Anahy-DVM: um módulo para distribuição de tarefas em aglomerados de computadores *

Marcelo Augusto Cardozo Jr
Gerson Geraldo H. Cavalheiro

Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950 - Bairro Cristo Rei - CEP 93.022-000 São Leopoldo - RS - Brasil
marceloac@turing.unisinos.br , gersonc@unisinos.br

Resumo

Anahy é um ambiente para processamento de alto desempenho (PAD) para aglomerados de computadores. Anahy foi concebido para oferecer uma interface aplicativa (API) para decomposição de aplicações em tarefas concorrentes e um núcleo de execução capaz de aplicar técnicas de escalonamento em nível aplicativo (FEITELSON, 1997) e suportar a execução de programas em aglomerados.

A API fornece ao usuário duas primitivas para sincronização, uma para criar fluxos de execução (*threads*) e outra para permitir que um fluxo aguarde o término de outro, com vistas a resgatar seu resultado. Desta forma, Anahy oferece um modelo de programação *multithread*, embora o núcleo de execução interprete a concorrência da aplicação em termos de tarefas, delimitadas pelas operações de criação e sincronização de fluxos, nas quais são igualmente determinadas as trocas de dados entre tarefas. A monitoração das instruções de criação e sincronização entre fluxos de execução manipulam um grafo de dependências entre as tarefas concorrentes. Este grafo é usado pelo escalonador para reger a execução da aplicação.

Esta API permite ao programador explorar uma arquitetura virtual dotada de uma memória compartilhada e múltiplos processadores virtuais (PVs). O PV é a unidade que efetivamente vai executar o código criado pelo programador. A comunicação entre estes PVs se dá através da memória compartilhada. Assim, um PV quando vai iniciar a execução de uma *thread* a ele submetida, lê os dados de entrada da memória compartilhada e no fim da computação escreve nesta memória os resultados obtidos. Os PVs são mapeados aos recursos reais de processamento da arquitetura por Anahy. Desta forma isola-se o programa de detalhes da arquitetura real, obtendo-se assim uma portabilidade de desempenho uma vez que o núcleo de escalonamento gerencia a execução do programa em termos das tarefas definidas nos contextos das *threads*.

Sobre esta arquitetura virtual, o núcleo de execução de Anahy realiza o escalonamento das atividades contidas no grafo na forma de um algoritmo de listas. Neste grafo os nós representam as tarefas e as arestas a dependência de dados. Este grafo contém um caminho crítico o qual determina o custo mínimo de execução. Anahy minimiza qualquer inserção de custo neste caminho. O algoritmo de listas adotado foi o de Graham (GRAHAM, 1969).

*Este trabalho foi desenvolvido em colaboração parcial com a HP Brasil R&D.

Em uma arquitetura distribuída, este grafo deve ser mantido nos diferentes módulos de memória, permitindo que todos os PVs, independente dos nós em que estão executando, possam comunicar via a memória compartilhada empregada na arquitetura virtual. Desta forma é necessário que existam serviços responsáveis pela manutenção do grafo. Também torna-se necessário um serviço que mantenha a memória distribuída compartilhada atualizada. Ambos serviços são implementados neste trabalho.

Quando executando em um único nó, Anahy funciona da seguinte maneira: no início da execução é criada a tarefa que contém o programa principal. Dentro desde programa eventualmente serão criadas outras tarefas, estas tarefas são inseridas no grafo. Quando há mais de um PV, estes utilizam do algoritmo de roubo de trabalho neste grafo para obter a tarefa que vão executar. Na ocasião de um roubo de trabalho, o núcleo de execução escolhe a *thread* mais próxima da raiz para enviar ao PV. A heurística utilizada supõe que tarefas mais antigas no grafo embutem uma maior probabilidade de gerar mais trabalho do que uma de nível mais baixo. Um PV quando começa a executar uma *thread*, não para sua execução salvo necessidades de sincronização. Assim, a escolha de uma *thread* de um nível mais alto também diminui a necessidade de sincronização entre PVs.

Em ambiente distribuído, o suporte deve funcionar da seguinte maneira: o nó que executa o programa principal inicia a computação. Os outros nós executam o serviço de roubo de trabalho. Este serviço escolhe um nó de maneira aleatória e envia uma mensagem pedindo trabalho ao nó escolhido. Se há trabalho que possa ser enviado, então é executado o serviço de envio de trabalho. Este serviço pega o descritor do trabalho e os dados que por este trabalho são manipulados e os empacota. Este pacote é enviado ao nó que requisitou o trabalho. Ao receber este pacote, o nó receptor desempacota este trabalho, restaura os dados na memória compartilhada e, por fim, inicia a execução do trabalho recebido. No caso de não haver trabalho para ser enviado, apenas um pacote negando a requisição é enviado. Ao receber este pacote, o nó realiza outra vez o roubo de trabalho. Na necessidade de uma sincronização com um trabalho que foi migrado, o nó envia uma mensagem ao nó que efetivamente executou este trabalho requisitando o resultado. O resultado é enviado caso disponível, caso contrário o núcleo executivo aguardará o resultado estar disponível para envia-lo ao nó requisitante.

A implementação deste trabalho utilizou-se de Mensagens Ativas (DALL'AGNOL et al., 2004) assim como emprega o conceito de *multithreading* (VALIANT, 1990).

Referências

DALL'AGNOL, E. C. et al. Construção de um mecanismo de comunicação para ambientes de processamento de alto desempenho. *Workshop em Sistemas Computacionais de Alto Desempenho*, 2004.

FEITELSON, D. G. *Job Scheduling in Multiprogrammed Parallel Systems*. [S.l.], 1997.

GRAHAM, R. L. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, v. 17, n. 2, p. 416–429, mar. 1969. ISSN 0036-1399 (print), 1095-712X (electronic).

VALIANT, L. G. A bridging model for parallel computation. *Communications of the ACM*, v. 33, n. 8, p. 103–111, Agosto 1990.