Implementação de uma Linguagem de Especificação de Aplicações Paralelas Baseada em XML para o Sistema JoiN

Cristina Enomoto e Marco Aurélio Amaral Henriques
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
Caixa Postal 6101 – Campinas, SP – 13083-970
[enomotolmarco]@dca.fee.unicamp.br

Resumo

Vários sistemas de grid e de computação distribuída existentes permitem a execução de aplicações com um fluxo básico, em que é feita a distribuição das tarefas para que sejam executadas em paralelo e depois a coleta dos resultados. Outros sistemas permitem definir uma relação de dependência de execução entre as tarefas, formando um grafo direcionado acíclico. Porém, mesmo com esse modelo, não é possível executar vários tipos de aplicações paralelas, como, por exemplo, algoritmos genéticos e de cálculo numérico que utilizam algum processamento iterativo. E é neste caso que se enquadra o sistema JoiN de processamento paralelo. Neste artigo é apresentada a implementação no sistema JoiN de uma nova linguagem de especificação de fluxo de execução de aplicações paralelas que permite um controle de fluxo mais flexível, viabilizando desvios condicionais e laços com iterações controladas. Os resultados mostram que, com a implementação desta nova linguagem, foi possível criar, com baixo custo, aplicações que antes eram consideradas impossíveis ou inadequadas para execução neste sistema.

1. Introdução

Os grids computacionais permitem a resolução de problemas complexos e simulações científicas, cuja execução não seria possível com a utilização de um único processador [1]. Um grid consiste de toda a infra-estrutura de hardware e software para prover um sistema capaz de coordenar recursos computacionais heterogêneos e distribuídos, oferecendo um acesso transparente a esses recursos.

Outro tipo de computação paralela é a computação distribuída pública. Ela utiliza computadores domésticos conectados à Internet para a realização de computação científica [2]. Uma característica de aplicações apropriadas para computação distribuída pública é que a comunicação entre tarefas é mínima ou inexistente [3].

A maioria destes sistemas possui algum modelo de especificação do fluxo de execução das aplicações. Vários destes modelos permitem especificar apenas um fluxo básico, no qual uma aplicação consiste em distribuir as tarefas para execução em paralelo e depois coletar os resultados. Outros modelos fazem o controle do fluxo de execução baseados no relacionamento de dependência entre tarefas, que pode ser representado por um grafo direcionado acíclico ou DAG (*Directed Acyclic Graph*). Porém, mesmo com esses modelos, não é possível executar vários tipos de aplicações paralelas como, por exemplo, algoritmos genéticos ou de cálculo numérico que utilizam algum tipo de processamento iterativo.

Em um outro trabalho, os autores analisaram diversos sistemas paralelos e seus respectivos modelos de especificação de aplicações [4]. Além disso, propuseram um novo modelo e uma nova linguagem de especificação voltados para sistemas de computação distribuída pública.

Com o objetivo de comprovar a viabilidade prática e as características de desempenho do novo modelo e de sua linguagem de especificação, neste trabalho é apresentada a implementação desta nova linguagem sobre a plataforma JoiN. A Seção 2 faz uma breve revisão sobre JoiN, a Seção 3 revê os pontos principais do novo modelo proposto, a Seção 4 descreve como o mesmo foi implementado na plataforma e a Seção 5 discute os resutados obtidos.

2. A plataforma JoiN

JoiN é um sistema de computação virtual maciçamente paralela baseado em computação distribuída pública que tira proveito das facilidades oferecidas pela linguagem Java [5]. A plataforma JoiN foi totalmente desenvolvida em Java, o que lhe garante portabilidade e a vantagem de possuir um ambiente de execução auto-contido e configurável. O sistema é composto por computadores heterogêneos e com baixo acoplamento, que formam uma estrutura com alto poder computacional. Entre seus objetivos estão a simplificação do modo com que os computadores se integram ao sistema e o estímulo à participação de voluntários.

JoiN possui quatro componentes com responsabilidades distintas:

- Servidor: responsável pelo gerenciamento da plataforma;
- Coordenador: responsável por gerenciar a execução das aplicações em um grupo de trabalhadores;
- Trabalhador: responsável pela realização do trabalho computacional útil, ou seja, pela execução das tarefas de aplicações. Um conjunto de trabalhadores forma, com um coordenador, um grupo da plataforma;
- *Jack* (*JoiN Administration and Configuration Kit*): é o módulo de configuração do sistema e responsável também por controlar a instalação, desinstalação, início ou interrupção da execução de aplicações paralelas.

