Ferramentas de Modelagem para a Predição de Performance Analítica em uma Plataforma de Processamento Paralelo

Roberto Hirochi Herai Marco Aurélio Amaral Henriques
{*rherai* | *marco*}*@dca.fee.unicamp.br*

Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia da Computação e Automação Industrial
Campinas, São Paulo, Brasil

Resumo

A predição de performance é um importante mecanismo para avaliar a utilização de recursos e estimar o tempo de execução de aplicações em sistemas de processamento paralelo. Este trabalho apresenta ferramentas que permitem gerar modelos de performance sem exigir o uso de linguagens especiais (não padronizadas) e nem de marcações específicas no código-fonte. As ferramentas consideram atrasos de computação e comunicação, bem como os causados pela contenção no uso de recursos compartilhados. Os modelos gerados podem ser combinados para realizar a predição de performance de uma aplicação sob diferentes situações no sistema paralelo.

1. Introdução

A predição de performance é um instrumento importante para a análise de aplicações paralelas de grande porte, especialmente aquelas voltadas para sistemas maciçamente paralelos. Em vista das grandes dimensões desses sistemas, um baixo custo nas predições é essencial para permitir seu uso de forma prática [1]. Técnicas analíticas apresentam essa característica e são uma alternativa eficiente para realizar a predição sob diferentes configurações do sistema e dimensões do problema [2]. Essas técnicas baseiam-se em informações estáticas do sistema paralelo (velocidade das máquinas e do enlace de comunicação) e da aplicação (operações realizadas pelo código) [3]. Essas informações, na maioria das vezes, são armazenadas em modelos que seguem uma determinada notação. Tais modelos representam os componentes fundamentais para a predição de performance de aplicações paralelas e, por isso, devem adotar uma abordagem eficiente na representação das partes envolvidas [4]. Algumas propostas de ferramentas de predição baseadas em técnicas analíticas abordam o problema da criação dos modelos dividindo-o em duas partes: modelo da aplicação e modelo de máquina.

Um dos problemas envolvidos na geração do modelo da aplicação é a necessidade da inserção de marcadores explícitos no código-fonte da mesma. Sistemas como PACE [5] e PAMELA [6] exigem que isso seja feito. As aplicações para PACE podem ser escritas em Fortran (77 ou 90) ou C. PAMELA propõe a criação de uma nova linguagem de programação, chamada Spar [7] que procura modelar efeitos de atraso causados pela contenção no uso de recursos. A ferramenta PerPreT [8] também propõe o uso de uma nova linguagem de programação, chamada LOOP, que limita as aplicações àquelas descritas pelo modelo de programação SPMD (*Single Program Multiple Data*). Neste modelo, o mesmo código é carregado em todas as máquinas para realizar a mesma tarefa sobre os seus diferentes dados [9]. Porém, PerPreT não exige marcação explícita no código.

Outra questão importante está relacionada aos tipos de dados e às operações realizadas sobre eles na geração dos modelos através da análise do código-fonte das aplicações desenvolvidas. Os sistemas PACE, PAMELA e PerPreT, por exemplo, consideram apenas operações sobre tipos de dados em ponto flutuante na geração do modelo da aplicação.

O processo de geração do modelo de máquinas para cada ferramenta mencionada é baseado na execução de algoritmos de *benchmark* (algoritmos de testes) no sistema paralelo, com o intuito de avaliar a performance da rede e das máquinas que o compõem.

Como visto, muito trabalho vem sendo feito para criar modelos que representam importantes características das aplicações e do sistema envolvido. E muitas ferramentas destinadas a isso exigem substancial esforço manual para representar as aplicações no formato do modelo desejado.

Dentro deste contexto, o presente trabalho apresenta duas novas ferramentas, as quais foram desenvolvidas para

a geração de modelos de performance de aplicações construídas com uma linguagem padrão (Java) e sem exigência de marcação especial no código-fonte. A primeira ferramenta, o J-CRES (*JoiN Computational Resource Evaluation Service*), realiza a avaliação dos recursos computacionais do sistema paralelo e cria o modelo de máquina. A segunda, J-ARC (*JoiN Analytical Representation Compiler*), cria um modelo analítico da aplicação paralela em função das operações de computação e comunicação presentes no código da aplicação.

Para implementação, testes e avaliação das ferramentas, foi escolhido o sistema JoiN [10][11], uma plataforma para o processamento paralelo de aplicações em ambientes heterogêneos distribuídos. Por este motivo, as ferramentas compartilham algumas das características de JoiN tais como adoção de um modelo de paralelismo de dados e o uso da linguagem Java.

