Uma Proposta de Escalonamento Distribuído para Exploração do Paralelismo na Programação em Lógica

Cristiano André da Costa[1]

Universidade Católica de Pelotas - UCPel
Escola de Informática
Rua Félix da Cunha, 412 - CEP 96010-000
Pelotas, RS, Brasil
cac@atlas.ucpel.tche.br

Cláudio Fernando Resin Geyer[2]

Universidade Federal do Rio Grande do Sul -UFRGS
Instituto de Informática
Caixa Postal 15064 - CEP 91591-970
Porto Alegre, RS, Brasil
geyer@inf.ufrgs.br

RESUMO

O presente trabalho apresenta uma proposta de escalonamento para exploração do paralelismo OU e do paralelismo E Independente na programação em lógica. O modelo de escalonamento é distribuído e recebe o nome de DSLP – *Distributed Scheduler for Logic Programming*. O DSLP pode ser empregado tanto em multiprocessadores como em multicomputadores. São utilizadas duas informações principais para auxílio ao escalonamento: da aplicação e do sistema. Através de uma análise da aplicação a ser executada pelo GRANLOG (*Granularity Analyzer for Logic Programming*) várias informações são geradas sobre um programa e utilizadas no auxílio à tomada de decisão. Alguns resultados preliminares já foram obtidos.

Palavras-chaves: Programação em Lógica, Paralelismo E, Paralelismo OU, Escalonamento Distribuído, Análise de Granulosidade.

ABSTRACT

This work presents a distributed scheduler for exploiting OR-Parallelism and Independent And-Parallelism in Logic Programming. The scheduler is distributed and called DSLP – Distributed Scheduler for Logic Programming. DSLP works on parallel and distributed computers. The proposal uses two main information to aid scheduling: system information, detected before and during the execution and application information, generated statically by GRANLOG (Granularity Analyzer for Logic Programming). Some preliminary results are available.

Keywords: Logic Programming, AND parallelism, OR parallelism, Distributed Scheduling, Granularity Analyses.

---

[1]Professor na Universidade Católica de Pelotas (UCPel/RS), Técnico em Eletrônica (ETFPel/RS, 1990), Bacharel em Ciência da Computação (UCPel/RS, 1994), Mestre em Ciência da Computação (UFRGS/RS, 1998).

[2]Professor na Universidade Federal do Rio Grande do Sul (UFRGS/RS), Engenheiro Mecânico (UFRGS/RS, 1978), Mestre em Ciência da Computação (UFRGS/RS, 1986), Doutor em Informática (Université Joseph Fourier, Grenoble, França, 1991).

1 Introdução

O paralelismo na programação em lógica pode ser explorado entre cláusulas de um mesmo procedimento (Paralelismo OU) ou entre metas de uma mesma cláusula (Paralelismo E). O Paralelismo E ainda pode ser subdivido em Dependente ou Independente de acordo com a existência ou não de conflito de ligações de variáveis. A exploração simultânea do paralelismo E e OU permite beneficiar uma gama maior de aplicações ([KER 94]).

Em um sistema paralelo, uma das tarefas mais desafiantes é a estratégia empregada na escolha das partes do programa que serão executadas em cada um dos processadores e em que ordem estas partes serão executados. A estratégia de escalonamento é responsável pela definição dessas tarefas. Várias políticas de escalonamento foram empregadas na programação em lógica, sendo em sua maioria centralizadas ([COS 96], [YAM 92]), isto é, apenas um processador fica responsável por escalonar tarefas. Entretanto, é desejável que vários processadores sejam responsáveis pelo escalonamento permitindo desta forma a utilização de mecanismos de tolerância a falhas e uma maior escalabilidade ([DAN 96]). O escalonamento distribuído pode ser utilizado desde que não sobrecarregue demasiadamente os custos de comunicação do sistema, uma vez que uma tendência atual é a execução de aplicações em multiprocessadores interconectados através de uma rede local.

Este artigo descreve uma estratégia de escalonamento distribuída para exploração do paralelismo E Independente e do paralelismo OU na programação em lógica. Esta política foi concebida para ser empregada em arquiteturas fracamente acopladas, isto é, multicomputadores. Entretanto, quando existem multiprocessadores disponíveis, o paralelismo interno também é explorado. Maiores informações sobre a proposta, bem como detalhes sobre um protótipo implementado, podem ser obtidas em [COS 98].

