AKSSim: uma ferramenta para a análise de algoritmos de lista em ambientes multithread dinâmicos

Cícero Augusto de S. Camargo[*], Alan S. de Araújo[†], Gerson Geraldo H. Cavalheiro

[1]*Laboratory of Ubiquitous and Parallel Systems* – LUPS
Centro de Desenvolvimento Tecnológico (UFPel)

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

1. Introdução

A crescente popularização das arquiteturas *multicore* tem fomentado ferramentas de programação que tenham a capacidade de extrair todo o potencial deste tipo de arquitetura. Ferramentas comerciais desenvolvidas sobre padrões como Pthreads e OpenMP fornecem recursos para programação de multiprocessadores. Ferramentas desenvolvidas no meio acadêmico, como Cilk [Blumofe and Leiserson 1995] e Athreads [Cavalheiro et al. 2007], já incorporam recursos que abstraem camadas inferiores, como escalonamento de tarefas.

Trabalhos anteriores [Blumofe and Leiserson 1995, Cordeiro et al. 2005] apresentam evidências da eficiência de algoritmos de lista em ambientes dinâmicos, porém não comparam a eficiência dos escalonamentos estático e dinâmico para as mesmas aplicações: em ambientes estáticos o algoritmo de escalonamento dispõe do conhecimento de toda a aplicação antes do início da execução, enquanto que em ambientes dinâmicos o algoritmo usa apenas a parte já evoluída do programa para a tomada de decisões.

Este trabalho apresenta AKSSim – *Anahy Kernel Schedulers Simulator*, uma ferramenta concebida para avaliar o impacto do uso de algoritmos de lista para o escalonamento de aplicações dinâmicas, desenvolvidas sob a programação *multithread* de Anahy [Cavalheiro et al. 2007]. Grafos representando atividades paralelas em termos de tarefas (DAGs) são gerados com o auxílio de AKSSim e, a partir destes, são obtidos grafos de threads (DCGs) que representam a mesma aplicação escrita em uma interface de programação *multithread*. Simulações são realizadas em AKSSim comparando, para uma mesma arquitetura, o desempenho de algoritmos de lista estáticos e dinâmicos para DAGs e DCGs (respectivamente).

2. Algoritmos de lista em ambientes *multithread*

Estratégias básicas de escalonamento de lista têm como unidade de escalonamento a tarefa, trecho de código sequencial delimitado por uma entrada e uma saída de dados. Algoritmos estáticos recebem como entrada um DAG descrevendo o programa a ser executado, e percorre este DAG atribuindo prioridade de execução às tarefas de acordo com uma política pré-estabelecida. As tarefas são inseridas em uma lista ordenada de forma

[*]Bolsista PIBIC CNPq
[†]Bolsista BIC/FAPERGS

que, quando da execução do programa, são consumidas conforme os processadores ficam ociosos, respeitando a ordem de prioridade atribuída.

No modelo de programação *multithread* programas são escritos em termos de fluxos de execução concorrentes, e tarefas são delimitadas por operações de criação e sincronização de tarefas. A interface de programação *multithread* de Anahy [Cavalheiro et al. 2007], por exemplo, fornece duas primitivas para manipulação de threads: *create* e *join*. Tais primitivas permitem que o programador crie novos threads, passando dados de entrada, ou force que um thread Γ_i aguarde o término de um thread Γ_j para ler da memória compartilhada eventuais dados produzidos por Γ_j. Destaca-se que Anahy permite a criar e sincronizar vários threads em apenas uma operação através da passagem de parâmetros específicos para as operações *create* e *join*. Um grafo dirigido cíclico, DCG (*Directed Cyclic Graph*), representa um programa *multithread* construído através de uma interface de programação como a de Anahy.

Em ambientes *multithread* dinâmicos novos threads são criados e inseridos no grafo conforme a execução evolui. Algoritmos de escalonamento lista para este tipo de ambiente devem prover políticas de prioridade que considerem somente a porção conhecida do DCG que descreve a aplicação. Embora internamente um DCG mantenha as dependências entre tarefas, estratégias de escalonamento para ambientes *multithread* consideram apenas o nível de threads do grafo.

3. AKSSim

Anahy Kernel Schedulers Simulator – AKSSim – é uma ferramenta de simulação inicialmente projetada para avaliar o impacto de mudanças na estratégia de escalonamento do ambiente Anahy. A ferramenta fornece recursos para geração de DAGs, obtenção o DCG correspondente e escalonamento dos grafos sobre m processadores idênticos, sob diversas heurísticas de escalonamento.

