Modelagem e Análise de Desempenho de uma Aplicação Paralela Utilizando *Lost Cycles Toolkit**

M.C.S. de Castro [1] W. Meira Jr. [2] C.L. de Amorim [3]

RESUMO

Neste trabalho realizamos a modelagem e o desempenho de uma aplicação particular na área de processamento sísmico. Para isso, utilizamos o *Lost Cycle Toolkit*, que se baseia principalmente em medidas dinâmicas dos programas de aplicação. O *toolkit* usa a priori o conhecimento das fontes e características de *overhead* em um sistema paralelo para guiar o processo de modelamento e avaliar o desempenho. Podem ser medidos o desbalanceamento de carga, o paralelismo, as perdas por sincronização, por comunicação e por contenção de recursos. Com este experimento foi possível observar o comportamento da aplicação quando submetida a uma quantidade variável de processadores.

ABSTRACT

In this work we made the modeling and studied the performance of a particular application in the area of sismic processing. We used the Lost Cycle Toolkit for this purpose. This toolkit is based on dynamic measurements of application programs. The toolkit uses prior knowledge of existing overheads in parallel sytems in order to guide the modeling process and to do performance evaluation. From the results produced by the toolkit we can measure load imbalance, insufficient parallelism, synchronization loss, communication loss and resource contention. The experiments allowed us to observe the behavior of the application when submmited to a varying number of processors.

[1]MSc (COPPE - 1991), Aluna de Doutorado da COPPE/Sistemas (UFRJ); áreas de interesse: Supercomputação e Processamento Paralelo; Professora Assistente da Universidade Federal de Juiz de Fora;

[2]MSc (UFMG - 1993), Aluno de Doutorado da Universidade de Rochester; áreas de interesse: Supercomputação e Processamento Paralelo;

[3]PhD (Imperial College - 1984), MSc (COPPE - 1979); áreas de interesse: Supercomputação e Processamento Paralelo; Professor Adjunto da COPPE/Sistemas;

COPPE/Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP 21945-970 - Rio de Janeiro - RJ
Email clicia @ rio.cos.ufrj.br

* Esse trabalho foi parcialmente financiado pela Finep, CNPq e Capes.

1 Introdução

A implementação de programas paralelos eficientes tem sido um desafio constante para os programadores de sistemas de computadores paralelos. Os programas paralelos apresentam comportamentos diferentes dependendo das relações entre a estrutura da aplicação paralela, a máquina na qual é executada, o número de processadores utilizados e a massa de dados de entrada. A medida de tempo de execução da aplicação pode variar muito em conseqüência destas relações. Dessa forma, o ajuste do desempenho dos programas paralelos pode levar a um consumo de tempo elevado.

Uma vez que o *overhead* paralelo pode surgir de várias fontes, o modelo do desempenho necessita de um entendimento analítico de uma extensa faixa de fatores gerados pelo *hardware* e pelo *software*. Uma variedade de técnicas de predição de desempenho tem sido propostas e podem ser baseadas em informações estáticas[1] e dinâmicas[2] e[3].

A análise de escalabilidade é uma abordagem para predição de desempenho, onde a meta é caracterizar a eficiência das combinações de máquinas e de aplicações. Ela desenvolve modelos de computação analíticos assintóticos e categorias de *overhead* selecionados como uma função do tamanho dos dados de entrada da aplicação e do número de processadores.

A ferramenta pp (*predicate profiling*) foi desenvolvida pelo grupo de pesquisas do departamento de Ciência da Computação da Universidade de Rochester. Esta ferramenta faz parte de um *toolkit* e tem como objetivo auxiliar o programador na implementação de uma aplicação paralela. O programador pode determinar as implementações alternativas e a escalabilidade da aplicação.

A abordagem do *toolkit* é baseada principalmente em medidas dinâmicas dos programas de aplicação utilizando técnicas independentes da máquina e da linguagem. Como a análise de escalabilidade, a abordagem utilizada divide os *overheads* paralelos em categorias que são acessíveis ao modelamento. O processo de construção dos modelos de desempenho usando estas categorias de *overhead* são chamados *lost cycles analysis*[4]. O *Lost Cycles Toolkit* usa a priori o conhecimento das fontes e características de *overhead* em um sistema paralelo para guiar o processo de modelamento. Ele automatiza, através de várias ferramentas, o processo de construção de um modelo de desempenho para uma aplicação paralela. O *Lost Cycles Toolkit* é composto pelas seguintes ferramentas: `pp`, `expgen`, `lca`, `modgen` e `xmodgen`.