2.1. O modelo de aplicação de JoiN

O modelo de aplicações de JoiN é baseado em blocos de dados, tarefas, lotes de tarefas e relações de dependência de dados [6]. Basicamente, uma aplicação paralela é formada por: (a) um conjunto de lotes de tarefas, no qual um lote está associado a uma tarefa e à sua multiplicidade (dentro de um lote são executadas as mesmas operações sobre conjuntos distintos de dados); (b) um conjunto de dependências de dados entre lotes de tarefas, com restrições para que não haja dependências cíclicas.

Uma aplicação JoiN possui, além do código a executar, uma especificação da estrutura da aplicação, mostrando como é o relacionamento de precedência entre os lotes de tarefas e a multiplicidade das tarefas em cada lote. Essa especificação é definida por meio de uma linguagem de especificação de aplicações paralelas (PASL – *Parallel Application Specification Language*).

Na prática, a especificação PASL é um arquivo texto formado por três seções distintas:

- *header*: contém o nome e a descrição da aplicação;
- *assignment*: definição dos caminhos (*paths*) até os códigos e identificação dos tipos de tarefas; os identificadores de tarefas são formados pela justaposição da letra *T* com um índice numérico;
- *data link*: definição dos lotes de tarefas que formam a aplicação e as relações de precedência entre lotes.

Para que a ligação de dados entre lotes de cardinalidades distintas seja possível, é necessário que as suas respectivas tarefas possuam interfaces de entrada e saída de dados que suportem essa ligação. Na Fig. 1 é mostrado um exemplo de um arquivo PASL.

```
// Seção Header
name = "Aplicação Exemplo";
description = "Exemplo de uma espec. PASL";

%% // separador de seção

// Seção Assignment
path = "/app/test1"; //diretório com o
                     //código da aplicação
T1 = "distribute.class"; //arquivo com a
     //computação realizada na tarefa
T2 = "process.class";
T3 = "gather.class";

%% // separador de seção

// Seção Data Link
// Relacionamento entre os lotes de tarefas
B1 = T1(1) << "infile"; //lote B1 possui 1
   // instância da tarefa T1 e recebe como
   // entrada o arquivo "infile"

B2 = T2(500) << B1; //lote B2 possui 500
    //instâncias da tarefa T2 e recebe os
    //dados da saída de B1

B3 = T3(1) << B2 >> ''outfile'';
 //lote B3 possui 1 instância da tarefa T3
 //recebe os dados de entrada da saída de
 // B2 e envia o resultado para "outfile"
```

Figura 1. Exemplo de uma especificação de aplicação do JoiN (arquivo PASL)

Quando uma aplicação é executada:

- o coordenador recebe do servidor a aplicação e distribui uma parte das tarefas e seus respectivos dados entre os trabalhadores sob sua coordenação; esta distribuição estática é feita com base em um fator de desempenho reportado pelos trabalhadores quando ingressam no sistema e executam alguns códigos de *benchmark* padrão;
- à medida que tarefas são concluídas e surgem trabalhadores ociosos, as demais tarefas são distribuídas dinamicamente, equilibrando a carga

computacional de acordo com o desempenho de cada trabalhador;
- após a distribuição de todas as tarefas de um lote e à medida que surgem trabalhadores ociosos, o coordenador inicia uma nova distribuição gradual das mesmas tarefas (replicação de tarefas) para estes trabalhadores, o que torna a aplicação tolerante a falhas nos trabalhadores e melhora o equilíbrio de carga;
- quando uma das réplicas de uma tarefa retorna seu resultado, réplicas idênticas que eventualmente tenham sido distribuídas são finalizadas ou têm seus resultados desprezados;
- devido ao relacionamento de precedência entre os lotes, antes de iniciar a execução das tarefas do próximo lote é necessário aguardar até que a última tarefa do lote corrente tenha terminado.

