Proposta para Execução Remota de Procedimentos*

Evandro Clivatti Dall'Agnol†, Eduardo Moschetta‡,
Gerson Geraldo H. Cavalheiro

Programa de Pós-Graduação em Computação Aplicada – PIPCA
Centro de Ciências Exatas e Tecnológicas
Universidade do Vale do Rio dos Sinos
São Leopoldo – RS – Brasil
Av. Unisinos, 950. Fone/fax: 590-8161
ecd,eduardom,gersonc@exatas.unisinos.br

Resumo

Este trabalho descreve uma proposta para modelagem de um ambiente de programação para alto desempenho (PAD) utilizando arquiteturas multiprocessadas com memória distribuída (*clusters* ou aglomerados). Características como o modelo de memória, o gerenciamento da distribuição de tarefas e o modo de descrição do paralelismo da aplicação são abordadas e validadas através de resultados apresentados.

Introdução

Aglomerados de computadores (*clusters*) têm se tornado uma solução para o processamento de alto desempenho devido ao seu baixo custo. Uma das principais dificuldades encontradas pelos programadores, neste tipo de arquitetura, é a exploração eficiente dos vários níveis de memória disponíveis. Este trabalho faz um breve estudo de alguns ambientes disponíveis, Cilk [BLU 95] e Athapascan 1 [GAL 98], descreve o modelo adotado para a implementação de Anahy e apresenta resultados de uma aplicação desenvolvida segundo este modelo.

Cilk

Cilk é uma linguagem para programação paralela de propósito geral, mas que alcança seu melhor desempenho quando explora aplicações com número elevado de tarefas concorrentes.

O paralelismo de Cilk é explorado principalmente através das primitivas de programação *spawn* e *sync*. *Spawn* cria um fluxo de execução e *sync* bloqueia até que todos os fluxos filhos terminem. Baseado no uso desse mecanismo pelo programador, Cilk cria um grafo de dependências para gerenciar o escalonamento de tarefas.

Cilk foi originalmente concebido para arquiteturas SMP, tendo sido implementado um protótipo para aglomerados. A versão para aglomerados implementa o compartilhamento de dados através de um mecanismo de paginação de memória entre os nodos. O

* Apoio: CNPq, FAPERGS, UNISINOS
† BIC-FAPERGS
‡ PIBIC/CNPq

esquema de paginação é transparente para o programador, sendo inteiramente gerenciado pelo *runtime* de Cilk.

Athapascan-1

Athapascan-1 é uma interface de programação para ambientes paralelos em que o programador descreve explicitamente a concorrência da aplicação. Implementada em C++, é estruturada em três camadas:

- a API que disponibiliza a interface de programação;
- um núcleo de gestão de compartilhamento de dados;
- núcleo de escalonamento.

Athapascan-1 faz uso de um modelo de memória compartilhada, pela qual as tarefas trocam dados entre si através da especificação de permissões de acesso de escrita e leitura. Estes dados são tratados como sendo de atribuição única, ou seja, podem ser escritos somente uma vez e lidos somente uma vez. A ordem de acesso a estes mesmos dados define um fluxo de dados utilizado para controle da ordem de execução das atividades geradas pelo programa ativo no momento.

A concorrência em programas Athapascan-1 é descrita explicitamente através da primitiva *fork*, que permite a criação de tarefas assíncronas tendo como parâmetros dados de entrada e dados de saída. A distribuição do processamento entre os nós da arquitetura é realizada através de um Núcleo de Escalonamento. Este núcleo gerencia a distribuição de tarefas tendo como base informações disponibilizadas através de um Grafo de Fluxo de Dados mantido ao longo da execução. O Núcleo de Escalonamento é reativo às transformações deste grafo, ou seja, sempre que uma tarefa é criada recebendo dados, ou termina sua execução disponibilizando dados, o Grafo de Fluxo de Dados é analisado para que as decisões de escalonamento sejam tomadas. Dentre essas possíveis decisões pode estar a utilização do tempo de comunicação entre nós trocando tarefas e dados para a execução de outras tarefas que estejam prontas e esperando para ter o uso da CPU.

Anahy

Anahy é um ambiente de programação para o processamento de alto desempenho em aglomerados. A exemplo de Athapascan-1 e Cilk, propõe uma interface de programação e um núcleo de escalonamento. Assim como Athapascan-1, Anahy possui um modelo de memória compartilhada para a troca de dados, também considerando-os como de atribuição única.

O paralelismo em Anahy é explícito, sendo definido pelo programador através do uso de primitivas do tipo *create* e *join*, especificando dados de entrada e saída. O uso desses dados é que acaba explicitando as dependências entre as tarefas. O controle de execução é feito através de um Grafo de Dependências, que possui implícito em seu modelo as informações sobre os dados trocados entre tarefas, o que o diferencia do grafo utilizado em Athapascan-1. A manipulação desse grafo torna-se mais eficiente pelo fato de não haver representação explícita dos dados em questão.

O gerenciamento da distribuição da carga computacional gerada pelo programa em execução é realizado através de um Núcleo de Escalonamento. Este Núcleo considera

as informações de dependências entre tarefas informadas pelo programador através da criação e sincronização de tarefas. A execução de um programa em Anahy é suportada por um *runtime* executando de forma cooperativa em todos os nodos de um aglomerado. A configuração local a um nó é esquematizada na Figura 1.