2 Modelo Proposto

O modelo proposto é chamado de DSLP – *Distributed Scheduler for Logic Programming* (Escalonador Distribuído para a Programação em Lógica). O DSLP realiza um escalonamento distribuído, isto é, mais de um nodo[1] do sistema possui módulo responsável pelo escalonamento de tarefas. A exploração do paralelismo E Independente e do paralelismo OU são realizadas por nodos distintos.

Os processadores envolvidos na computação são reunidos em um ou mais grupos. Cada um destes grupos possui um nodo OU, responsável pela exploração do paralelismo OU e pelo escalonamento de tarefas, e alguns nodos E, encarregados da exploração do paralelismo E Independente. Entre os nodos OU existe um fluxo de mensagens para troca de informações de carga e de exportação de trabalho entre grupos. Internamente ao grupo também existe uma troca de mensagens entre os nodos E e o escalonador, para exportação de trabalho E entre nodos do mesmo grupo.

O DSLP utiliza dois tipos de informações para auxílio ao escalonamento: informações da aplicação, obtidas através de anotações de granulosidade feitas pelo GRANLOG ([BAR 96]), e informações do sistema.

O GRANLOG (*Granularity Analyzer for Logic Programming*) é um modelo para análise automática de granulosidade na programação em lógica. O modelo realiza uma análise estática, gerando informações que podem ser utilizadas dinamicamente em tarefas como o escalonamento. A análise de granulosidade determina o tamanho dos grãos, isto é, a complexidade dos módulos

[1] No DSLP, nodo é cada um dos processadores envolvidos na computação;

que deverão ser executados seqüencialmente em um único processador. Esta complexidade é determinada através do número de resoluções máximas que cada uma das tarefas deve executar. Uma resolução em Prolog é o tempo necessário para a execução de uma instanciação.

Para que o escalonamento funcione adequadamente em várias arquiteturas diferentes de forma eficiente, são utilizados parâmetros do sistema. É possível citar como parâmetros do sistema: velocidade de comunicação, tamanho de fila de espera do processador e poder computacional. Dependendo do sistema em questão, alguns parâmetros podem ser desconsiderados (por exemplo: em um sistema homogêneo o poder computacional não necessita ser utilizado pelo DSLP). O poder computacional e os custos de comunicação, por serem constantes, são informados estaticamente pelo usuário. O tamanho da fila de espera do processador é obtido durante a execução do DSLP.

Através das informações da aplicação e do sistema o escalonador executará em paralelo aplicações explorando ambas as fontes de paralelismo na programação em lógica. Caso a proposta seja empregada em multiprocessadores conectados em rede, recomenda-se manter os grupos em processadores na mesma máquina[2]. Desta forma o paralelismo E é explorado internamente ao multiprocessador e o paralelismo OU no multicomputador.

3 Estrutura Básica

A estrutura básica do DSLP é apresentada na figura 1. Nesta figura são apresentados dois grupos cada um possuindo um nodo OU e alguns nodos E. Os nodos OU são constituídos de dois processos, um denominado trabalhador OU, responsável pela execução dos programas através do paralelismo OU, e o outro chamado escalonador. Os nodos E possuem processos denominados trabalhadores E. Esses processos, realizam a execução de programas em lógica explorando o paralelismo E Independente.

Entre os escalonadores é formado um anel lógico, como mostra a figura 2. Este anel utiliza um protocolo de comunicação bidirecional com passo máximo. O anel é bidirecional, para que cada escalonador possa se comunicar com ambos os vizinhos.

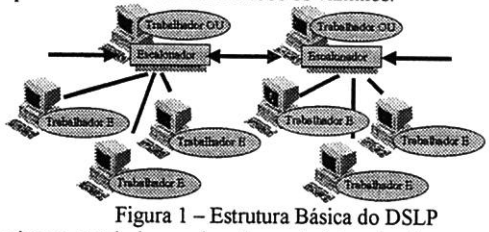

Figura 1 – Estrutura Básica do DSLP

A partir do número total de escalonadores é determinado o passo máximo, isto é, quantidade máxima de vezes que uma mensagem pode ser enviada no anel (normalmente as mensagens são enviadas em ambas as direções). Desta forma, não ocorre de uma mensagem passar mais de uma vez pelo mesmo nodo.