2.2. Limitações do modelo de aplicações de JoiN

Muitos algoritmos evolutivos utilizam um processamento iterativo para a resolução de um problema. Um exemplo é a ferramenta computacional chamada *Phylogenetic Tool Project* (PTP) [7] que permite encontrar soluções quase ótimas em espaços de estados com elevado número de candidatos ao longo do processo de busca. A ferramenta precisa administrar populações de candidatos que devem ser tratadas individualmente ao longo das gerações. Estas populações podem ser distribuídas em diferentes tarefas e processadas em paralelo, o que faz de PTP uma aplicação candidata a tirar proveito de JoiN. Porém, a característica iterativa de PTP apresenta uma necessidade para a qual o modelo PASL não está preparado, isto é, o processamento paralelo deve ser repetido um número de vezes que é dependente da convergência do algoritmo para um resultado esperado. Se este número de iterações fosse fixo, a ferramenta PTP poderia ser especificada em PASL como uma seqüência (provavelmente longa) de lotes de tarefas, cada um dedicado a uma iteração ou geração das populações. Para permitir a implementação de PTP em JoiN foram necessárias algumas adaptações neste último de forma que uma parte da aplicação (externa ao sistema) pudesse interagir com ele e determinar quando as repetições do processamento paralelo deveriam continuar ou terminar. Apesar destas adaptações terem funcionado e demonstrado como o PTP se beneficiava do paralelismo oferecido pela plataforma JoiN, estava claro que esta era uma solução específica para aquele problema e de difícil aplicação em outras situações. O modelo de especificação de aplicações paralelas precisava ser estendido para dar suporte a este tipo de aplicação iterativa sem um número pré-determinado de iterações. Além disso, era preciso ainda melhorar o suporte precário que PASL dava a aplicações iterativas com um número fixo de repetições e prover suporte para outros tipos de aplicação.

Foi para atender estas necessidades que os autores propuseram em outro trabalho um novo modelo de especificação de aplicações paralelas mais genérico e flexível [4]. O presente trabalho complementa aquele, já que detalha e avalia a implementação prática do modelo em uma plataforma real (JoiN).

3. O modelo de especificação de aplicações

O novo modelo proposto pelos autores na ref. [4] provê suporte a novas estruturas de controle de fluxo: seqüencial, processamento iterativo, processamento de repetição condicional e desvio condicional. Estas estruturas estão baseadas no conceito de *blocos de execução* descrito a seguir.

3.1. O conceito de bloco de execução

Para permitir a execução de mais de uma estrutura de controle distinta em uma mesma aplicação, foi introduzido o conceito de *bloco de execução*. Ele é um conjunto lotes de tarefas que são executadas a partir de uma estrutura de controle (seqüencial, iterativo, repetição condicional e *switch/case*). Uma aplicação passa a ser vista como um conjunto de *blocos de execução* que são executados seqüencialmente. O resultado do último lote de tarefas de um *bloco de execução* é passado como dado de entrada para o primeiro lote do próximo *bloco de execução*. A Fig. 2 ilustra o fluxo de execução de uma aplicação com três *blocos de execução*, neste exemplo como os blocos são definidos na ordem: EB_1, EB_2 e EB_3, o fluxo de execução dos blocos será:

$$\mathbf{EB_1 \rightarrow EB_2 \rightarrow EB_3}$$

3.2. Processamento seqüencial

Este é o *bloco de execução* mais básico, composto por um grupo de lotes de tarefas que têm um relacionamento de dependência seqüencial entre si. A computação realizada neste *bloco de execução* é representada pelo modelo de fluxo DAG. O modelo de aplicações PASL de JoiN segue esta estrutura.

3.3. Processamento iterativo

Este *bloco de execução* permite que um grupo de lotes de tarefas seja executado em um laço com um

número de iterações fixo. Enquanto as iterações não terminarem, os resultados do último lote neste bloco são passados como dados de entrada para o primeiro lote do próprio bloco. Após o término da última iteração, os resultados são passados para o primeiro lote do bloco de execução seguinte. Deve ser utilizado quando o número de repetições do laço é conhecido com antecedência.