A ferramenta `pp` (*predicate profiling*) mede o *overhead* em cada categoria em tempo de execução. É ela que gera um arquivo de eventos para análise posterior. O arquivo de eventos é denominado `elog`. A `expgen` (*experiment generation*) é uma ferramenta para geração de experimentos automática que garante a eficiência da experiência pela incorporação da metodologia de projeto experimental. A ferramenta `lca` (*lost cycles analysis*) é usada para ajuste dos modelos das categorias de *overhead* através dos dados experimentais. Para garantir a precisão nos modelos de desempenho resultantes, é utilizada a ferramenta `modgen` (*model generation*). Essa ferramenta é utilizada pelo usuário de modo interativo. Nela são mostrados os componentes dos modelos de desempenho, e aqueles com ajuste deficiente podem ser alterados. O processo inteiro é coordenado através de uma *interface* gráfica fornecida pela ferramenta `xmodgen`.

Na seção 2 estão descritos os princípios básicos para *lost cycles analysis*, cujas idéias estão embutidas no *Lost Cycles Toolkit*. Na seção 3 está apresentado um estudo de caso mostrando como foi utilizado o *toolkit* com uma aplicação particular na área de processamento sísmico. Na seção 4 fazemos algumas observações sobre a utilização da ferramenta. Finalmente, na seção 5, apresentamos as nossas conclusões.

2 Lost Cycles Analysis

Lost cycles analysis é um conceito utilizado para automatizar a construção de modelos de desempenho de uma aplicação como função de fatores de tempo de execução, tais como o número de processadores (p) e o tamanho dos dados de entrada (n).

Para modelar o desempenho de uma aplicação paralela é necessário construir expressões que levem em consideração o tempo relativo a computação pura (serial) e o *overhead* paralelo. Esse consiste de todo tempo adicional de processador gasto pela implementação paralela.

O *overhead* paralelo pode surgir de várias fontes diferentes num sistema paralelo. Encontrar expressões matemáticas precisas que o descreva pode ser muito difícil. *Lost cycles analysis* trata esta dificuldade usando noções de decomposição de *overhead* paralelo. Para tal, o *overhead* é decomposto em categorias que sejam importantes fontes de ineficiência, incluam todas as fontes de perda de desempenho e que tenham estados mutuamente exclusivos, isto é, que as fontes de *overhead* medidas não pertençam a duas ou mais categorias ao mesmo tempo[4].

As categorias escolhidas de acordo com estas propriedades expressam *overhead* em unidades de medida comuns. Essas unidades são chamadas *lost cycles* (LC) e formam uma única métrica uniforme para comparar diferentes tipos de *overhead.*

Basear-se em medidas de expressões lógicas que reconheçam *lost cycles* é uma abordagem que pode ser útil[5]. Tais expressões são chamadas predicados de desempenho (*performance predicates*). Assim, o conjunto de predicados de desempenho atende as caracteríticas necessárias a decomposição das categorias utilizando a métrica LC. A seguir descrevemos o conjunto de predicados de desempenho utilizados:

Load Imbalance (LI): ciclos de processador gastos ociosamente, enquanto existem tarefas disponíveis;
Insufficient Parallelism (IP): ciclos de processador gastos ociosamente, aguardando tarefas;
Synchronization Loss (SL): ciclos de processador gastos adquirindo um *lock*, ou esperando em uma barreira;
Communication Loss (CL): ciclos de processador gastos aguardando dados se moverem através do sistema;
Resource Contention (RC): ciclos de processador gastos aguardando o acesso a um recurso de *hardware* compartilhado.

Foi implementado o *predicate profiling* destas categorias para programas Fortran rodando no computador paralelo de memória compartilhada KSR1 da Kendal Square Incorporation. A implementação chamada pp consiste de uma biblioteca ligada ao código executável e um *post-processor* de registro de eventos que produz o *profile*,

isto é registra todo o comportamento do programa de aplicação. As chamadas à biblioteca devem ser inseridas dentro do código fonte.

A decomposição do *overhead* paralelo facilita o modelamento de desempenho, pois é possível contabilizar as categorias de *overhead* paralelo usando modelos *default*. Os modelos *default* incorporados no *toolkit* são simples[6], e incorporam pelo menos três comportamentos típicos por categoria. Além de armazenar modelos *default* por cada categoria, a ferramenta `lca` encontra o melhor modelo *default* utilizando um método de minimização baseado em Decomposição de Valores Singulares[7].

