Aplicando Algoritmos de Escalonamento com Múltiplas Listas no Ambiente Multithread de Anahy 3*

Cícero Augusto de S. Camargo†, Alan S. de Araújo‡, Guilherme Porto B. Cousin§, Gerson Geraldo H. Cavalheiro

[1]Programa de Pós-Graduação em Computação (PPGC)
Universidade Federal de Pelotas (UFPEL)

{cadscamargo,asdaraujo,gpbcousin,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Algoritmos de escalonamento de lista são conhecidos por sua eficiência em ambientes estáticos e largamente utilizados no núcleo de ambientes multithread dinâmicos. Anahy emprega o modelo multithread utilizando várias listas gerenciadas em paralelo, visando minimizar sobrecustos de execução. Este trabalho discute questões de implementação do ambiente de Anahy, apresentando estruturas de dados que melhoram sua eficiência.*

1. Introdução

Ferramentas de programação concorrente possuem crescente importância hoje, devido à popularização de multiprocessadores, pelo advento das arquiteturas *multicore*. Contudo, extrair alto desempenho destas arquieturas não é uma tarefa trivial. Um requisito essencial nas ferramentas é desvincular o programa concorrente do hardware paralelo por meio de uma camada de escalonamento. Algoritmos de lista [Graham 1966] são eficientes no escalonamento de programas descritos no modelo de fluxo de dados e amplamente utilizados em ambientes multithread dinâmicos. Este trabalho descreve aspectos da implementação do ambiente multithread Anahy 3 [Cavalheiro et al. 2007] e apresenta desafios na aplicação do escalonamento de lista em seu núcleo[1].

2. Algoritmos de escalonamento de lista no ambiente Anahy 3

Algoritmos de lista recebem como entrada um conjunto de processadores e um grafo de tarefas descrevendo o programa. Percorrendo este grafo, atributos de prioridade são calculados para as tarefas, as quais são ordenadas em uma lista, consumida por processadores que ficam ociosos. Aplicados a ambientes multithread dinâmicos, algoritmos de lista devem ser adaptados para lidar com threads, a unidade de escalonamento nestes ambientes, e com excuções onde threads são criados e destruídos conforme o programa evolui.

O ambiente de execução de Anahy é composto por Processadores Virtuais (PVs) e *jobs* (threads, na nomenclatura de Anahy). Cada PV possui sua própria lista de jobs prontos e sua própria pilha de jobs bloqueados esperando pelo término de outros jobs. O cálculo de atributos de prioridade para classificar jobs se vale de poucos dados. Como não

*Financiado por PRONEX/FAPERGS/CNPq GREEN-GRID Comp. de Alto Desempenho Sustentável.
†Bolsista CAPES.
‡Bolsista PIBIC/CNPq
§Bolsista PIBIC/CNPq
[1]Disponível para download em: https://github.com/cicerocamargo/Anahy-3

conhecemos o custo de processamento dos jobs *a priori*, assume-se que todos possuem custo unitário. Também não conhecemos as dependências futuras dos jobs, assim o atributo de prioridade utilizado é o *co-nível*: maior quantidade de pontos de escalonamento entre a tarefa atual do job até a tarefa inicial do programa. Cada PV executa o escalonamento buscando o job com maior co-nível em sua lista de prontos; se não há nenhum, o PV rouba trabalho nas listas dos outros PVs, buscando um job com o menor co-nível.

3. Resultados

A cada operação, a lista de um PV deve ser reordenada para manter as prioridades. Além disso, as estruturas precisam ser dinâmicas, como o ambiente. Anahy 3, implementada em C++, sofria um sobrecusto significativo por utilizar contêineres da STL[2] em função das operações *sequenciais* de gerenciamento de memória executadas nos contêineres. Assim, foram desenvolvidas estruturas para minimizar a concorrência à memória. *AnahySmartStack* é a estrutura de dados da pilha dos PVs e *AnahySmartHeap* mantém referências para jobs prontos em um PV. Ambas as estruturas alocam e liberam grandes blocos inteiros de elementos sob demanda. *AnahySmartHeap* também mantém os jobs organizados pelo seu co-nível. Ambas as estruturas são baseadas em *templates* de C++ e podem ser utilizadas como propósito geral para manipular tipos de dados genéricos.

O desempenho das estruturas foi comparado com o dos contêineres da STL. Sequências de 100000000 operações foram executadas sobre ambas as estruturas. Foi utilizado o compilador g++ 4.2.1, e os testes foram executados em uma máquina com 8 *cores*, calculando-se a média de 10 execuções para 1, 2, 4, 6 e 8 PVs. *AnahySmartStack* leva, em média, 0,64% do tempo de *std::stack* para realizar as mesmas operações, enquanto *AnahySmartHeap* leva, em média, 0,66% do tempo de *std::list*, mesmo com os custos de manutenção do *heap* binário. As estruturas de Anahy ainda melhoram sua eficiência em relação a STL conforme o número de PVs aumenta, pois a concorrência pela memória em Anahy cresce a uma taxa menor do que em STL, devido ao menor número de chamadas que as estruturas de Anahy fazem às funções `malloc` e `free`.

4. Conclusões e trabalhos futuros

Algoritmos de lista são estratégias simples, aplicáveis a ambientes multithread dinâmicos, porém requerem atenção com os sobrecustos de execução das tarefas de escalonamento. Os resultados demonstram que a otimização das estruturas de dados do núcleo trazem eficiência para as operações de escalonamento, o que é revertido em um menor tempo de execução do programa. Anahy 3 apresentou resultados preliminares satisfatórios, porém várias melhorias estão previstas: utilização de estruturas não bloqueantes na composição dos PVs, distribuição de trabalho nas operações de criação dos jobs e implementação de um núcleo híbrido, com listas compartilhadas e distribuídas pelos PVs.

Referências

Cavalheiro, G. G. H., Gaspary, L. P., Cardozo, M. A., and Cordeiro, O. C. (2007). Anahy: A programming environment for cluster computing. In *VII High Performance Computing for Computational Science*, Berlin. Springer-Verlag. (LNCS 4395).

Graham, R. L. (1966). Bounds for certain multiprocessor anomalies. *The Bell Syst. Tech. J.*, CLV(9).

[2] *Standard Template Library*: http://www.sgi.com/tech/stl/