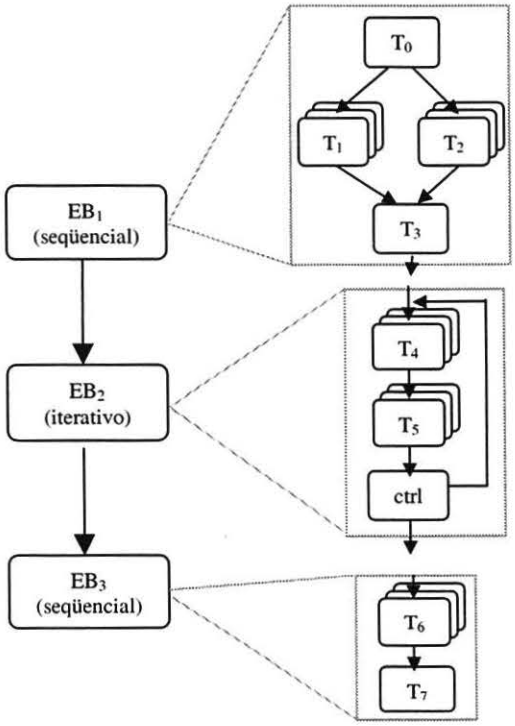

Figura 2. Exemplo de uma aplicação: seqüência de blocos de execução

3.4. Processamento de repetição condicional

Esta estrutura é utilizada no processamento iterativo em que a decisão de saída do laço não é a quantidade máxima de repetições, mas sim um outro critério, que pode ser a verificação da precisão de um resultado, por exemplo. É necessário ainda o estabelecimento de um valor limite para o número de iterações para evitar que o programa fique em um laço infinito. Portanto, o bloco de repetição condicional possui dois critérios de saída: (1) o limite de iterações foi atingido ou (2) uma condição definida foi satisfeita. Para verificar a condição de saída, o bloco de repetição condicional necessita de um lote extra de tarefas, chamado lote de verificação ou controle. Este lote verifica os resultados do último lote executado e retorna um resultado do tipo *boolean*, indicando se são necessárias mais iterações ou não. Este lote de controle precisa ser o último lote no *bloco de execução* para facilitar o gerenciamento nas duas condições de saída do laço.

O lote de controle pode ser considerado um lote virtual. Ele apenas recebe o resultado do estágio anterior e verifica se o critério de saída é satisfeito ou não. Os dados recebidos como entrada são passados sem nenhuma modificação para o próximo estágio, que pode ser o primeiro lote do mesmo *bloco de execução* ou o primeiro lote do próximo *bloco de execução*.

3.5. Desvio condicional – *Switch/Case*

O *bloco de execução Switch/Case* permite que um ou outro lote de tarefas seja executado dependendo dos dados de entrada do bloco. Da mesma forma que o *bloco de execução* de repetição condicional, ele também precisa ter um lote específico de verificação, porém com uma finalidade distinta. O lote de controle do *switch/case* retorna um identificador indicando o lote de tarefas que deve ser executado no próximo estágio, isto é, determina que tarefas devem ser instanciadas. Esta estrutura é utilizada quando são necessários processamentos diferenciados, dependendo do valor dos parâmetros (dados de entrada) do bloco.

3.6. Especificação em XML

Para implementar o novo modelo de aplicação, foi proposta uma nova linguagem de especificação de aplicações paralelas chamada XPWSL (*XML based Parallel Workflow Specification Language*) [4]. A nova linguagem foi descrita em XML (*eXtensible Markup Language*) já que esta é uma linguagem na qual é possível definir novos rótulos. Isto torna XPWSL mais flexível, pois permite que novos atributos e elementos sejam facilmente acrescentados. Outra vantagem de XML é que praticamente todas as linguagens de programação já oferecem algum suporte para o *parser* de XML, o que pode simplificar a interpretação e a geração automática de arquivos XPWSL a partir de uma ferramenta gráfica de especificação, por exemplo.

4. Implementação do novo modelo de aplicações na plataforma JoiN

Cada tarefa em JoiN é uma classe que implementa a interface AppCode:

```
public interface AppCode {
   public Serializable
                    taskrun(Serializable par);
}
```

Esta interface define o método *taskrun*, que implementa a computação realizada em um trabalhador. Basicamente, quando o trabalhador recebe uma tarefa do coordenador, ele apenas executa o método *taskrun* com os parâmetros (`par`) recebidos do coordenador e em seguida retorna o resultado para o coordenador.

O coordenador é responsável pelo controle da execução da aplicação e para isso ele utiliza o serviço *C_ApplicationManager* (C_ indica coordenador) que possui três componentes principais: *TaskScheduler*, *ExecutionEngine e o JAS*. O relacionamento entre as principais classes do *C_ApplicationManager* é ilustrado na Fig. 3.

