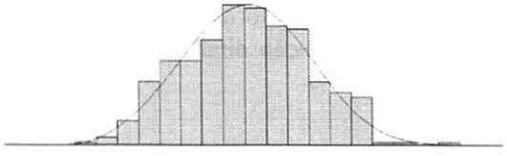

Figura7 : Comunicação (CG,8, B)

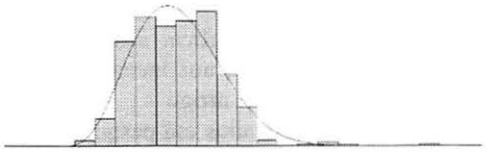

Figura8 : Comunicação (CG,16, A)

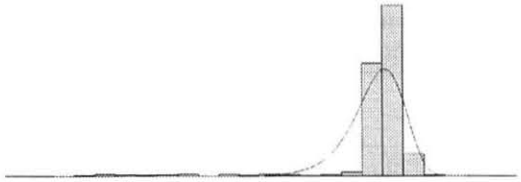

Figura9 : Computação (CG,32,A)

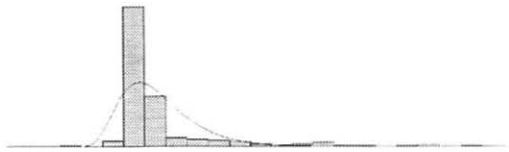

Figura10 : Comunicação (CG,64, B)

4.3. O kernel IS

Este programa utiliza um algoritmo de *bucket sort* paralelo, ordenando um vetor de inteiros cujos valores são determinados aleatoriamente. As principais rotinas MPI envolvidas são MPI_AllReduce, MPI_Alltoall e MPI_Alltoallv. Em geral, o IS apresenta um comportamento estável, no qual o ARENA identifica boas aproximações para weibull e erlang, para comunicação e weibull e lognormal para a computação. No caso "A" para 64 processadores, porém, detectamos um desvio padrão maior, possivelmente decorrente do aumento do paralelismo, para um caso computacionalmente mais leve, como o caso "A", fazendo com que a média seja deslocada para a esquerda (próxima do mínimo), dando um aspecto de "cauda pesada" no histograma (Figuras 11 a 14).

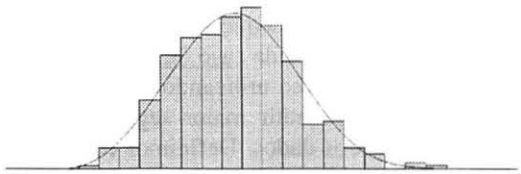

Figura11 : Comunicação (IS,8, A)

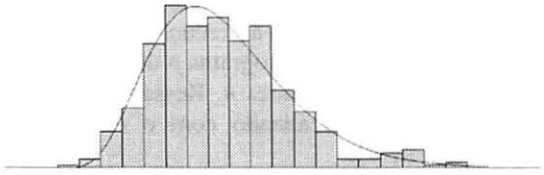

Figura12 : Comunicação (IS,16, B)

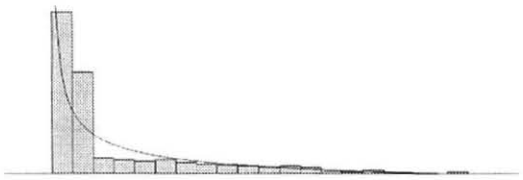

Figura13 : Comunicação (IS,64, A)

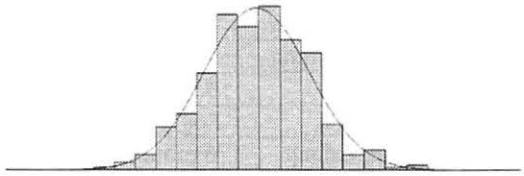

Figura14 : Computação (IS,64,A)

4.4. O kernel MG

O algoritmo implementado no kernel MG procura obter uma solução aproximada para um problema de Poisson, utilizando quatro iterações do algoritmo *V-cycle multigrid*. Assim como no EP, predominam as rotinas coletivas (MPI_Allreduce).

Os experimentos comprovaram a semelhança nos histogramas do MG, com os do EP, predominando as distribuições Lognormal e Erlang, como melhores aproximações, de acordo com o ARENA. Não há grandes variações nos formatos dos histogramas entre as classes A e B; esse "bom comportamento" pode ser justificado pelo fato da variação da classe afetar poucos parâmetros do problema (Figuras 15 a 18).