Sempre que uma mensagem de um escalonador para outros é gerada, o passo máximo é empacotado na mesma. Quando uma mensagem de escalonador é recebida e deve ser enviada a outro nodo OU, o passo é decrementado e re-enviado. O passo máximo é obtido dividindo-se o número de escalonadores por dois. Caso o número de escalonadores seja par o passo para a esquerda é uma unidade menor que o da direita.

[2] Por exemplo: 3 máquinas com 4 processadores, cria-se três grupos com 1 nodo OU e 3 nodos E cada um;

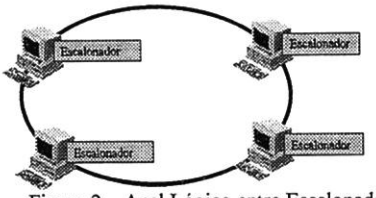

Figura 2 – Anel Lógico entre Escalonadores

Outras alternativas poderiam ser empregadas para a comunicação entre os escalonadores ao invés do anel lógico com passo máximo. Propostas de escalonadores algumas vezes utilizam primitivas de difusão (*broadcast*) e difusão seletiva (*multicast*) para distribuir informações entre os escalonadores. Neste trabalho, foi procurado minimizar o custo envolvido com a comunicação, devido a prioridade de que o sistema trabalhe em arquiteturas fracamente acopladas (multicomputadores). Como as primitivas de difusão podem onerar os custos de comunicação do sistema ([HEI 96]), as mesmas não foram utilizadas neste trabalho.

Em algumas arquiteturas, o custo de envio de um *broadcast* é o mesmo do que o de envio de uma mensagem para um único destinatário. Porém, todos os processadores que recebem a mensagem tem custos envolvidos no recebimento e desempacotamento desta. Heiss ([HEI 96]), considera que o custo de recebimento e empacotamento é alto e que algoritmos de escalonamento que utilizam primitivas de difusão distribuem mensagens que não necessitam ser recebidas por todos os processadores do sistema (por exemplo, nodos sobrecarregados não necessitariam receber mensagens de exportação).

Na proposta do DSLP os nodos OU tem sempre escalonadores associados. No caso da existência de um único escalonador só é explorado o paralelismo E.

4 Conclusões

Este artigo introduziu o modelo DSLP (Distributed Scheduler for Logic Programming). O modelo proposto já possui um protótipo implementado em PVM (*Parallel Virtual Machine*) para execução em uma rede de estações SUN. Os primeiros resultados são estimulantes e contribuem para a continuidade do trabalho. Atualmente, somente a exploração do paralelismo E está implementada.

Referências Bibliográficas

[BAR 96] BARBOSA, Jorge V. **GRANLOG: um modelo para análise automática de granulosidade na programação em lógica**. Porto Alegre: CPGCC-UFRGS, 1996. Dissertação de Mestrado.

[COS 96] COSTA, Cristiano A. da. **Um estudo das propostas que integram o paralelismo E/OU na Programação em Lógica**. Porto Alegre: CPGCC-UFRGS, 1996. 64p. (TI-493).

[COS 98] COSTA, Cristiano A. **Um Propostas de Escalonamento Distribuído para exploração do Paralelismo na Programação em Lógica**. Porto Alegre: CPGCC-UFRGS, 1998. 104 p. Dissertação de Mestrado.

[DAN 96] DANDAMUDI S. et al. **Performance of hierarchical processor scheduling in shared-memory multiprocessor systems**. Ottawa: Carleton University, 1996. (Technical Report TR– 96 –21).

[EVA 92] EVANS, D. J. et al. Dynamic load balancing using task-transfer probabilities. **Parallel Computing**, New York, v.19, p. 897-916, 1992.

[HEI 96] HEISS, Hans. Comunicação pessoal numa visita ao II/UFRGS, agosto 1996.

[KER 94] KERGOMMEAUX, J.C.; CODOGNET, Philippe. **Parallel Logic Programming Systems.** Grenoble: Universite Joseph Fourier-Grenoble I, 1994. 52p. Technical Report.

[YAM 92] YAMIN, Adenauer C. **Modelos de Implementação do Paralelismo OU na Programação em Lógica**. Porto Alegre: CPGCC-UFRGS, 1992. 114p. (TI-280).