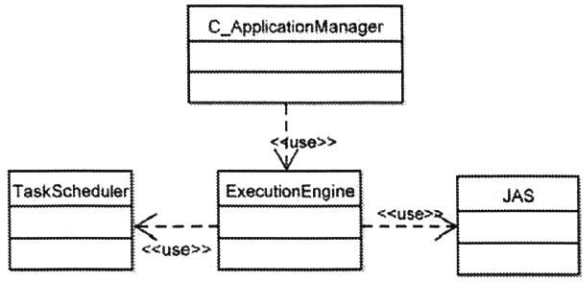

Figura 3. Relacionamento das principais classes do C_ApplicationManager de JoiN

O JAS (JoiN Application Specification) é responsável pela interpretação do arquivo PASL da aplicação.

O *TaskScheduler* realiza uma avaliação de desempenho do hardware (benchmark) que é utilizada para o escalonamento das tarefas nos trabalhadores.

O *ExecutionEngine* controla o fluxo da aplicação utilizando uma lista com todas as suas tarefas e organizando os dados de entrada e os resultados dos trabalhadores. O ExecutionEngine não realiza um controle de estado sofisticado da aplicação; ele apenas agrupa as tarefas de acordo com o relacionamento de precedência em *prontas* e *não prontas*. São consideradas *prontas* as tarefas que não dependem de nenhuma entrada de dados ou os dados de que elas dependem já tenham sido processados e estão disponíveis. A execução dos lotes de tarefas é feita a partir da lista de tarefas *prontas* e à medida que as tarefas vão sendo executadas, elas são removidas da lista de *não prontas* e são passadas para a lista de *prontas*.

No PASL não é permitido nenhum ciclo no grafo de dependências, garantindo que o fluxo não fica em um laço infinito. Não é possível implementar uma estrutura de repetição nesse modelo porque não existe a persistência de informações de estado.

Para suportar as novas estruturas de controle de XPWSL foi acrescentada uma nova definição: o *bloco de execução*. Como foi discutido na Seção 3, uma aplicação JoiN pode ser vista como um conjunto de *blocos de execução*; cada bloco de execução contém um objeto *ExecutionEngine* com as tarefas que serão processadas neste bloco; dependendo de qual seja o seu tipo, ele também pode possuir um método de controle.

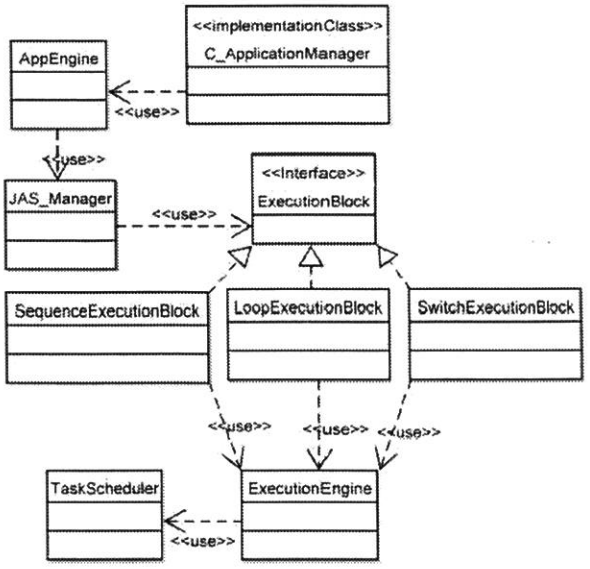

Figura 4. Relacionamento de classes no novo modelo de especificação de aplicações

Foi criada também uma nova classe, o *AppEngine*, para fazer o controle do fluxo de execução entre os blocos de execução. A função do AppEngine é ser apenas um envoltório (*wrapper*). O escalonamento dos lotes de tarefas no *C_ApplicationManager* continua sendo baseado no *ExecutionEngine* com a diferença de que em lugar de processar uma aplicação, apenas um *bloco de execução* será processado; isso é transparente para o *C_ApplicationManager*. O *AppEngine* apenas controla a ordem de execução dos blocos de execução. O relacionamento entre as classes do novo modelo de especificação e o *C_ApplicationManager* é ilustrado na Fig. 4.