Os parâmetros que AKSSim considera para a geração de DAG são: $comprimento$ e $profundidade$, os quais definem o tamanho da estrutura do DAG, $custo$ e $desvio$, que definem os custos de processamento de cada tarefa variando para mais ou para menos dentro do $desvio$ estipulado. *Número de processadores* define o paralelismo da arquitetura a ser considerada no escalonamento da aplicação e $overhead$ indica que um taxa de *overhead* será associada a tarefas de escalonamento.

O DAG é gerado nos moldes de um programa *multithread*, sendo que cada sequência de tarefas gerada representa um thread cujo código faz chamadas aninhadas a operações *create* e *join*. O valor do parâmetro *comprimento* representa quantos pares de tarefas que terminam executando *create* e *join* existirão nesta sequência de tarefas. O usuário de AKSSim ainda pode definir que o valor de *comprimento* seja considerado o comprimento máximo para as sequências, sendo que cada sequência de tarefas possuirá um comprimento c', onde $1 \leq c' \leq comprimento$.

AKSSim possui uma interface de comunicação com o software GROOVE[1] (***GR**aphs for **O**bject-**O**riented **VE**rification*), o qual, carregado com uma gramática de grafos adequada, é capaz de obter um DCG equivalente a um DAG gerado por AKSSim. Por motivos de espaço os padrões de tranformação não serão detalhados neste trabalho.

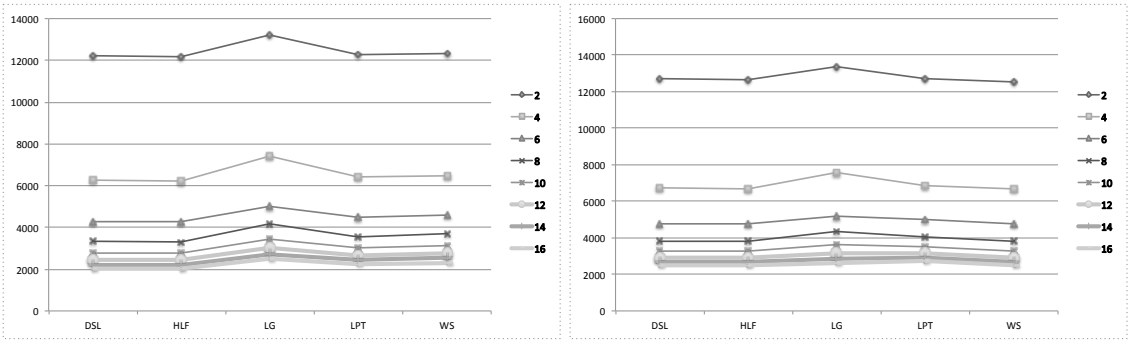
[1] http://groove.cs.utwente.nl/

Após o obtenção do DCG correspondente ao DAG gerado, são aplicadas estratégias de escalonamento de lista, estáticas para o caso de DAGs e dinâmicas para o caso de DCGs. Diversas estratégias estáticas foram implementadas em AKSSim, como os algoritmos HLF (*Highest Level First*) [Hu 1961], DLS(*Dynamic Level Scheduling*) [Sih and Lee 1993] e LPT (*Largest Processing Time*), o qual ordena a lista pelo custo das tarefas de forma decrescente. As estratégias *multithread* dinâmicas implementadas foram o algoritmo LG (Lista Global), implementado em Anahy [Cavalheiro et al. 2007], e o WS (*Work Stealing*), implementado no ambiente Cilk [Blumofe and Leiserson 1995].

4. Resultados

Duas baterias de simulações foram realizados em AKSSim, calculando, para todos os grafos gerados, os escalonamentos para 2, 4, 6, 8, 10, 12, 14 e 16 processadores (uma vez sem considerar *overhead* e outra considerando *overhead* de $1u.t.$). Os resultados serão apresentados em gráficos nas figuras 1 e 2. Nestes gráficos os elementos do eixo x denotam algoritmos de escalonamento enquanto a escala do eixo y mede unidades de tempo, sendo que as legendas dos eixos foram suprimidas para melhor visualização da informação contida no gráfico. Cada linha padronizada representa um número de processadores usado, sendo este formato escolhido por permitir uma melhor visualização do comportamento de cada algoritmo de escalonamento em função do número de processadores em uso.