O restante deste trabalho é organizado da seguinte forma. A seção 2 define as características do sistema paralelo no qual as ferramentas são aplicáveis. A seção 3 detalha a técnica de modelagem de performance utilizada pelas ferramentas para geração de seus modelos. A seção 4 apresenta as duas ferramentas propostas e a seção 5, as conclusões.

2. Características do Sistema Paralelo

Para a geração dos modelos de performance pelas ferramentas mencionadas, é necessário definir antes as características do sistema paralelo no qual elas são aplicáveis.

A estrutura do sistema paralelo permite definir quais máquinas farão parte do modelo de máquinas. E, a partir da estrututura das aplicações e da técnica de escalonamento de tarefas, é possível criar um modelo da aplicação que descreva fatores de atraso presentes na execução real de alguma aplicação.

2.1. Estrutura do Sistema

A estrutura do sistema é formada por quatro tipos de elementos (Fig. 1):

- um nó de entrada (servidor): atua como primeiro ponto de contato de novos computadores que entram no sistema para atuar como nós trabalhadores;
- um ou mais nós de administração: envia requisições de execução de aplicações ao nó servidor;
- vários nós de coordenação de grupo: responsáveis pelo monitoramento, divisão, distribuição e coleta do resultado das tarefas;
- vários nós de processamento (nós trabalhadores): dedicados exclusivamente ao processamento das tarefas enviadas pelo nó coordenador.

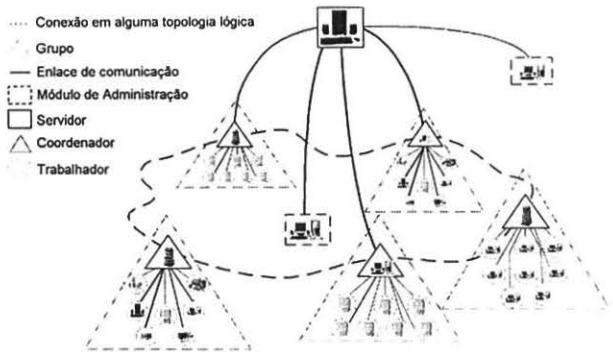

Figura 1. Estrutura do sistema paralelo

Um grupo no sistema é formado por um coordenador e por vários trabalhadores. Os grupos são independentes, interconectados por alguma topologia lógica (hipercubo, anel, árvore) e seus nós, chamados de unidades de processamento, têm a capacidade de executar diferentes aplicações sobre blocos de dados em linhas de execução (*threads*) independentes.

Unidades trabalhadoras só podem comunicar-se com suas respectivas unidades coordenadoras, as quais são capazes de se comunicar com outras unidades coordenadoras e com o servidor. O módulo de administração comunica-se apenas com o servidor.

2.2. Estrutura das Aplicações

A estrutura das aplicações, que segue a definição apresentada em [12], não se restringe ao modelo SPMD. Ela é formada por um conjunto de A lotes de tarefas e por um conjunto de B relações de dependência de dados entre os lotes.

A cada lote de tarefas L_{i,c_i}, está associado um nível indicado pelo índice i ($i = 1..A$) e uma cardinalidade c_i, definida como sendo o número de tarefas no nível i. Uma tarefa de um lote é definida como uma seqüência ordenada de operações aritméticas, matemáticas, de desvio condicional ou de atribuição, entrada ou saída de dados, que implementa algum tipo de processamento finito e determinístico sobre um bloco de dados de entrada a fim de produzir um bloco de dados de saída [12].

Define-se também que os lotes de tarefas L_{1,c_1} e L_{A,c_A} possuem cardinalidade igual a um e são responsáveis pelo processamento do bloco inicial e final da aplicação, respectivamente.

Um bloco de dados de saída de uma tarefa só pode fazer parte de um bloco de dados de entrada de uma tarefa de nível posterior.

Para especificar uma aplicação paralela, é utilizada a linguagem PASL (*Parallel Application and Specification Lan-*

guage) [12]. Ela é composta por 2 seções funcionais, como mostra a Fig. 2.

```
//seção de declarações
path=aplicacoes/numerosPrimos
T1=DivideDados.class;
T2=ProcessaDados.class;
T3=ColetaResultados.class;

//seção de ligações
T1=L1(1)«dados.dat;
T2=L2(100)«T1;
T3=L3(1)«T2»resultado.dat;
```

Figura 2. Especificação de uma aplicação paralela em PASL

- declarações: indicam os caminhos das classes e dos dados que serão utilizados pela aplicação. No exemplo, a variável *path* indica o diretório e $T1, T2, T3$ as classes utilizadas pela aplicação;
- ligações: indicam os fluxos de dados entre os lotes de tarefas, sua cardinalidade e os arquivos de entrada e saída da aplicação. No exemplo tem-se que $T1$, que compõe o lote $L1$ com cardinalidade 1, recebe de entrada o arquivo *dados.dat* para processamento. A tarefa $T2$, que compõe o lote $L2$ com cardinalidade 100, lê a saída produzida pelo processamento de $T1$. Finalmente a tarefa $T3$, que faz parte do lote $L3$ com cardinalidade 1, lê a saída de $T2$ e escreve o resultado final no arquivo *resultado.dat*.