O JAS_Manager substitui o JAS, fazendo a interpretação de XPWSL e realizando o controle dos blocos de execução. Todos os tipos de *bloco de execução* devem derivar da interface básica ExecutionBlock:

```
public interface ExecutionBlock {
 public void initialize(Object[] _result);
 public ExecutionEngine getExecutionEngine (
          JAS_Manager jasMan, long appNumber);
 public boolean
              moreIterations(Object[] _result);
 public void finalize();
}
```

Esta interface comum especifica os seguintes métodos:

- ***initialize*** é o primeiro método chamado quando o controle passa para o bloco de execução. Ele recebe como parâmetro o resultado do último *ExectutionBlock*;
- ***getExecutionEngine*** retorna o *ExecutionEngine* apropriado, contendo os lotes de tarefas a serem processados e as relações de dependência. Este método é chamado no momento em que o *ExecutionEngine* é executado;
- ***moreIterations*** é um método booleano utilizado pelo controle iterativo para determinar se o controle deve ser passado para o próximo bloco de execução ou o mesmo bloco deve ser re-executado. Nos *blocos de execução* que não são iterativos esse método retorna sempre *falso*. Este método é chamado após a execução do último lote de tarefas do bloco de execução;
- ***finalize*** é o método utilizado para realizar um processamento de finalização do bloco, sendo chamado imediatamente antes do controle ser passado para o próximo bloco de execução.

Em JoiN, o controle da execução de uma aplicação é feito pelo módulo coordenador. Este módulo possui componentes responsáveis pela alocação das tarefas às máquinas trabalhadoras e também para o controle do fluxo de execução das tarefas da aplicação, organizando os dados de entrada e os resultados dos trabalhadores.

Para suportar as novas estruturas de controle, foi acrescentado ao coordenador o suporte necessário para os blocos de execução. Como a implementação da plataforma é feita em Java, foram definidas classes para cada tipo de bloco de execução (seqüencial, desvio condicional e repetição com número fixo de iterações ou condicional). Cada uma das estruturas de controle deve implementar a interface *ExecutionBlock* e as classes correspondentes são detalhadas a seguir.

`public class SequenceExecutionBlock`

Esta classe representa a estrutura básica na qual não é aplicado nenhum método de controle nos dados de entrada. Esta classe representa o modelo de dependências de tarefas já existente em PASL.

`public class SwitchExecutionBlock`

Nesta estrutura é necessário definir em *XPWSL* qual a classe de desvio condicional que está associada ao bloco. Essa classe de controle deve implementar a interface *AppSwitchControl* que é carregada quando o controle da execução é passado para este bloco. O resultado desta tarefa de controle é comparado com os valores das cláusulas definidas no *switch* para determinar quais lotes serão executados.

`public class LoopExecutionBlock`

Este *bloco de execução* pode ser executado com ou sem a definição de uma classe de controle. A classe de controle do laço deve implementar a interface *AppLoopControl*. Se for especificada uma classe de controle do laço em XPWSL, o resultado do último lote de tarefas é verificado por esta classe para definir se o critério de saída foi satisfeito ou não. O controle de execução passará para o próximo *bloco de execução* se o número máximo de iterações for realizado ou se o critério de saída foi satisfeito. Se a classe de controle não for especificada, apenas o número máximo de iterações é verificado.

Em JoiN, os lotes de tarefa com cardinalidade 1, isto é, que são executados em uma única instância, geralmente são utilizados para realizar a distribuição de dados (preparar um vetor com os dados a serem enviados para cada trabalhador) ou para coletar os resultados dos trabalhadores. Por esse motivo, são executados no próprio coordenador.

As classes de controle (AppSwitchControl e AppLoopControl) também possuem cardinalidade 1. Como estas classes devem implementar uma verificação de um conjunto de dados que as tarefas de um lote retornaram ao coordenador e como esta verificação é feita por meio de um lote de cardinalidade 1, o novo modelo com os Blocos de Execução também se beneficia do tratamento de lotes unitários no próprio coordenador da plataforma JoiN.

Para um desenvolvedor de aplicações paralelas em JoiN não há grandes diferenças na preparação dos programas ao adotar a nova linguagem de especificação de fluxo XPWSL. As diferenças principais do novo modelo em relação ao sistema original baseado em PASL está na linguagem de especificação de aplicações e nas classes de controle para suportar as novas estruturas de controle. A implementação das classes com o código das tarefas continua sendo feita da mesma forma que no modelo PASL .

Para utilizar a estrutura condicional e a de repetição condicional, é necessário implementar o tratamento

para realizar o controle delas, visto que cada uma possui uma interface específica.

O bloco de repetição condicional utiliza uma classe de controle *boolean* que recebe o resultado do último lote de tarefas do bloco de execução e retorna um *boolean* indicando se o critério de saída do laço foi satisfeito ou não.