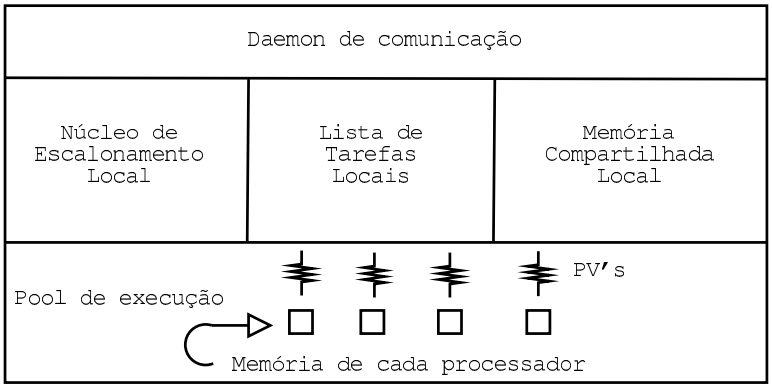

Figura 1: Anahy modelado localmente

No Pool de Execução encontram-se os processadores virtuais (PVs), cada um com sua área de memória privada utilizada para manipulação dos dados locais às tarefas ativas no momento de sua execução. Os PVs são os responsáveis pela execução das tarefas encontradas na Lista de Tarefas Locais (LTL). A Memória Compartilhada Local é de onde os PVs lêem e escrevem dados na ativação e no término da execução das tarefas, respectivamente. Esta memória possui uma camada de software capaz de controlar a movimentação de dados entre os nodos.

O Núcleo de Escalonamento Local (NEL) é o responsável pela política de escolha das tarefas, dentre as existentes na LTL, a serem executadas localmente nos PVs ou enviadas a outro nó perante uma requisição de trabalho tratada pelo Daemon de Comunicação. O NEL também é o responsável por, quando se considerar subutilizado, fazer pedidos de roubo de trabalho para o Daemon de Comunicação enviar a algum outro nó.

O Daemon de Comunicação, por sua vez, é o responsável por receber pedidos de roubo de trabalho por parte de outros nós, que se considerarem subutilizados, e enviar para estes nós tarefas e dados de acordo com as decisões do NEL. Como descrito acima, o Daemon também faz pedidos de trabalho a outros nós e, quando os recebe, armazena as tarefas na LTL e seus dados na área de Memória Compartilhada Local.

Resultados

A Tabela 1 apresenta os últimos resultados obtidos por uma aplicação desenvolvida com o objetivo de validar o modelo proposto para Anahy. Trata-se de um reconhecedor de imagens distribuído [MOS 2002] que, dentre outros métodos de reconhecimento, utiliza a comparação de histogramas de cores do sistema RGB, método com o qual estes resultados foram gerados. O reconhecedor, concebido segundo o modelo apresentado neste texto, basicamente compara um fragmento de imagem desejado com um banco de imagens inteiro, retornando regiões de imagens que mais se aproximem do fragmento utilizado. O ambiente utilizado foi um aglomerado heterogêneo de três nós bi-processados, sendo dois Pentium III 1Ghz e um Pentium III 600Mhz, todos com 512 Mb de memória RAM e interligados por uma rede Fast Ethernet.

	Seqüencial	Bi-processado		Distribuído	
Fragmento	Tempo	Tempo	*Speedup*	Tempo	*Speedup*
Pequeno	3.324 s	1.675 s	1.98	627 s	5.30
Médio	2.674 s	1.329 s	1.92	517 s	5.17
Grande	53 s	34 s	1.56	22 s	2.41

Tabela 1: Tempos de processamento

Nota-se nas colunas "*speedup*" que o ganho de desempenho é bastante próximo ao número de processadores existentes. Não se aproxima ainda mais, no caso distribuído, pelo fato de os testes seqüenciais terem sido realizados em um nodo de 1Ghz e os testes distribuídos em um *cluster* heterogêneo com alguns processadores de menor capacidade. Isto significa um bom aproveitamento dos recursos disponíveis e uma boa distribuição da carga existente. Os tempos obtidos com a utilização de fragmentos grandes são consideravelmente menores devido ao fato de o reconhedor filtrar o banco de imagens antes de efetuar a comparação, ou seja, imagens menores que o fragmento não são consideradas.

Conclusão

Atualmente, Anahy conta com um protótipo funcional oferecendo suporte à execução paralela sobre uma arquitetura SMP. A API fornecida permite a descrição da concorrência da aplicação através de primitivas de manipulação de *threads* (*athread_create* e *athread_join*). Esta API foi modelada para ser compatível com o padrão de threads POSIX, e o ambiente foi construído utilizando ferramentas de programação em software livre e implementando recursos padronizados, visando garantir a sua própria portabilidade [GAR 2001].

Com base nas aplicações já desenvolvidas, os próximos passos estão direcionados para a implementação de um protótipo distribuído como descrito neste artigo. Outros recursos clássicos de bibliotecas de *threads*, como mecanismos de sincronização (*mutexes*, por exemplo) serão incorporados à Anahy visando manter compatibilidade com códigos de aplicações já existentes.

Referências

[BLU 95] BLUMOFE, R. D. et al. Cilk: an efficient multithreaded runtime system. **ACM SIGPLAN Notices**, v.30, n.8, p.207–216, Aug. 1995.

[GAL 98] GALILéE, F. et al. Athapascan-1: on-line building data flow graph in a parallel language. In: PACT'98, 1998, Paris, France. **Anais...** [S.l.: s.n.], 1998.

[GAR 2001] GARZãO, A. S.; REAL, L. C. V.; CAVALHEIRO, G. G. H. Ferramentas para desenvolvimento de um ambiente de programação sobre agregados. In: WORKSHOP EM SOFTWARE LIVRE, 2001, Porto Alegre, Brasil. **Anais...** [S.l.: s.n.], 2001.

[MOS 2002] MOSCHETTA, E.; OSóRIO, F. S.; CAVALHEIRO, G. G. H. Reconhecimento de imagens em aplicações críticas. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 2002, Vitória, Brasil. **Anais...** [S.l.: s.n.], 2002.