2.3. Escalonador de Tarefas

Define-se escalonador de tarefas como sendo o componente de um sistema de processamento paralelo responsável pela alocação das tarefas de uma aplicação às unidades de processamento, de forma a equilibrar a carga de trabalho.

Pode ser assumido que lotes de tarefas que apresentam cardinalidade igual a um são alocados diretamente na unidade coordenadora (para minimizar tempo de comunicação) e que os lotes de tarefas com cardinalidade maior que um são alocados nas unidades trabalhadoras de forma proporcional ao seu fator de performance relativo W_p ($p = 1..P$, onde P é o número de unidades trabalhadoras).

Para calcular o fator de performance entre as unidades trabalhadoras, podem ser executados programas de *benchmark* em cada uma delas, dividindo-se o tempo gasto pela unidade p pelo tempo da unidade mais rápida de seu grupo. A distribuição de tarefas de um lote entre as P unidades trabalhadoras pode ser feita de forma que o número de tarefas (S_p) por unidade (p) seja proporcional ao fator de performance da unidade (W_p) [12].

3. Técnica de Modelagem de Performance

Um mecanismo que pode ser utilizado para a geração dos modelos de performance é aquele que segue a abordagem adotada na ferramenta PAMELA [6]. Esta ferramenta implementa um modelo de mesmo nome, apresentado em [2], mas exige o uso de uma linguagem de programação não padronizada, além da colocação de marcadores especiais no programa.

PAMELA propõe um mecanismo de modelagem considerando efeitos de contenção, onde são criados 3 modelos (Fig. 3). O primeiro é referente ao modelo da aplicação paralela, o segundo referente ao modelo das máquinas e da rede que compõe o sistema paralelo, e o último equivalente à combinação dos dois modelos anteriores para produzir tempos de predição. A qualidade dos dados fornecidos por PAMELA pode ser melhorada com a adoção de intervalos [13], resultando em um sistema conhecido como PAMELA Estendido [11].

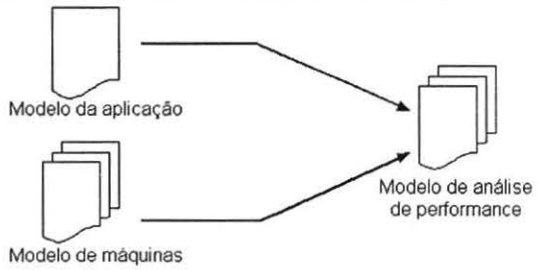

Figura 3. Modelo de performance

Na técnica de intervalos, são selecionados um tempo mínimo e máximo ($[t_{min}, t_{max}]$) de uma lista de medições para um mesmo *benchmark* em uma unidade de processamento. Seu uso permite criar intervalos que refletem de forma mais adequada a variação de desempenho dos recursos analisados.

Neste trabalho é proposta a geração dos dois primeiros modelos de performance através de duas ferramentas, devido à sua grande importância na representação das características de performance de um sistema paralelo. Para isso, é feita uma breve descrição da notação utilizada para a geração de cada modelo.

3.1. Modelo da Aplicação

Uma notação que pode ser utilizada para a geração do modelo da aplicação é a da linguagem PAMELA. Esta linguagem possui um conjunto de operadores que permite modelar uma aplicação paralela em relação a atrasos gerados por processos de comunicação e computação presentes em seu código.

A seguir é apresentado um conjunto reduzido de operadores básicos da linguagem com o intuito de ilustrar como funciona o seu processo de modelagem.

- $delay(\tau)$: representa o atraso de tempo τ para algum tipo de operação realizada pela aplicação;
- ; : operador binário de execução de processos seqüenciais;
- ||: operador binário de execução de processos em paralelo;
- $seq(j = a, b)Op_j$: operador de redução seqüencial, equivalente a $Op_a; ...; Op_b$;
- $par(j = a, b)Op_j$: operador de redução paralelo, equivalente a $Op_a||...||Op_b$;
- $use(R, \tau)$: operador que indica o uso de um determinado recurso R por um tempo τ, para representar a contenção no uso do recurso no modelo da aplicação. O uso de um recurso está associado a um fator de contenção (F_R), denotando a quantidade de processos que utilizam-no simultaneamente (em paralelo).