Em resumo, para criar um modelo de desempenho para uma aplicação usando *lost cycles analysis* temos os seguintes passos:

1. O usuário projeta um experimento para capturar as medidas de *overhead* necessárias. Baseado na entrada do usuário, a ferramenta **expgen** determina as execuções da aplicação necessárias e cria um programa *script* para execução;

2. O usuário roda os experimentos pela execução do *script*, e a ferramenta `pp` automaticamente extrai as medidas do *overhead* de cada execução;

3. Usando as medidas do programa como entrada, a ferramenta `lca` encontra o modelo *default* de uma variável com melhor ajuste para cada categoria de *overhead*. Se não há modelo *default* suficientemente preciso para cada categoria, o usuário é alertado pela *interface* **xmodgen**[4], que possui a capacidade para sugestão de um modelo alternativo;

4. Usando a ferramente **modgen**, o usuário combina no final o modelo de uma variável em um modelo multivariável da aplicação. Fontes de imprecisão no modelo final podem ser traçadas através da *interface* **xmodgen** para categorias particulares ou escolha de experimentos, permitindo ao usuário ajustar modelos ou compensar dados discrepantes.

3 Geração do Modelo

Para avaliar a ferramenta *Lost Cycles Toolkit*, foi utilizada uma aplicação em particular, denominada SPLIT-STEP. Esta aplicação implementa um procedimento de modelagem sísmica, utilizando maciçamente FFTs complexas. Os dados de entrada correspondem a sismogramas sintéticos representados por matrizes da ordem de 100×512, que são gerados internamente. O programa foi escrito em Fortran 77 e submetido ao processamento serial e paralelo no computador paralelo KSR1 da Kendal Square.

O sistema KSR1 é uma máquina de memória compartilhada distribuída onde foram realizados experimentos com o número de processadores iguais a 1, 2, 4, 8, 16, 20 e 24.

Na paralelização da aplicação (figura 1) foi utilizado o paralelismo de dados em quatro *loops* distintos, o *loop* do domínio de ômega (TAG 2), dois *loops* relativos a

[4] A ferramenta **xmodgen** possui uma janela que dá a visão geral do ajuste do modelo, relacionando a cada categoria de *overhead* um fator. Os fatores próximos a 1.0 representam modelos precisos e valores menores representam modelos imprecisos.

FFT complexa (TAG 4 e TAG 6) e um *loop* com processamento utilizando operações de seno e cosseno (TAG 5).

```
TAG 0
 :
TAG 1
DO 180 IZ = NZ, MAX (LZ - 1), -1          !loop de DZ (indice IZ)
      :
    TAG 2
    DO IW = IWMIN, IWMAX                  !loop de omega (indice IW)
         :
       TAG 3
       DO JK = 2, JKMAX                   !loop de KX (indice JK)
          :
       END DO
         :
    END DO
      :
    TAG 4
    DO I = IWMIN, IWMAX                   !loop de FFTC
       :
    END DO
      :
    TAG 5
    DO I = IWMIN, IWMAX                   !loop com funcoes matematicas
       :
    END DO
      :
    TAG 6
    DO I = IWMIN, IWMAX                   !loop de FFTC
       :
    END DO
180 CONTINUE
```

Figura 1: Esquema do corpo principal do programa de aplicação SPLIT-STEP

Desempenho da Versão Serial

O ponto de partida foi colocar o programa SPLIT-STEP serial para ser executado com instrumentação em sete diferentes partes do programa. Com isso, pudemos ver o tempo consumido em cada parte, relativo a cada categoria de *overhead*. A tabela 1, a seguir, ilustra os tempos obtidos para o conjunto de predicados de desempenho citado anteriormente na seção 2 e inclui o fator que indica a quantidade de computação pura RT (**Remaining Time**) e o tempo total de processamento TT. Estes tempos apresentados na tabela 1 são a média de duas execuções. Todos os tempos são dados em segundos. Esse programa e todos os experimentos restantes foram executados e os dados de *predicate profiling* coletados usando a ferramenta pp.