O bloco *switch/case* utiliza uma classe de controle que retorna uma *string*. Esta classe é carregada assim que o controle passa para esse bloco de execução; os dados de entrada do bloco de execução são analisados pela tarefa de controle e seu resultado será comparado com os valores de cada cláusula definida no XPWSL. A utilização de uma classe de controle para definição da escolha da cláusula permite que o desenvolvedor defina qualquer critério e um número indeterminado de tratamentos diferenciados.

5. Resultados

5.1. Testes funcionais

Cada uma das novas estruturas de controle propostas em XPWSL foi implementada, testada e validada separadamente em JoiN. Em seguida, algumas aplicações paralelas que utilizam estas estruturas foram especificadas e executadas com XPWSL. Vários algoritmos genéticos, que não poderiam ser implementadas na plataforma JoiN sem as novas estruturas de XPWSL, se tornaram viáveis, abrindo o leque de aplicações que podem tirar proveito desta plataforma.

Um exemplo é a ferramenta PTP descrita na Seção 2.2. No passado, sua implementação no sistema JoiN só foi possível após uma customização deste sistema para a ferramenta, o que não consistiu em uma solução adequada para outros projetos.

A utilização da linguagem XPWSL tornou possível a implementação integral desse tipo de aplicação na plataforma JoiN, já que XPWSL oferece o suporte necessário para o controle, em tempo de execução, do número de iterações de acordo com os resultados obtidos nos cálculos (estrutura de repetição condicional).

Entre as aplicações implementadas e testadas, encontra-se também um pacote em Java para algoritmos genéticos [8]. Esta aplicação considera uma população de cromossomos e um conjunto de operadores genéticos que define a probabilidade de cada um ser escolhido entre os demais. A partir dessa população é iniciado um processo iterativo de atribuição de um grau de *fitness* e a geração de uma nova população. Este programa exercitou com sucesso as estruturas de repetição com um número fixo de iterações e a estrutura seqüencial. Sua especificação em JoiN usando PASL torna-se inviável à medida que aumenta o número de iterações desejadas, pois passa a exigir um número muito elevado de lotes de tarefas em seqüência. Usando XPWSL, a especificação da aplicação é praticamente a mesma para qualquer número desejado de iterações. O pacote foi executado em paralelo com sucesso sob diversas condições.

5.2. Testes de desempenho

Uma vez demonstrada a viabilidade prática de XPWSL em JoiN, foi feita uma análise do impacto desta linguagem no desempenho do sistema. Para isso, várias aplicações de teste foram executadas em duas versões: usando PASL e XPWSL.

Como o suporte às novas estruturas foi todo implementado no módulo coordenador e como o objetivo dos testes é analisar o impacto das novas estruturas de controle, foram utilizados apenas lotes de tarefa com cardinalidade unitária. Tanto em PASL como em XPWSL, os lotes unitários foram executados no coordenador e, como os tipos de tarefas são exatamente os mesmos, a diferença dos tempos de execução representa o impacto de XPWSL no desempenho.

A primeira comparação feita foi entre códigos que apresentavam apenas seqüências de lotes de tarefas. A comparação do tempo de execução deste tipo de aplicação básica contendo apenas relacionamento de dependências entre lotes mostrou que não há degradação nem melhoria significativos de desempenho do sistema quando XPWSL é adotada.

A segunda comparação se deu entre aplicações que necessitam de laços com número fixo de iterações. Como PASL define apenas o relacionamento de dependências entre lotes, cada uma das iterações foi definida explicitamente em PASL e, em XPWSL, foi utilizado o bloco de repetição com o número de iterações a ser realizado. Como esperado, o tempo gasto com a linguagem mais simples (PASL) foi menor que com XPWSL; porém a diferença foi desprezível (ordem de ms) em comparação com o tempo total de processamento de uma aplicação paralela (ordem de minutos ou horas). Portanto, o impacto de XPWSL nas aplicações que façam uso deste tipo de laço não deverá ser percebido pelos usuários e desenvolvedores.