Os operadores PAMELA definidos acima são suficientes para modelar uma ampla gama de aplicações paralelas e efeitos de atraso que influenciam no tempo de execução das aplicações.

3.2. Modelo de Máquinas

O modelo de máquinas procura descrever as características de computação e comunicação de um sistema paralelo heterogêneo. O modelo procura refletir, na forma de intervalos, a variação de performance de cada unidade de processamento na execução de operações de baixo nível sobre determinados tipos de dados. As características de comunicação são analisadas pela velocidade média no envio de pacotes de uma unidade de processamento a outra.

Sendo assim, o modelo de máquinas pode ser definido pelos parâmetros:

$opName\_dataType_p = delay(inter\_opName\_dataType_p)$
e
$send_{p,q}(dataSize) = delay(inter\_send_{p,q} \times dataSize)$.

O parâmetro $inter\_opName\_dataType_p$ é um intervalo de valores representando o tempo gasto na execução de operações *opName* (soma, subtração, multiplicação, divisão, etc.) sobre determinados tipos de dados *dataType* (inteiro, ponto flutuante, caracter, etc.) na unidade de processamento p, $0 \leq p \leq P$, sendo $P + 1$ o número total de unidades de processamento (1 coordenadora e P trabalhadoras). A operação $inter\_send_{p,q} \times dataSize$ indica o tempo gasto na execução da operação $send_{p,q}$ no envio de dados de tamanho $dataSize$ da unidade de processamento p para a q.

A partir dos modelos definidos, pode-se então criar uma descrição precisa da aplicação paralela e um modelo de máquinas que represente as características de performance do sistema paralelo.

A próxima seção apresenta as ferramentas desenvolvidas neste trabalho. Elas são responsáveis pela geração do modelo da aplicação e do modelo de máquinas mencionados utilizando a notação descrita nesta seção.

4. Proposta de Ferramentas para Geração de Modelos de Performance

Esta seção apresenta as ferramentas J-CRES (*JoiN Computational Resource Evaluation Service*) e J-ARC (*JoiN Analytical Representation Compiler*) que foram desenvolvidas para a geração de modelos de performance. Elas foram baseadas em Java e elaboradas a partir do modelo da plataforma JoiN, uma plataforma cuja arquitetura e modelo de tarefas são aqueles mostrados nas Figs. 1 e 2 respectivamente.

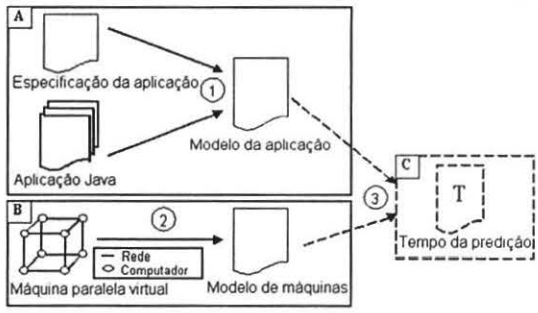

Figura 4. Etapas do processo de geração dos modelos de performance

As duas ferramentas apresentam uma funcionalidade bem definida (Fig. 4), na qual a primeira (quadro A da figura) é responsável pela análise da estrutura da aplicação e de sua especificação, e a segunda (quadro B da figura) pela avaliação de desempenho de todos os recursos disponibilizados no sistema paralelo para a execução de aplicações.

Com os dois modelos disponíveis, pode-se estimar o tempo de execução que a aplicação levará para ser executada (quadro C da figura). A geração de cada modelo envolve algumas etapas (marcadas por números na figura):

1. análise do código da aplicação em Java e da sua especificação paralela em PASL através da ferramenta J-ARC, para a geração do modelo da aplicação;
2. avaliação dos recursos computacionais do sistema paralelo, através de algoritmos de *benchmark* executados

pela ferramenta J-CRES em cada unidade de processamento, para a geração do modelo de máquinas;

3. combinação do modelo de máquinas com o modelo da aplicação para a geração das predições.

A seguir, cada uma das duas ferramentas será apresentada em detalhes.

4.1. Modelo da Aplicação com J-ARC

A estrutura da ferramenta J-ARC é apresentada na Fig. 5, equivalente ao quadro A da Fig. 4.

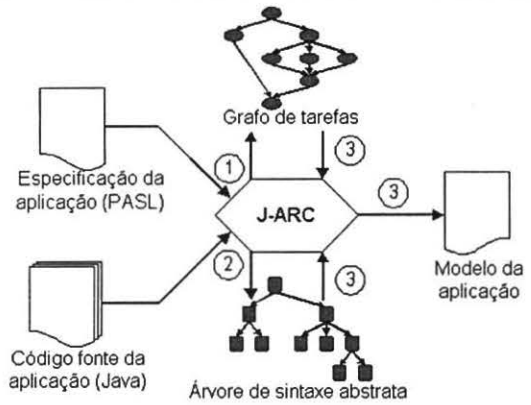

Figura 5. Estrutura da ferramenta J-ARC

O processo de geração do modelo da aplicação é composto por três etapas (indicado por números na Fig. 5) e cada uma delas é descrita a seguir.