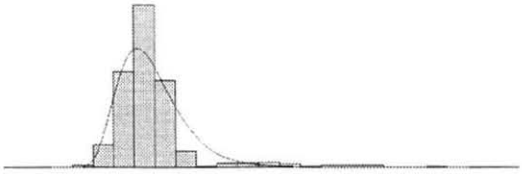

Figura15 : Computação (MG,8,A)

Figura16 : Comunicação (MG,16, A)

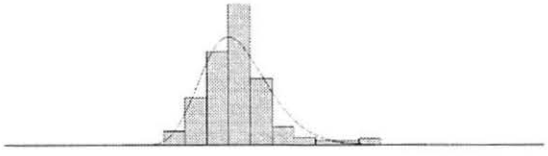

Figura17 : Comunicação (MG,64,B)

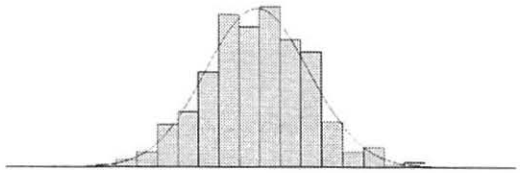

Figura18 : Computação (MG,64, A)

4.5. A aplicação simulada LU

O LU procura utilizar uma discretização para solucionar um sistema de equações diferenciais parciais (PDEs). Pelo método da sobre-relaxação sucessiva e simétrica (SSOR). É o único *benchmark* caracterizado por enviar muitas mensagens de tamanho curto, predominando as rotinas ponto a ponto, com uma minoria de operações MPI_Allreduce, fazendo um uso intenso dos canais e comunicação, de forma que a partir de 16 processadores, a comunicação passa a ter tempo superior à computação. O ARENA sugere distribuições normal e lognormal como boas aproximações para as comunicações; nota-se pouca variação nos tempos de computação, como pode ser observado nas Figuras 19 a 22.

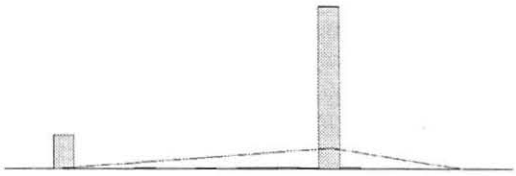

Figura19 : Computação (LU,8, A)

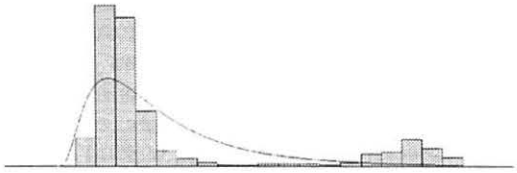

Figura20 : Comunicação (LU,8, A)

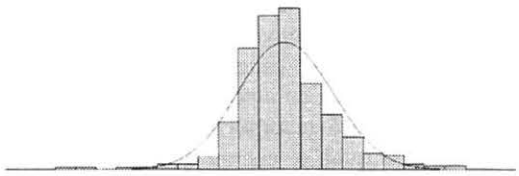

Figura21 : Comunicação (LU,32,A)

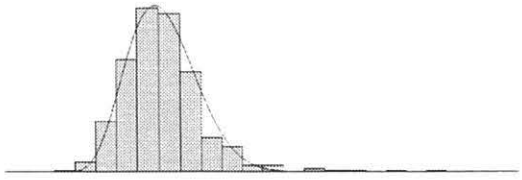

Figura22 : Computação (LU,32 A)

5. Conclusões e trabalhos futuros

É bastante discutível a hipótese de que modelos de comunicação podem ser modelados simplesmente com variáveis aleatórias exponenciais. A importância dada à distribuição exponencial, porém, não é por acaso, uma vez que sua ausência de memória característica permite um mapeamento quase direto entre redes de Petri estocásticas e cadeias de Markov. Em processamento de alto desempenho, temos muitas variáveis a serem consideradas, como arquitetura, ambiente de passagem de mensagens, topologia dos processadores, e o próprio paralelismo da aplicação, de modo que nem sempre o uso de exponenciais constitui uma boa aproximação. O kernel EP apresentou comportamento mais próximo do ideal, possivelmente pelo uso menos freqüente de canais de comunicação, o que nos leva a crer na possível influência do acoplamento do hardware paralelo no desempenho dos programas. Nas tabelas 1 a 4, apresentamos um resumo dos resultados deste experimento, onde para cada caso de medição listamos as duas distribuições com os melhores ajustes, segundo a ferramenta ARENA.