	TAG 0 MODSSA	TAG 1 DZ	TAG 2 W	TAG 3 KX	TAG 4 FFTC	TAG 5 IW	TAG 6 FFTC
LI	0	0	0	0	0	0	0
IP	0	0	0	0	0	0	0
SL	0	0	0	0	0	0	0
CL	0.007	0.004	0.003	0.0009	0.0002	0.00008	0.0004
RC	0.001	0.0009	0.0009	0.0004	0.00002	0.00003	0.00001
TT	337.01	332.01	113.73	79.99	74.56	67.54	75.08
RT	337.01	332.01	113.73	79.99	74.56	67.54	75.08

Tabela 1: SPLIT-STEP versão serial

Na tabela 1 podemos observar que existem frações de tempo insignificantes devido a perdas na movimentação dos dados através do sistema e por contenção nos recursos.

Desempenho da Versão Paralela

Em seguida, foram inseridas no programa fonte diretivas de paralelização em cada um dos quatro *loops* escolhidos para serem paralelizados. A tabela 2 mostra os tempos da paralelização do *loop* de ômega relativos a cada categoria de *overhead*. Foi escolhido para os testes de paralelização um número de processadores igual a oito. Os tempos são dados em segundos e cada categoria representa o somatório de todos os tempos gastos em cada processador. A esta tabela, correspondente aos dados coletados com a ferramenta pp, foi adicionada uma linha com o tempo médio para um processador (PT - **Processor Time**).

	TAG 0 MODSSA	TAG 1 DZ	**TAG 2 W**	TAG 3 KX	TAG 4 FFTC	TAG 5 IW	TAG 6 FFTC
LI	20.58	20.58	**20.58**	0	0	0	0
IP	1645.39	1610.20	**5.46**	0	0	0	0
SL	0	0	**0**	0	0	0	0
CL	11.92	11.91	**6.22**	4.79	4.67	0.15	0.15
RC	2.98	2.29	**1.15**	0.52	0.45	0.10	0.10
TT	2060.48	2020.26	**186.27**	169.70	81.23	68.68	75.93
RT	380.28	375.26	**152.88**	164.37	76.11	68.43	75.67
PT	257.56	252.53	**23.28**	21.21	81.23	68.68	75.93

Tabela 2: Paralelização do *loop* de ômega com 8 processadores

A partir da tabela 2 podemos construir a tabela 3, que ilustra as porcentagens de cada categoria de *overhead* em relação ao tempo total.

Na tabela 3 podemos observar que 79.85% do tempo total de processamento são gastos devido à insuficiência de paralelismo e 18.45% pela computação pura. As porcentagens relativas as outras categorias de *overhead* podem ser consideradas irrelevantes em relação as duas citadas. O *speedup* foi 1.30 a a eficiência relativa igual a 16.35%.

	TAG 0 MODSSA	%
LI	20.58	0.99
IP	1645.39	79.85
SL	0	0
CL	11.92	0.57
RC	2.98	0.14
RT	380.28	18.45

Tabela 3: Porcentagens das categorias de *overhead* com paralelização no *loop* de ômega

A tabela 4 mostra os tempos com paralelização nos *loops* de ômega e o primeiro *loop* com FFT complexa. A tabela 5 apresenta os percentuais em relação ao tempo total.

	TAG 0 MODSSA	TAG 1 DZ	**TAG 2 W**	TAG 3 KX	**TAG 4 FFTC**	TAG 5 IW	TAG 6 FFTC
LI	28.09	28.09	**26.10**	0	**1.98**	0	0
IP	1122.95	1084.66	**4.68**	0	**7.80**	0	0
SL	0	0	**0**	0	**0**	0	0
CL	12.15	12.14	**6.27**	4.80	**0.30**	4.67	0.16
RC	2.37	2.37	**1.10**	0.49	**0.18**	0.44	0.11
TT	1543.31	1499.55	**188.29**	188.65	**85.95**	75.17	75.01
RT	377.74	372.28	**150.12**	183.34	**75.63**	70.05	74.74
PT	192.87	187.44	**23.52**	23.58	**10.74**	75.17	75.01

Tabela 4: Paralelização do *loop* de ômega e do *loop* de FFT complexa com 8 processadores

	TAG 0 MODSSA	%
LI	28.09	1.82
IP	1122.95	72.76
SL	0	0
CL	12.15	0.78
RC	2.37	0.15
RT	377.74	24.47

Tabela 5: Porcentagens das categorias de *overhead* com paralelização no *loop* de ômega e no primeiro *loop* de FFT complexa

Na tabela 5 observamos que aumentou o paralelismo. A insuficiência de paralelismo reduziu-se de 1645.39 segundos para 1122.95 segundos. Do tempo total de

processamento são gastos nela 72.76%. A computação pura consumiu 24.47% e o desbalanceamento de carga 1.82% do tempo total. O *speedup* aumentou para 1.74 e a eficiência relativa foi igual a 21.84%.