4.1.1. Geração da Representação Intermediária da Linguagem PASL

O primeiro passo (etapa 1 na figura) é a criação de uma representação intermediária do código PASL, na forma de um grafo de tarefas, conforme a estrutura de aplicações definida. A representação possui a estrutura de um grafo conectado, acíclico e direcionado, com os nós representando os lotes de tarefas e as conexões, as relações de dependência de dados entre elas.

Por exemplo, para a especificação da aplicação da Fig. 2, tem-se uma representação em grafo de tarefas equivalente à Fig. 6. A estrutura na forma de um grafo de tarefas é construída por um compilador feito especialmente para analisar a linguagem PASL.

Cada nó do grafo é composto por alguns atributos como cardinalidade, identificação do lote e as conexões com outros nós para identificar a origem/destino dos dados processados pela tarefa. A partir da representação criada, pode-se identificar como as tarefas serão alocadas no sistema paralelo e o fluxo de dados entre elas sem ser necessário o uso de marcadores especiais dentro do código-fonte.

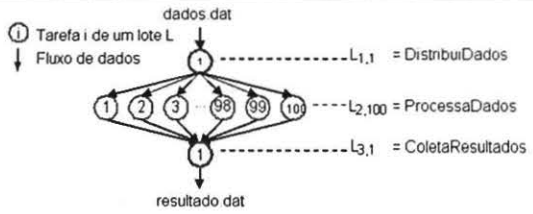

Figura 6. Grafo de tarefas

4.1.2. Geração da Representação Intermediária do Código da Aplicação

Após a análise da especificação da aplicação, é feita uma verificação do grafo de tarefas gerado para identificar as classes que serão executadas pelas tarefas (etapa 2 da Fig. 5).

A análise do código-fonte relativo a estas classes visa gerar a parte do modelo correspondente à execução das tarefas. Ela é feita por um compilador específico no qual o código é analisado e armazenado em uma estrutura de dados que permite um acesso eficiente ao mesmo. O desenvolvimento do compilador foi feito com base nas ferramentas JLex (análise léxica) e Cup (análise sintática) para a geração de um parser em Java [14].

Deve ser notado que a análise feita diretamente sobre o código-fonte da aplicação em Java, não exige o porte do mesmo para uma linguagem não padronizada, como fazem outras implementações de PAMELA [6][7].

A análise léxica verifica as cadeias de símbolos do código da aplicação, a partir de uma estrutura léxica da linguagem a ser analisada. Ela faz a identificação de cadeias de caracteres a partir do código a ser analisado, detectando se são palavras reservadas, constantes numéricas, operadores matemáticos, etc.

A análise sintática é feita por um analisador sintático em Java a partir da gramática pré-estabelecida para esta linguagem. Este analisador, também conhecido como parser, utiliza como entrada a saída produzida pelo analisador léxico referente a análise do código da aplicação.

O resultado destas análises é a criação de uma estrutura de dados que representa o código-fonte da aplicação na forma de uma árvore de sintaxe abstrata (*Abstract Syntax Tree - AST*), como mostrado na Fig. 7. A AST representa a estrutura sintática do código-fonte original e armazena muitas informações para analisar e até reconstruir o código.

A AST é projetada tal que os blocos do código são agrupados em uma única sub-árvore, com um nó raiz usado para reconhecer a estrutura abaixo dela.

Para cada classe presente no grafo, é criada uma representação intermediária na forma de uma outra AST que, apesar de apresentar-se como uma estrutura bastante extensa para um simples trecho de código, mapeia todas as informações do mesmo na árvore mencionada.