Uma classe de distribuições, porém, pode ser utilizada para representar sistemas de maneira mais exata, sem perder as propriedades markovianas das SPNs. São as distribuições *Phase-Type*, cujas variáveis derivam de uma combinação de exponenciais, dando origem a novas extensões para as SPNs[1]. As aproximações obtidas sugerem distribuições para as quais se encontram propostas na literatura várias técnicas para ajuste usando distribuições *Phase-Type*, como Erlang (um caso especial de P*hase Type*), Weibull e Lognormal [4,3,10].

De fato, o próximo passo deste trabalho será utilizar esta abordagem para gerar um modelo SPN equivalente dos benchmarks NPB já implementados no modelo #, aos quais já foram desenvolvidos modelos em redes de Petri *lugar/transição*. Posteriormente, será desenvolvida uma metodologia de avaliação baseado nessa abordagem, a qual será incorporada no ambiente de programação #, o qual se encontra em desenvolvimento como uma ferramenta de apoio. Possivelmente deverá ser acoplado ao compilador # um back-end que procurará obter informações de desempenho (usando uma ferramenta como o MPIBench, por exemplo) das rotinas de computação e comunicação, para a partir daí, realizar as necessárias aproximações por fase no design das SPNs correspondentes, tornando o processo de tradução de programas # para SPNs quase automático.

Tabela 1:Distribuições de Melhor Ajuste para Tempos de Comunicação EP e CG

	EP	CG
A8	Lognormal, Gamma	Lognormal,Gamma
A16	Lognormal, Gamma	Beta,Weibull
A32	Beta, Gamma	Beta,Erlang
A64	Erlang, Gamma	Lognormal,Gamma
B8	Lognormal, Erlang	Weibull,Beta
B16	Lognoral,Beta	Beta,Lognormal
B32	Normal, Erlang	Gamma,Erlang
B64	Erlang, Gamma	Lognormal,Gamma

Tabela 2: Distribuições de Melhor Ajuste para Tempos de Comunicação IS, MG e LU

	IS	MG	LU
A8	Weibull,Norm	Logn.,Gamma	Logn.,Erl.
A16	Gamma.,Erl	Erl.,Gamma	Beta,Triang
A32	Gamma,Erl	Logn.,Erl.	Norm.,Beta
A64	Beta,Weibull	Erl,Logn.	Weib,Norm
B8	Normal,Weib	Logn.,Gamma	Triang,Beta
B16	Erlang,Gamma	Logn.,Gamma	Logn,Erl
B32	Weibull,Norm	Normal,Beta	Triang,Beta
B64	Weibull,Norm	Logn.,Erl	Triang,Beta

Tabela 3: Distribuições de Melhor Ajuste para Tempos de Computação EP e CG

	EP	CG
A8	Beta,Weibull	Weibull,beta
A16	Lognormal,Exponencial	Lognormal,Beta
A32	Lognormal,Erlang	Weibull,Beta
A64	Normal,Erlang	Weibull,Beta
B8	Weibull.Normal	Triangular,Beta
B16	Lognormal,Exponencial	Beta,Gamma
B32	Lognormal,Beta	Lognormal,Erlang
B64	Normal,Beta	Lognormal,Beta

Tabela 4: Distribuições de Melhor Ajuste para Tempos de Computação IS,MG e LU

	IS	MG	LU
A8	Logn.,Gamma	Logn.,Erlang	Triang,Norm
A16	Logn.,Erlang	Weib.,Beta	Beta,Logn
A32	Weib.,Normal	Gamma,Erl	Beta,Gam
A64	Normal,Beta	Logn.,Erlang	Logn,Gam.
B8	Logn.,Erlang	Logn.,Gamma	Logn,Beta
B16	Triang,Logn	Normal,Beta	Triang,Beta
B32	Weib.,Beta	Logn.,Erlang	Logn,Gam
B64	Normal,Weib.	Logn.,Gamma	Nomr,Weib

6. Agradecimentos

Ao Departamento de Sistemas Computacionais da UPE, FDPE/UPE Centro de Informática da UFPE e ao Departamento de Computação da UFC pelo uso de recursos computacionais e a Glauco E. Gonçalves e Fernando A. A. Lins, mestrandos do Centro de Informática da UFPE pelo auxílio no tratamento estatístico de dados do *kernel* IS.