A tabela 6 mostra os tempos com paralelização nos *loops* de ômega e nos dois *loops* com FFT complexa.

	TAG 0 MODSSA	TAG 1 DZ	**TAG 2 W**	TAG 3 KX	**TAG 4 FFTC**	TAG 5 IW	**TAG 6 FFTC**
LI	21.18	21.18	**17.20**	0	**1.74**	0	**2.21**
IP	609.10	571.18	**6.06**	0	**6.08**	0	**8.00**
SL	0	0	**0**	0	**0**	0	**0**
CL	12.33	12.30	**1.95**	0.75	**0.27**	4.65	**4.79**
RC	2.15	2.15	**0.78**	0.21	**0.15**	0.39	**0.37**
TT	1033.81	990.47	**184.74**	169.01	**83.41**	75.45	**92.57**
RT	389.03	383.65	**158.72**	169.03	**75.13**	70.41	**77.18**
PT	129.22	123.80	**23.09**	21.12	**10.42**	75.45	**11.57**

Tabela 6: Paralelização do *loop* de ômega e dos dois *loops* com FFT complexa

A tabela 7 relativa a paralelização do *loop* de ômega e nos dois *loops* de FFT complexa ilustra as porcentagens de cada categoria de *overhead* em relação ao tempo total.

	TAG 0 MODSSA	%
LI	21.18	2.04
IP	609.10	58.91
SL	0	0
CL	12.33	1.19
RC	2.15	0.20
RT	389.03	37.63

Tabela 7: Porcentagens das categorias de *overhead* com paralelização no *loop* de ômega e nos dois *loops* de FFT complexa

Com a paralelização de mais um *loop*, podemos ver a mesma tendência de comportamento na tabela 7. Diminuiu o tempo consumido com insuficiência de paralelismo 58.91%, aumentou o desbalanceamento de carga 2.04% e a computação pura 37.63%. A perda por comunicação foi 1.19% do tempo total. O *speedup* aumentou para 2.60 e eficiência foi de 32.60%.

Finalmente na tabela 8 podemos ver o resultado de tempos dos quatro *loops* escolhidos para paralelização.

	TAG 0 MODSSA	TAG 1 DZ	**TAG 2 W**	TAG 3 KX	**TAG 4 FFTC**	**TAG 5 IW**	**TAG 6 FFTC**
LI	26.48	26.48	**21.15**	0	**1.90**	**1.62**	**1.80**
IP	92.80	54.26	**5.58**	0	**9.38**	**6.64**	**7.74**
SL	0	0	**0**	0	**0**	**0**	**0**
CL	3.58	3.55	**1.88**	0.72	**0.32**	**0.32**	**0.31**
RC	1.87	1.87	**0.75**	0.20	**0.20**	**0.19**	**0.20**
TT	526.17	482.12	**202.07**	183.43	**88.07**	**77.00**	**86.50**
RT	401.41	395.94	**172.69**	182.50	**76.26**	**68.29**	**76.43**
PT	65.77	60.26	**25.25**	22.92	**11.00**	**9.62**	**10.81**

Tabela 8: SPLIT-STEP versão paralelizada nos quatro *loops*

Construindo a tabela 9 relativa a paralelização dos quatro *loops* podemos ver as porcentagens de cada categoria de *overhead* em relação ao tempo total.

	TAG 0 MODSSA	%
LI	26.48	5.03
IP	92.80	17.63
SL	0	0
CL	3.58	0.68
RC	1.87	0.35
RT	401.41	76.28

Tabela 9: Porcentagens das categorias de *overhead* com paralelização nos quatro *loops*

Através da tabela 9, podemos notar que 17.63% são relativos a insuficiência de paralelismo, 5.03% são por desbalanceamento de carga, 0.68% de perda por comunicação e 76.28% por computação pura. O *speedup* aumentou para 5.12 e a eficiência relativa foi de 64.05%.

Geração do Modelo de Desempenho

Para geração do modelo de desempenho da aplicação, foi utilizada a ferramenta **expgen**. O fator escolhido para ser variado foi o número de processadores p, cujos valores foram 2, 4, 8, 16, 20 e 24.