A Fig. 7 ilustra um exemplo de código com sua respectiva representação na forma de uma AST, que é obtida através do compilador criado especialmente para este fim. Ela permite analisar qualquer aplicação escrita na linguagem Java, que se restrinja à gramática definida em [15]. Esta geração requer que todo o código-fonte das classes utilizadas pela aplicação esteja disponível. A automatização

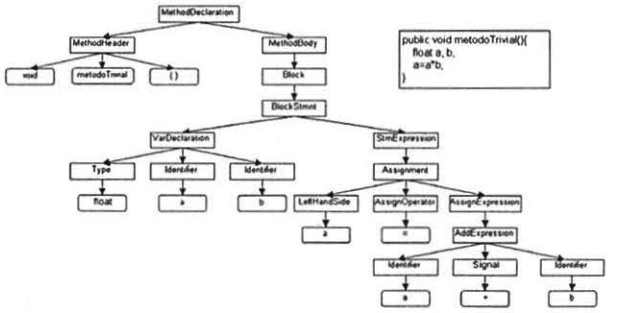

Figura 7. Árvore de sintaxe abstrata

deste processo garante que o programa de entrada seja totalmente analisado, gerando para cada classe da aplicação uma representação na forma de uma AST. Isso torna possível uma navegação na hierarquia da estrutura criada e, a partir de qualquer parte identificada, pode-se capturar informações relativas ao custo de blocos de computação e comunicação do código da aplicação, relativos ao atraso causado pela sua execução no sistema paralelo. Essas informações são utilizadas na geração do modelo da aplicação.

4.1.3. Geração do Modelo da Aplicação

Para a criação do modelo da aplicação pela ferramenta J-ARC, foi desenvolvido um outro compilador que, a partir das representações intermediárias (árvore e grafo de tarefas), coleta informações do código de uma aplicação paralela (etapa 3 da Fig. 5). Em particular, são extraídos dois tipos de informação:

- estrutura paralela do programa: inclui porções sequenciais de computação na forma de tarefas, mapeamento das tarefas em *threads* ou em diferentes processadores e fluxo de informações no sistema paralelo durante a execução das aplicações;
- estimativa de custo computacional: equivale ao tempo estimado de trechos específicos de código.

A coleta das informações da estrutura paralela é feita sobre a análise do grafo de tarefas correspondente à especificação da aplicação em PASL. Esta coleta tem por objetivo gerar a parte do modelo relacionada à execução em paralelo dos lotes de tarefas e das tarefas que compõem cada um dos lotes. O mapeamento das tarefas e a comunicação gerada no sistema são definidos em função da cardinalidade de cada lote, que indicará a forma como elas serão escalonadas no sistema paralelo, bem como a quantidade de tarefas (S_p) alocadas em cada unidade trabalhadora, como explicado na seção 2.3.

As informações relativas à estimativa de custo computacional são obtidas a partir de operações realizadas sobre dados do tipo inteiro, ponto flutuante (simples e de precisão dupla) e caracter, bastando indicar ao compilador o código da classe da aplicação e selecionar qual método será analisado.

A Fig. 8 ilustra um exemplo simples de geração do modelo de uma aplicação pela ferramenta J-ARC, onde as informações relativas à estrutura paralela e à estimativa de custo computacional de uma aplicação são coletadas.

Na figura, o trecho de código à esquerda (método $metodoSimples$) representa o processamento a ser realizado por uma tarefa do lote $L2$ e a parte direita representa o modelo da aplicação cuja especificação paralela é descrita pela Fig. 2. O código executado pelos lotes ($L1$ e $L3$) é omitido para simplificar o exemplo.

O modelo da aplicação é construído utilizando a notação da linguagem PAMELA, descrito na seção 3.1. O modelo gerado apresenta três parâmetros: um relativo ao tamanho (N) do problema, um relativo a quantidade de (P) máquinas envolvidas e um relativo ao número de tarefas (S_p) por máquina (seção 2.3).

No modelo, L representa o processamento sequencial dos lotes $L1$, $L2$ e $L3$. Os lotes $L1$ e $L3$ possuem cardinalidade 1 e são escalonados e processados no coordenador. Sua função é distribuir os dados pelas tarefas e coletar os resultados. O lote $L2$ possui cardinalidade igual a 100 e, desta forma, será escalonado entre as P unidades trabalhadoras.

O método $metodoSimples$ do código é mapeado no modelo com o mesmo nome para representar a estimativa de custo computacional para cada tarefa. Na estimativa, é considerada a sobrecarga da estrutura de repetição for e as operações realizadas pela tarefa.

Uma das vantagens da ferramenta J-ARC é que ela não requer a utilização de marcadores diretamente no código. Desta forma, não há trabalhos extras, tais como modificação do código-fonte, uso de novas bibliotecas específicas para a

```
...
int metodoSimples(int N){
  int res=0;
  for(int n=0;n<N;n++)
    res+ =n×n+N/res-n;
  return res;
}
...
```

(a) Código-fonte

```
L=L1;L2;L3
...
//processado no coordenador
L1=...
...
//processamento do lote L2
L2=seq(p=1,P){
  par(task=1,S_p){
    metodoSimples(N_{p,task});
    send_{p,0}(res);
  }
}
//modelo em cada trabalhador
metodoSimples(N_{p,task}){
  assign_int_p;
  seq(n=0,N_{p,task}-1){
    //sobrecarga do laço
    (N_{p,task}+1)× assign_int_p;
    N_{p,task}× cmp_int_p;
    N_{p,task}× sum_int_p;
    //operações da função
    sum_int_p;
    assign_int_p;
    multiply_int_p;
    sum_int_p;
    div_int_p;
    sum_int_p;
  }
}
//processado no coordenador
L3=...
...
```

