Usando a Linguagem pFun para a Programação de GRIDs

Ibero C. K. Benítez[1], André R. Du Bois[1]

[1]Centro Politécnico – Universidade Católica de Pelotas (UCPel)

Caixa Postal 402 – CEP 91.501-970 – Pelotas – RS – Brazil

{ibero,dubois}@ucpel.tche.br

1. Introdução

Atualmente, alguns aplicativos necessitam de uma alta taxa de processamento para serem executados, no entanto, uma máquina apenas não consegue prover esse poder de processamento necessário para a execução desses aplicativos. Existem alguns métodos para a resolução desse problema como por exemplo: programação paralela em máquinas multi-core, e clusters, que constitui em um conjunto de computadores interligados através de redes de alto desempenho, com a característica de que cada máquina possui a mesma arquitetura. Uma outra proposta de solução interessante é empregar o processamento em GRID, esta técnica consiste em um conjunto de clusters com arquiteturas heterogêneas distribuídos globalmente, a idéia é dividir a computação disponível para cada uma das máquinas.

Programas escritos em linguagens funcionais podem ser facilmente particionados para uma execução em paralelo. Os programas desse tipo de linguagem têm a característica de serem formados por sub expressões que podem ser simplificadas separadamente, além de serem programas pequenos com grande poder expressivo. Dessa maneira, os programas funcionais são adequados para a computação paralela em GRIDs, uma vez que é permitida partição desses programas e a conseqüente execução em paralelo de cada uma de suas partes.

O objetivo do presente trabalho, é fazer um estudo sobre o uso da linguagem pFun [Du Bois, A. et al.] no desenvolvimento de programas paralelos para GRIDs. Para avaliar a linguagem, foi desenvolvida uma aplicação para calcular a aproximação do Pi através do método Monte Carlo. Essa aplicação foi executada em um pequeno GRID e a partir dos resultados, algumas modificações na máquina virtual da linguagem são propostas para melhorar o desempenho de programas pFun em GRIDs.

2. A Linguagem pFun

A linguagem pFun é uma linguagem funcional de paralelismo semi-explícito construída especialmente para a programação de diferentes arquiteturas paralelas incluindo, máquinas multi-core e clusters. Esta linguagem está sendo implementada no projeto de pesquisa *Mobilidade de Código usando uma Linguagem Funcional Pura*, na UCPel. pFun possui um compilador, que gera byte-codes, os quais são executados sobre

a máquina virtual pFun. Assim como toda a linguagem funcional pura, pFun é baseada no modelo matemático do cálculo lambda e, ainda, possui as seguintes características: é puramente funcional (inexistência de atribuição), avaliação estrita (os argumentos da função são avaliados antes da chamada da função) e funções da alta ordem (funções que recebem funções como argumento). A seguir são apresentadas as primitivas de paralelismo, e a máquina virtual da linguagem pFun.

2.1 Programação Paralela na Lingagem pFun

A criação de programas paralelos em pFun é feita através de duas primitivas da linguagem: `par` e `sync`. A primeira indica que uma tarefa pode ser executada em paralelo, mas quem decide se essa tarefa será avaliada em paralelo ou não é a máquina virtual pFun. A segunda devolve o resultado para o programa de uma avaliação que pode ter sido executada em paralelo.

Primitiva `par`: `par :: a -> Par a`

A primitiva `par` recebe como argumento uma expressão de qualquer tipo e retorna uma referência para um objeto `Par` (tarefa).

Primitiva `sync`: `sync :: Par a -> a`

A primitiva `sync` recebe como argumento uma referência para um `Par` e devolve o resultado da avaliação dessa tarefa.

As primitivas de paralelismo `par` e `sync` podem ser vistas como construções de baixo nível para a programação paralela, entretanto, com o uso delas é possível escrever construções mais poderosas, como por exemplo, os esqueletos de paralelismo. Estes são funções de alta ordem que encapsulam padrões recorrentes da computação. Essas funções podem ser usadas pelos programadores para escrever software paralelo mais facilmente, uma vez que o programador não precisa se preocupar com alocação e sincronização de tarefas.

Um esqueleto disponível na linguagem pFun é o `parMap`, que recebe como argumentos uma função e uma lista e avalia em paralelo a aplicação dessa função a todos os elementos da lista, gerando uma lista de resultados.

2.2 A Máquina Virtual pFun

O modelo distribuído pFun está baseado na idéia de que usuários voluntários possam doar tempo ocioso de CPU de seus computadores. Essa arquitetura distribuída é composta por dois tipos de hosts: *slaves* e *work servers.* Os *slaves,* após conectados a um *work server,* ficam pedindo por uma tarefa para executar. O *work server* gera tarefas para os *slaves*, estes recebem o trabalho, o executam e enviam o resultado de volta ao seu *work server*. Geralmente vamos ter sempre um *work server* principal que inicia a avaliação do programa pFun. Toda a vez que ocorre uma chamada a primitiva `par`, uma nova tarefa é adicionada a fila de tarefas do *work server,* que depois a distribui para um *slave* executar. Um usuário que deseja contribuir com o projeto, pode instalar uma

máquina *slave* em casa e este deve ser configurado para acessar um *work server* conhecido. Um cluster pode ser usado no sistema pFun, instalando uma das máquinas como *work server* e as outras como *slaves*.