O usuário pode fornecer, também, um *replication count*, que determina o número de vezes que cada experimento é realizado. Isto é, o programa de aplicação paralelizado é executado em p processadores pelo número de vezes especificado por *replication count*. O *replication count* escolhido foi 3. Quanto maior a redundância nos experimentos maior será a precisão dos resultados.

Com os valores de variação de p, a ferramenta **expgen** gera os experimentos automaticamente. A saída gerada por **expgen** é um conjunto de *scripts* de programa, um para cada dos seis experimentos realizados correspondendo aos valores de p. Cada um destes *scripts* é executado três vezes (valor definido no *replication count*), produzindo informação de *predicate profiling* que é armazenado para uso durante a geração do modelo.

As medidas produzidas pela ferramenta **expgen** foram submetidas à ferramenta **modgen** para criar o modelo de desempenho da aplicação. Foi gerado o modelo de uma variável para cada categoria de *overhead* que captura os efeitos de variação de p. O modelo para a aplicação é simplesmente a soma destes modelos, cuja formula é

$$2.315p^2 + 366.9 - 0.1933p^2 \ \ log\ p - 4.436p + 1.478p \ \ log\ p.$$

Para cada ponto, a ferramenta **modgen** gerou um modelo de uma variável invocando a ferramenta `lca`. Essa gera um modelo para cada categoria de *overhead* ajustando os dados medidos aos modelos de *overhead* padrão. **modgen** determina quais modelos gerar, e chama `lca` para ajustar um modelo a um subconjunto dos pontos de dados medidos.

O gráfigo da figura 2 ilustra o modelo obtido para as categorias de *overhead* no trecho de programa identificado pela TAG 0.

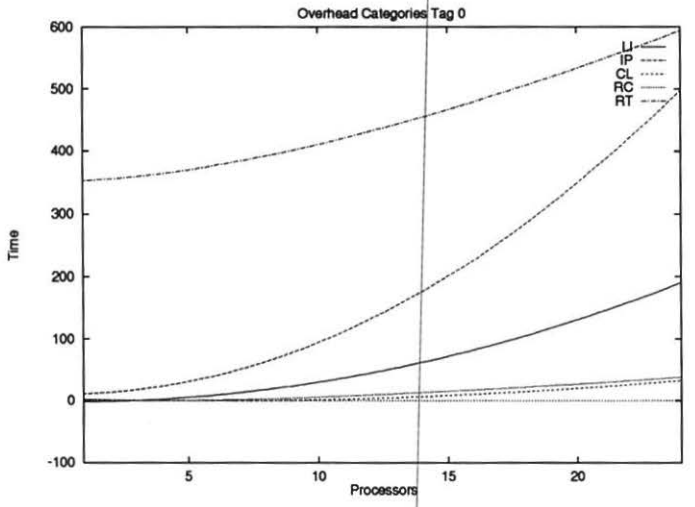

Figura 2: Modelo das Categorias de *Overhead* em Relação a TAG 0

A saída do passo de geração do modelo é produzida pela ferramenta **xmodgen**, que produz uma *interface* X para os resultados de **modgen**.

A janela *Fitting Overview*[6] desta ferramenta dá um coeficiente médio para cada modelo de uma variável representando um par de fator/categoria de *overhead*. Valores próximos a 1.0 representam modelos precisos e valores menores modelos imprecisos. Existem outras janelas nesta ferramenta que auxiliam o usuário na verificação e ajuste do modelo para visualização gráfica do comportamento do modelo de cada

categoria de *overhead*. A figura 3 mostra algumas das janelas que a ferramenta possui.