(b) Modelo da aplicação

Figura 8. Modelo da aplicação para um código simples

marcação do código ou criação de novos arquivos de configuração para a geração do modelo. O sistema JoiN impõe a utilização de um arquivo de especificação para as aplicações paralelas desenvolvidas nele e esse arquivo pode ser visto como uma forma indireta de marcação.

A ferramenta J-ARC já foi implementada e encontra-se em fase de testes.

A seguir, será descrita a ferramenta J-CRES desenvolvida neste trabalho para a geração do modelo de máquinas, que representa uma das partes fundamentais na análise e predição de performance.

4.2. Modelo de Máquinas com J-CRES

J-CRES é parte da ferramenta responsável pela geração do modelo de máquinas das unidades de processamento do sistema paralelo. Ela é baseada na execução de *benchmarks* nas unidades coordenadora e trabalhadoras para avaliar suas capacidades de processamento. A avaliação é feita sobre operações básicas em diferentes tipos de dados e na troca de mensagens, definindo os parâmetros $opName\_dataType_p$ e $send_{p,q}$ do modelo de máquinas.

A estrutura de funcionamento do J-CRES é apresentada na Fig. 9. Considerando a execução dos *benchmarks* feita

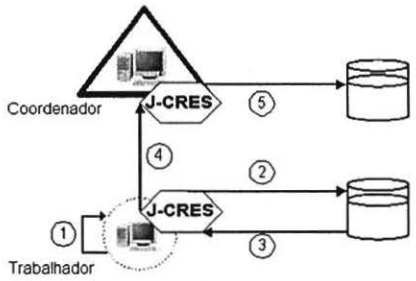

Figura 9. Estrutura funcional do J-CRES

nas unidades trabalhadoras, tem-se a seguinte ordem de execução (indicado por números na figura):

1. execução dos algoritmos de *benchmark*;
2. armazenamento dos resultados de cada operação sobre determinados tipos de dados no sistema de arquivos local;
3. determinação do intervalo de valores para cada um dos testes realizados;
4. envio dos intervalos ao coordenador;
5. atualização no coordenador dos intervalos que estão armazenados no modelo de máquinas do sistema, relativos a cada unidade de processamento.

A geração do modelo de máquinas é realizada sempre que a unidade de processamento fica ociosa. Os novos dados, para cada teste, são armazenados seguindo uma política de filas FIFO (*First In First Out*) com um tamanho limitado. Isto é, quando a quantidade de medições atinge um limite, os valores mais antigos vão sendo substituídos pelos mais novos.

Para uma determinada máquina p de um grupo, ($0 \leq p \leq P$, onde P é o número de unidades trabalhadoras e $p = 0$ é a unidade coordenadora), com K medições para cada tipo de operação realizada pelo *benchmark*, tem-se um conjunto de dados na forma $t\_opName\_dataType_p = [cp_1; cp_2; ...; cp_K]$ para operações de computação e $t\_send_{p,q} = [cm_1; cm_2; ...; cm_K]$ para operações de comunicação. A partir destes valores, é possível construir os intervalos $inter\_opName\_dataType_p = [cp_{min}; cp_{max}]$ e $inter\_send_{p,q} = [cm_{min}; cm_{max}]$

O mecanismo de geração do modelo de máquinas, que foi desenvolvido para a ferramenta J-CRES e adaptado na plataforma JoiN, não causa impactos de performance na execução das tarefas das aplicações, pois os testes de performance são feitos somente quando a unidade de processamento está ociosa.

Os algoritmos de *benchmark* utilizados na realização dos testes são específicos para a avaliação de sistemas na execução de aplicações científicas de grande porte [16], e fazem

parte do pacote Java Grande Benchmark Suite, disponível em [17].

O modelo de máquinas criado fornece informações com relação à capacidade de computação e de comunicação do sistema paralelo e o modelo da aplicação fornece informações relativas às operações de computação e de comunicação presentes no código. Essas informações permitem que os modelos sejam analisados para identificar inúmeras características da aplicação em relação à sua performance, aos efeitos causados pela contenção de recursos do sistema utilizado e ao tempo que a aplicação poderá levar para ser executada em um sistema paralelo heterogêneo. Isto possibilita que sejam avaliadas diferentes estratégias de projeto de uma aplicação, sob diferentes configurações do sistema e do tamanho do problema.