7. Referências

[1] A. Bobbio, M. Telek. "Non Exponential Stochastic Petri Nets: an Overview of Methods and Techniques." In: Computer Systems Science and Engineering, Vol. 13, No. 6, pages 339-351. 1998

[2] A. Heindl, R. German. "Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets." *Performance evaulatioin : an international journal.* Elsevier, 2001.

[3] A. Horváth, M. Telek. "Approximating Heavy Tailed Distributions with Phase Type Distributions", Proceedings of International Conference on Matrix-Analytic Methods in Stochastic Models, Leuven, Belgium, 2000.

[4] A. Riska, V. Diev and E. Smirni."Efficient Fitting of Long-Tailed Data Sets into Phase-Type Distributions", ACM Sigmetrics Performance Evaluation Review, vol. 20, n. 3, p. 6-8, 2002.

[5] C. A. Petri. "Kommunikation mit Automaten". Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, 1(1), 1966.

[6] D.A Grove. and P.D. Coddington 2004 "Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers", Proc. Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'04), Santa Fe, USA 2004.

[7] D.A. Grove. "Performance Modeling of Messaging Passing Parallel Programs", PhD Thesis, University of Adelaide, 2003

[8] D. H. Bayley, T. Harris, W. Shapir, R. van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0", *Tech. Rep. NAS-95-020*, NASA Ames Research Center, Dec. 1995, http://www.nas.nasa.org/NAS/NPB.

[9] E. Best, J.Esparza , B. Grahlmann, S. Melzer, S.Römer, and F. Wallner. "The PEP Verification System." In Workshop on Formal Design of Safety Critical Embedded Systems (FEmSys'97), 1997.

[10] E. Ishay, "Fitting Phase-Type Distributions to Data from a Telephone Call Center", Dissertação de Mestrado, Israel Institute of Technology, 2002.

[11] F.H Carvalho Junior, R.D.Lins. "Using Aspects for Supporting Procedural Modules in # Programming". In Proceedings of the EURO-PAR 2005. Lisboa, Portugal, 2005.

[12] F.H. Carvalho Junior. "The # Model for Parallel Programming: From Processes to Componentes with Insignificant Performance Overheads." Workshop on Components and Frameworks for High Performance Computing, 2005.

[13] F.H Carvalho Junior, R.D. Lins, N.C Quental. On the Implementation of SPMD Applications using Haskell#. *The 15th Symposium on Computer Architecture and High Performance Computing* - XV SBAC-PAD, São Paulo, 2003.

[14] F. H. Carvalho Junior, R.D. Lins. "Topological Skeletons in Haskell#", Proceedings of the IEEE International Parallel and Distributed Processing Sympoium, Nice, France, 2003.

[15] F. H. Carvalho Junior, R.D. Lins. "Haskell#: Parallel Progamming Made Simple and Efficient", Journal of Universal Computer Science, vol. 9, n. 8, 2003.

[16] F.H. Carvalho Junior, R.D. Lins, R.M.F. Lima. "Translating Haskell# into Petri Nets", Proceedings of the 5th International Meeting on High Performance Computing for Computational Science (VECPAR'2002), Porto, Portugal Jun. 2002.

[17] G. Bolch, S.Greiner H. de Meer, K.S> Trivedi"Queueing Networks and Markov Chains – Modelling and Performance Evaluation with Computer Science Applications" John Wiley and Sons inc. 1998.

[18] M.A. Marsan, G. Balbo G, G. Conte, S. Donatelli, G. Franceschinis. "Modelling with generalized stochastic Petri Nets." Università degli studi di Torino – Dipartimento di informática, 1995.

[19] M. Greiner, M. Jobmann and L. Lipsky, "The Importance of Power-Tail Distributions for Modeling Queue Systems", Operations Research, vol. 47, n. 2, pp. 303-326, 1999.

[20] M. Shaw, "Procedure Calls are the Assembly Language of Software Interconnection: Connectors Deserve First-Class Status", LNCS, (International Workshop on Studies of Software Design), Springer-Verlag. 2004.

[21]P.R.M. Maciel, R.D. Lins, P.F. Cunha. "Introdução às Redes de Petri e Aplicações."X Escola de Computação - Campinas - SP. Jul 1996.

[22]R. Jain. "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