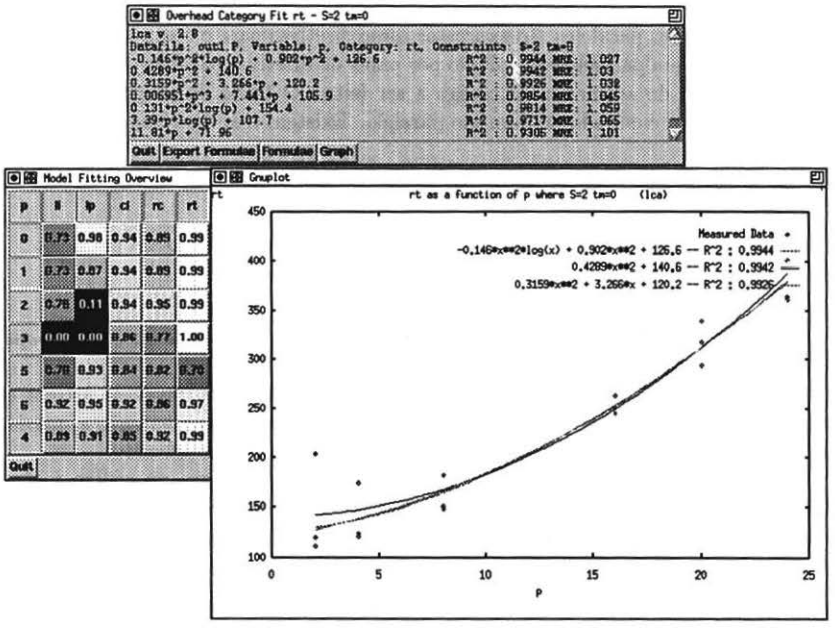

Figura 3: Visualização Gráfica do Comportamento do Modelo Gerado

4 Análise da Utilização da Ferramenta

Os comandos de instrumentação do código fonte possuem sintaxes e semânticas simples. A instrumentação é feita através de chamadas a biblioteca *pplib(3F)* que determina o *overhead* paralelo. Essas chamadas podem ser colocadas em qualquer lugar no programa, podem aparecer várias vezes e podem ser executadas múltiplas vezes durante a execução do programa. Para o programa de aplicação SPLIT-STEP foram utilizadas as seguintes chamadas:

- para determinar o *overhead* paralelo;
  subroutine pp_start_profiling_tag (itag)
  subroutine pp_end_profiling_tag (itag)

- para criar e terminar *teams*. Um *team* é um grupo de *threads* paralelas com um número de identificação;
  subroutine pp_create_team (iteam_id, num_threads)
  subroutine pp_stop_team (iteam_id)

- para indicar que o próximo *loop* será paralelizado em *iteam_id* partes. *subroutine pp_loop_start (iteam_id)*
  subroutine pp_loop_end (iteam_id)

Os construtores paralelos são expressos como diretivas para o compilador. O trabalho realizado em paralelo é envolvido por pares de diretivas paralelas que indicam o início e o fim do segmento de código a ser feito. O texto do programa dentro dessas diretivas é chamado de domínio paralelo. Existem três diretivas que são:

- *parallel regions* que executam múltiplas instâncias de um segmento de código em paralelo;
- *parallel sections* que executam múltiplos segmentos de código em paralelo;
- *tile families* que executam *loops* em paralelo.

Na aplicação SPLIT-STEP foi utilizado somente o construtor *tile* em quatro *loops* diferentes.

As informações sobre o *Lost Cycles Toolkit* foram obtidas através de artigos e de manuais. Para a utilização prática das ferramentas, sua documentação ajudaria muito se fosse mais elaborada com exemplos para agilizar o aprendizado.

De modo geral, não houve problemas em inserir as diretivas no código fonte. Porém, ainda são diretivas onde o programador necessita de um conhecimento sobre a aplicação que esta sendo paralelizada para inserí-las nos lugares apropriados.

Houve uma tentativa de realizar a paralelização do código com níveis diferentes de paralelismo, utilizando o construtor *parallel regions*. Com o pouco de tempo de utilização da ferramenta não foi possível detectar o erro no uso deste construtor. Foi observado erro na geração do arquivo de eventos.

Uma observação relevante é que no código não devem ser inseridas diretivas de criação de *teams* que jamais serão utilizados durante a execução, pois a ferramenta se perde na geração do arquivo de eventos `elog`.

Uma outra observação é que um programa executado de modo interativo pode ter problema na geração, por exemplo, dos dados de tempo. Esses aparecem com valores negativos. Este tipo de problema não ocorre quando os experimentos são submetidos utilizando-se uma fila em *background*.

Depois do programa de aplicação ter sido paralelizado, foram feitos experimentos variando-se o número de processadores. Observou-se uma discrepância nos tempos medidos para 2, 4 e 8 processadores. Das três execuções da aplicação para cada um desses valores de p, a primeira tomada de tempo diferiu muito das outras duas. A diferença ficou em cerca de 47.48% para 2 processadores, 51.19% para 4 processadores e 23.75% para 8 processadores. Este fato indica que o sistema pode variar muito na sua distribuição de processos. Deveriam ter sido feitas uma quantidade de medidas muito maior que três execuções para que os resultados pudessem ser validados com maior precisão. Outras investigações tem sido feitas para se verificar porque este fato ocorreu.