Desta forma, a próxima etapa para a predição de performance da aplicação é a combinação do modelo de máquinas com o modelo da aplicação, substituindo os parâmetros $opName\_dataType_p$ e $send_{p,q}$ pelos operadores de atraso de tempo $delay(inter\_opName\_dataType_p)$ e $delay(inter\_send_{p,q})$ respectivamente. A partir disso pode ser feita uma análise sobre os efeitos causados pela contenção de recursos com o uso do operador use (seção 3.1). Isso permite estimar o tempo total de execução das aplicações paralelas sob várias condições de forma estática.

A ferramenta J-CRES também já está implementada e em fase de testes de integração com a ferramenta J-ARC.

5. Conclusões

Este artigo apresentou duas ferramentas para geração automática de modelos analíticos de performance, fundamentais para a predição de tempo de aplicações paralelas. A primeira, J-CRES, representa os recursos computacionais de um sistema paralelo por meio de medições de tempo obtidas pela execução de *benchmarks* e gera um modelo de máquina. A segunda, J-ARC, avalia a estrutura paralela e o código-fonte da aplicação e gera um modelo da aplicação. Combinando as saídas destas ferramentas, obtém-se uma estimativa de tempo de execução sob variadas configurações da aplicação e da plataforma paralela.

As ferramentas propostas têm como principais vantagens em relação àquelas existentes na literatura a não exigência de marcação específica no código-fonte, o uso de uma linguagem de programação padrão (Java), a possibilidade de modelar atrasos causados por contenção de recursos e o uso de intervalos para representar os tempos de execução previstos.

Agradecimentos

Os autores agradecem à CAPES pelo apoio parcial no desenvolvimento deste trabalho.

Referências

[1] M. E. Crovella. *Performance Prediction and Tuning of Parallel Programs*. Tese de Doutorado, University of Rochester, Rochester, USA, 1994.

[2] A. J. C. van Gemund. *Performance Modeling of Parallel Systems*. Tese de Doutorado, Delft University of Technology, Netherlands, Sweden, 1996.

[3] M. J. Clement. *Analytical Performance Prediction of Data-Parallel Programs*. Tese de Doutorado, Oregon State University, Oregon, USA, 1994.

[4] A.J.C. van Gemund. Symbolic Performance Modeling of Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 2, pp. 154-165, Feb. 2003.

[5] S. A. Jarvis, D. P. Spooner, L. C. Keung, e G. R. Nudd. Performance Prediction and its use in Parallel and Distributed Computing Systems. *17th IEEE International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, 2003.

[6] A. J. C. van Gemund. Automatic Cost Estimation of Data Parallel Programs. Technical Report 1-68340-44(2001)09, Delft University of Technology, Netherlands, Sweden, 2001.

[7] C. van Reeuwijk. Spar Language Specification. Relatorio Técnico PDS-2001-003, Delft University of Technology, Netherlands, Sweden, 2001.

[8] J. Brém, M. Madhukar, E. Smirni e L. Dowdy. PerPreT - A Performance Prediction Tool for Massively Parallel Systems. *Proceedings of the Joint Conference Performance Tools / MMB*, Heidelberg, Germany, 1995.

[9] S. Prakash. *Performance Prediction of Parallel Programs*. Tese de Doutorado, University of California, California, USA, 1997.

[10] M. A. Amaral Henriques. A Proposal for Java Based Massively Parallel Processing on the Web. *Proceedings of The First Annual Workshop on Java for High-Performance Computing*, Rhodes, Greece, 1999.

[11] E. J. H. Yero. *Estudo sobre Processamento Maciçamente Paralelo na Internet*. Tese de Doutorado, Universidade Estadual de Campinas, São Paulo, Brasil, 2003.

[12] F. O. Lucchese. *Um Mecanismo para Distribuição de Carga em Ambientes Virtuais de Computação Maciçamente Paralela*. Dissertação de Mestrado, Universidade Estadual de Campinas, São Paulo, Brasil, 2002.

[13] J. M. Schopf e F. Berman. Performance Prediction Using Intervals. Relatorio Técnico CS-97-541, University of California, San Diego, USA, 1997.

[14] A. W. Appel. Modern Compiler Implementation in Java. Published by Cambridge University Press, ISBN 0-521-58388-8, New York, Cambridge, 1998.

[15] J. Gosling, B. Joy, e G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[16] L. A. Smith, J. M. Bull, e J. Obdrzálek. A parallel java grande benchmark suite. *Proceedings of the ACM/IEEE conference on Supercomputing*, Colorado, USA, 2001.

[17] The Benchmark Suite. http://www.epcc.ed.ac.uk/javagrande/. Último acesso: 08/2004.