Na primeira bateria de simulações foram gerados 10 DAGs com parâmetros de *comprimento máximo* = 4, *profundidade* = 5, *custo* = 50 e *desvio* = 0. Estes experimentos fixam os custos das tarefas porém variam a estrutura dos DAGs. Os resultados destes experimentos são apresentados na Figura 1.

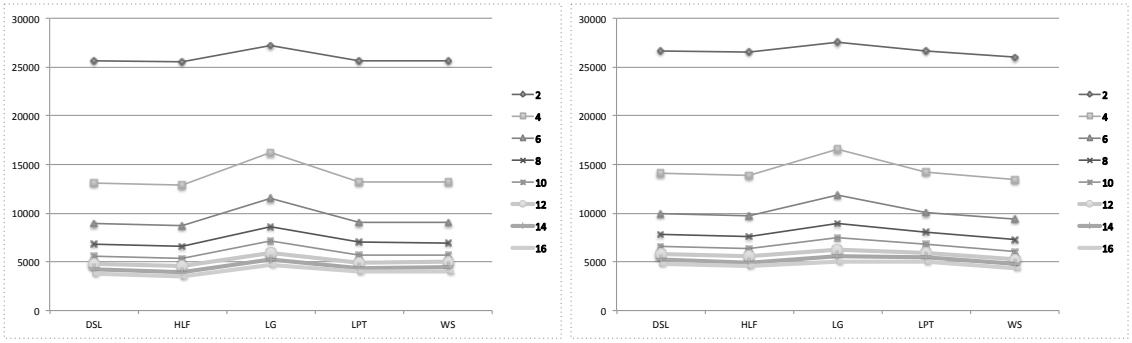


(a) (b)

Figura 1. Resultados da primeira bateria de testes (a) sem e (b) com *overhead*.

Os tempos de execução da Figura 1 refletem a média dos escalonamentos para os 10 grafos. Observa-se que o algoritmo HLF geralmente resulta no melhor escalonamento, pois este é ótimo quando todas as tarefas possuem custo de processamento igual. Estratégias *multithread* dinâmicas (LG e WS) possuem um acréscimo menor no tempo total de execução em relação às demais quando o *overhead* é considerado. O algoritmo LG apresentou os maiores desvios-padrão para os números de processadores usados.

Na segunda bateria de simulações foram gerados 10 DAGs com parâmetros de *comprimento fixo* = 4, *profundidade* = 4, *custo* = 50 e *desvio* = 20. Nestes casos estrutura dos DAGs é fixa, porém os custos das tarefas variam conforme o desvio estipulado. Resultados para estes experimentos são apresentados na Figura 2.



(a) (b)

Figura 2. Resultados da segunda bateria de testes (a) sem e (b) com *overhead*.

Os resultados apresentados na Figura 2 são semelhantes aos anteriores. Destaca-se nestes experimentos o bom desempenho do Work Stealing, com os melhores tempos médios para todos os números de processadores, quando o *overhead* é considerado. Estes resultados indicam uma adequação entre o WS e aplicações com estrutura de *forks* e *joins* aninhados, mesmo quando as tarefas possuem custos de processamento diferentes.

5. Conclusões

Diversas soluções baseadas em algoritmos de lista são conhecidas para o escalonamento de DAGs estáticos. Anahy e Cilk usam variações de algoritmo de lista para o escalonamento de aplicações *multithread* dinâmicas com sucesso. AKSSim é uma ferramenta de simulação que fornece recursos para comparar, para uma mesma aplicação, tempos de escalonamento obtidos por algoritmos de lista projetados para diferentes ambientes de programação e execução.

Os resultados indicam que estratégias de escalonamento de lista são aplicáveis em ambientes *mutithread* dinâmicos, com destaque para o desempenho da técnica de *Work Stealing*, a qual atingiu resultados, muitas vezes, superiores aos de estratégias estáticas. Foi possível observar, também, uma melhor absorção do *overhead* gerado por operações de escalonamento, pelas estratégias *multithread* dinâmicas implementadas por AKSSim.

Referências

Blumofe, R. D. and Leiserson, C. E. (1995). Cilk: an efficient multithreaded runtime system. *ACM SIGPLAN Not.*, 30(8).

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).

Cordeiro, O., Peranconi, D., Villa Real, L., Dall'Agnol, E., and Cavalheiro, G. (2005). Exploiting multithreaded programming on cluster architectures. In *HPCS*, Guelph.

Hu, T. (1961). Parallel sequencing and assembly line problems. *Journal of Operations Research*, 9:841–848.

Sih, G. C. and Lee, E. A. (1993). A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4:175–187.