Esta mesma aplicação foi submetida anteriormente a outras máquinas paralelas: iPSC/860 da Intel, IBM 3090/600-6VF da IBM e no protótipo NCP I8 da COPPE/UFRJ; cujos resultados foram muito melhores do que aquele obtido na Kendal Square Research KSR1. Nas máquinas citadas a eficiência ficou acima de

90% e na KSR1 não ultrapassou 65%. Este fato também tem sido investigado para se chegar as possíveis causas. Uma delas é o modelo de programação da KSR1, o modo como ele é implementado. Como os *loops* paralelizados são internos a um *loop* que é executado por 100 vezes, a criação e destruição dos processos causam um grande *overhead.*

As informações dadas pela ferramenta são úteis, pois com elas podemos obter noções de como os segmentos do programa podem ser particionados, se em partes maiores ou menores para se obter um melhor desempenho.

No modelo de desempenho da aplicação, se a fórmula resultante gerada tiver muitos termos, pode levar a perda de seu significado físico. Se ainda necessitar de ajustes na precisão da expressão matemática, isso pode ser uma tarefa complexa. O ajuste interativo permitido pela ferramenta supõe um conhecimento prévio da aplicação que talvez o programador não possua intuitivamente.

A *interface* gráfica é simples de ser utilizada e útil na visualização das curvas de desempenho obtidas.

5 Conclusões

A predição de desempenho pode ajudar o programador na tarefa de ajustar e avaliar a escalabilidade e o desempenho de programas paralelos. Contudo, é difícil medir todas as fontes potenciais de ineficiência em programas paralelos.

O *Lost Cycles Toolkit* foi desenvolvido para tornar possível a predição de desempenho pelos programadores de sistemas paralelos. Seus pontos positivos são tentar automatizar o processamento de modelamento o máximo possível, possuir uma ferramenta gráfica interativa de fácil uso e dar informações relevantes ao ajuste de escalabilidade. Como pontos ainda a serem melhorados são aumentar os padrões *default* para ajuste do desempenho de cada categoria de *overhead*, de modo a simplificar a participação do programador, ver uma forma onde o significado físico do modelamento não seja perdido em uma expressão matemática e resolver os problemas relativos a utilização interativa que pode provocar problemas na geração dos tempos, investigar as possíveis causas para os diferentes tempos obtidos quando da realização dos experimentos para avaliação de desempenho e investigar também causas para as diferenças de desempenho em relação a outras máquinas paralelas como iPSC/860, IBM 3090/600-6VF e o protótipo NCP I8.

Agradecimentos

Agradecemos ao professor Thomas J. LeBlanc do Departamento de Ciência da Computação da Universidade de Rochester pelo seu convite para visitar esta instituição e pela oportunidade de desenvolvimento deste trabalho.

Referências

[1] Clement, Mark J. and Quinn, Michael J., 'Àlytical Performance Prediction on Multicomputers", Proceedings of Supercomputing'93, pp 886-894, November 1993.

[2] Pease, D., Ghatoor, A., Ahmad, I., Andrews, D., Foudil-Bey, K., Karpinski, T., Mikki, M. and Zerrouski, M.,"Paws: A Performence Evaluation Tool for Parallel Computing Systems", IEEE Computer, pp 18-29, January 1991.

[3] Zimran, E., Rao, M. and Segall, Z.,"Performance Efficient Mapping of Applications to Parallel and Distributed Architectures", Proceedings of the 1990 International Conference on Parallel Processing, pp II-147-II-154, August 1990.

[4] Crovella, Mark E. and LeBlanc, Thomas J., "Parallel Performance Prediction Using the Lost Cycles Analysis", Proceedings Supercomputing'94, November, 1994, pp. 600-609.

[5] Crovella, Mark E., "Performance Prediction and Tuning of Parallel Programs", Department of Computer Science, University of Rochester, PhD Dissertation, TR 573, August 1994.

[6] Crovella, Mark E., LeBlanc, Thomas J. and Meira Jr., Wagner, "Performance Measurement and Modeling with the Lost Cycles Toolkit", Department of Computer Science, University of Rochester, Technical Report, TR 580, April 1995.

[7] Crovella, Mark E., LeBlanc, Thomas J. and Meira Jr., Wagner, "The Statistical Foundations of Lost Cycles Analysis", Department of Computer Science, University of Rochester, November, 1994.