3. Avaliação da linguagem pFun em um GRID

A aplicação consiste em uma implementação paralela do método Monte Carlo usando a linguagem pFun. O método Monte Carlo é um método estatístico, o qual é utilizado em aplicações que usam em sua resolução simulações que estão relacionadas com o acaso ou possíveis de serem enunciadas através de probabilidades estatísticas. O paralelismo é introduzido na aplicação usando o esqueleto `parMap` da linguagem pFun. O programa foi executado num pequeno GRID (Figura_1).

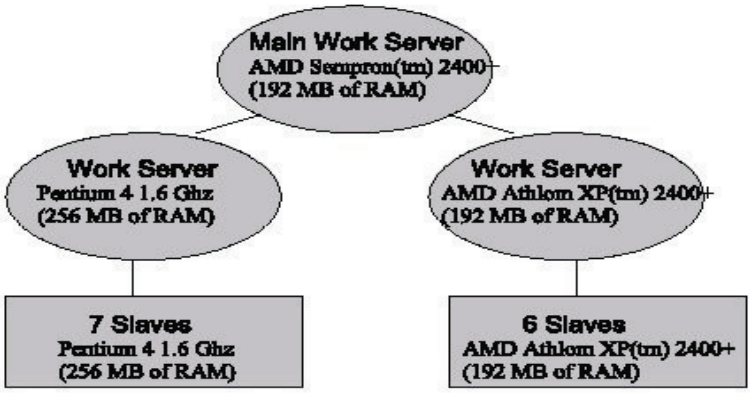

Figura_1: GRID usado nos testes

A aplicação cria tarefas em dois níveis usando `parMap`, inicialmente cria 30 tarefas, para serem executadas em paralelo, que calculam o valor do Pi. Cada uma delas divide seu trabalho e cria 15 novas tarefas também utilizando `parMap`. O tempo de execução da aplicação em uma máquina do cluster da esquerda (um Pentium 4 1.6 Ghz) foi de 4 horas, 58 minutos e 13 segundos. Em uma máquina do cluster da direita (um Athlom XP 2400+), o tempo de execução foi de 2 horas, 49 minutos e 20 segundos. Usando o GRID da Figura 1, o tempo de execução foi de 27 minutos e 18 segundos que possui um speed up de **11** comparado com as máquinas mais lentas e de 6.2 comparado às mais rápidas.

Dessa forma, os resultados mostram que a máquina virtual consegue distribuir o trabalho de forma igual para todas as máquinas, entretanto, no final da execução do programa, computadores mais rápidos ficam sem trabalho enquanto esperam por computadores mais lentos terminar o processamento.

4. Conclusão

A linguagem funcional paralela pFun, assim como as outras linguagens funcionais, estimula os programadores a escrever programas pensando somente em funções e expressões. A programação funcional traz consigo várias vantagens, seus programas são pequenos e com alto poder de expressão.

O desenvolvimento, a implementação e a avaliação da versão do método Monte Carlo em paralelo foi uma boa proposta para explorar o potencial de paralelismo da linguagem pFun devido a necessidade da alta taxa de processamento exigida por esta aplicação.

Nos experimentos realizados notou-se que em alguns momentos existem máquinas lentas trabalhando enquanto máquinas mais rápidas ficam ociosas pois não existe mais trabalho para ser realizado. Isso acaba gerando uma grande perda de desempenho. Uma proposta para solucionar esse problema seria o uso de uma técnica chamada Workqueue With Replication (WQR) [CIRNE; SANTOS NETO, 2005]. Esta técnica consiste em, após não haver mais tarefas na fila de tarefas do w*ork server*, associar réplicas das tarefas que estão sendo executadas nas máquinas lentas às máquinas ociosas. Se uma tarefa terminar, as réplicas que estavam em execução são abortadas. A solução final devolvida ao work server vem da máquina que terminar a execução primeiro.

Referências

Cousineau; Guy; Michel M. "**The Functional Approach to Programming**". Cambridge, UK: Cambridge University Press, 1998.

Cirne, W. ; Santos Neto, E. "**Grids Computacionais: Da Computação de Alto Desempenho a Serviços sob Demanda**" Mini-curso no 23° do Simpósio Brasileiro de Redes de Computadores. Disponível por www em: http://walfredo.dsc.ufcg.edu.br/papers/SBRC.2005.v18.pdf, 2005, (dezembro 2008).

Du Bois, A. et al. **"pfun: A semi-explicity parallel purely functional language",** In First Workshop on Languages and Tools for Parallel and Distributed Programming (LTPD 2007). Gramado - RS. 2007.

Du Bois, A. "**Um Estudo sobre Programação Funcional e a Linguagem Haskell".** UCPel, 1998.

Hudak, P. "**Conception, Evolution, and Application of Functional Programming Languages".** ACM Computing Surveys 21, ed no. 3 (1989): 359-411.

Vivan, G. Giulian; Ferreira, P. Vinícius; Pernas, M. Ana; Du Bois, R. André; Yamin, C. Adenauer. "**Padrões de Projeto para Mobilidade de Código no Desenvolvimento de Aplicações de Alto Desempenho no** *Middleware* **GRADEp"**. Salão de IC – ERAD 2008.