Outra comparação foi voltada para o impacto no desempenho causado pelo bloco de execução com repetição condicional. Neste caso, a classe de controle, que define se o laço deve ou não ser terminado, pode precisar ser recarregada dinamicamente a cada iteração, dependendo da memória disponível. Além do tempo de (re)carga da classe, há ainda o tempo de execução da mesma, o qual também tem impacto no

desempenho. Como cada aplicação requer classes de controle de laços distintas, é difícil avaliar a priori o impacto que tal classe terá no desempenho. O programador da aplicação paralela deve estar ciente de que critérios de parada muito complexos na classe de controle poderão comprometer o desempenho da aplicação, já que terão de ser executados a cada iteração. Entretanto, os exemplos que foram implementados (reconstrução de árvores filogenéticas, por exemplo) utilizaram critérios de parada para repetições condicionais relativamente simples, tornando praticamente imperceptível a variação de desempenho causada pelas classes de controle de laços. Esta comparação foi feita para a mais longa situação possível para uma aplicação específica, isto é, fez-se com que o laço condicional fosse executado o número máximo de vezes especificado e comparou-se com o tempo gasto na execução (por um mesmo número de vezes) de um laço similar, porém sem a classe de controle.

O impacto no desempenho causado pelo bloco de execução de desvio condicional se assemelha ao causado pelo bloco de repetição condicional, já que ambos são mais fortemente influenciados pelo tempo de (re)carga e de execução da classe de controle. Também neste caso, o impacto final dependerá da complexidade do código incluído na classe de controle.

A implementação do novo modelo de especificação de aplicações paralelas em JoiN pode ser considerada bem sucedida já que tornou possível a execução de outros tipos de aplicações sem acarretar em uma queda de desempenho perceptível. Além disso, o uso de XML na implementação oferece um novo grau de liberdade e flexibilidade não existentes no modelo anterior de JoiN, os quais permitirão a adição de novas funcionalidades no futuro sem grandes esforços.

Apesar da implementação ter sido feita na plataforma JoiN, entendemos que XPWSL pode ser implementado em outros sistemas paralelos, desde que estes tenham em sua arquitetura um módulo gerenciador de tarefas como o coordenador em JoiN.

6. Conclusões e trabalhos futuros

Foi apresentada a implementação do modelo de especificação XPWSL na plataforma JoiN, visando permitir a definição de novas estruturas de controle de fluxo de aplicações nesta plataforma. Estas novas estruturas de controle podem ser combinadas em uma única especificação, viabilizando a implementação de aplicações mais complexas que as permitidas anteriormente.

Além disso, foi mostrado que é desprezível a redução no desempenho causada pela adoção da linguagem XPWSL, mais completa e flexível que sua antecessora em JoiN.

Como continuação desse trabalho, planejamos refinar o modelo XPWSL para permitir a definição de novos atributos para aplicações, tais como as restrições de execução (requisitos mínimos de memória e velocidade de CPU) para cada tarefa. Assim, será possível realizar um escalonamento mais eficiente e equilibrado das tarefas nos processadores, reduzindo o tempo de execução das aplicações paralelas.

7. Referências

[1] I. Foster and C. Kesselman, "Computational Grids" of the book "The Grid : Blueprint for a New Computing Infrastructure", M. Kaufmann, ISBN 1-55860-475-8, 1999.

[2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky and D. Werthimer, "SETI@home: An experiment in public resource computing", Communications of the ACM, Vol.45 No.11, pp 56-61, Nov. 2002.

[3] D. Anderson, "Public Computing: Reconnecting People to Science", Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madri, Espanha, Nov. 2003.

[4] C. Enomoto and M. A. A. Henriques, "A Flexible Specification Model based on XML for Parallel Applications", 17th International Symposium on Computer Architecture and High Performance Computing - SBAC-PAD, Rio de Janeiro, 2005.

[5] E. J. H. Yero, F. O. Lucchese, F. S. Sambatti and M. A. A. Henriques., "Join: The Implementation of a Java-based Massively Parallel Grid", Future Generation Computing Systems, Elsevier, Vol. 21, pp. 791-810, 2005.

[6] F. O. Lucchese, "Um Mecanismo para Distribuição de Carga em Ambientes Virtuais de Computação Maciçamente Paralela", dissertação de mestrado , Faculdade de Engenharia Elétrica e de Computação - Unicamp, 2002.

[7] O. Prado, F. J. Von Zuben, M. A. A. Henriques. "PTP and JoiN as Software Packages for Phylogenetic Inference", Conferência Internacional de Bioinformática e Biologia Computacional, Ribeirão Preto, Brasil, Mai. 2003.

[8] "GAJIT - Simple Java Genetic Algorithm Package" home page http://www.micropraxis.com/gajit/, último acesso em Ago. 2005.