Estudo da Aplicação de Estratégias de Escalonamento de Lista em Ambientes Multithread Dinâmicos

Cícero Augusto De S. Camargo, Gerson Geraldo H. Cavalheiro

Departamento de Informática – Universidade Federal de Pelotas (UFPEL)
Pelotas – RS – Brasil

cicero.camargo@anahy.org, gerson.cavalheiro@ufpel.edu.br

Resumo. *Um dos maiores desafios na busca de alto desempenho na execução de programas paralelos fazer um escalonamento eficiente das tarefas geradas por esta execução. Este trabalho apresenta um estudo sobre a aplicação de algoritmos de escalonamento de lista, bem como o desenvolvimento de uma ferramenta para a geração de DAGs que representem o modelo do ambiente Anahy e simulação de escalonamentos de lista sobre os DAGs gerados.*

1. Introdução

Algoritmos de escalonamento de lista são citados na literatura [Graham 1969] como eficientes para escalonamento de grafos de fluxos de dados em arquiteturas multiprocessadas. Estes algoritmos procuram otimizar o tempo de execução de uma aplicação partindo do conhecimento do caminho crítico para execução das tarefas deste grafo. Existem diversas estratégias de escalonamento de lista aplicadas a grafos estáticos, algumas destas documentadas em Adam (1974). Estas técnicas baseiam-se em valorar as tarefas do grafo segundo alguma política de prioridade e no uso de uma lista ordenada (segundo o critério de prioridade adotado) para controlar a ativação da execução destas.

Em ambientes como Cilk [Blumofe 1995], onde o grafo é gerado de forma dinâmica, variantes de algoritmos de lista alcançam bons índices de desempenho . Em particular, Cilk utiliza com sucesso a técnica de roubo de trabalho (*work stealing*) em arquiteturas multiprocessadas. Deve-se observar que, em Cilk, técnicas de implementação do núcleo de execução foram desenvolvidas para permitir a execução de um subconjunto de grafos de fluxo de dados, uma vez que a interface de programação deste ambiente permite apenas a descrição de paralelismo aninhado.

Este artigo apresenta um estudo sobre algoritmos de escalonamento de lista voltado a identificar estratégias de execução aplicáveis a ambientes *multithread* dinâmicos. O objetivo é desenvolver estratégias de escalonamento de threads quando estas reagrupam seqüências de tarefas descritas em termos de fluxos de dados. Os resultados serão aplicados no núcleo de escalonamento de Athreads [Cavalheiro 2006].

2. Escalonamento de Listas

Programas paralelos podem ser descritos em termos de um grafo de fluxo de dados entre tarefas, ou, simplesmente, DAG (Directed Acyclic task Graph). Em um DAG, cada nó representa uma tarefa e cada aresta entre dois nós representa dependência de dados entre tarefas. Escalonar um DAG consiste em, dado um conjunto finito de processadores, determinar a seqüência de tarefas que cada processador deve executar em um determinado instante de tempo.

Grande parte dos algoritmos de escalonamento é baseada no escalonamento de lista. Genericamente, esta classe de algoritmos consiste de dois passos: no primeiro passo as tarefas do grafo são classificadas de acordo com alguma política de prioridade e as mesmas tarefas são inseridas em uma lista ordenada por ordem decrescente de prioridade das tarefas; no segundo passo, as tarefas são mapeadas para os processadores disponíveis, levando respeitando a prioridade de execução delas.

Os atributos de prioridade permitem identificar o caminho crítico da execução de tarefas da aplicação descrita no DAG. Dois dos mais importantes atributos para conferir prioridade às tarefas do DAG [Ahmad 1995] são contemplados pela ferramenta: o t_level (*top level*) e o b_level (*bottom level*). O t_level de um nó N_i é o tamanho do maior caminho no DAG, partindo de um nó de entrada até N_i (excluso). Nesta abordagem, o tamanho do caminho entende-se pela soma dos custos de todos os nós e arestas ao longo do caminho. O b_level de um nó N_i é o tamanho do maior caminho no DAG partindo de N_i até um nó de saída. Para a ferramenta, o b_level é um atributo estático relacionado com o caminho crítico do DAG enquanto o t_level é um atributo dinâmico, pois é atribuído depois das tarefas serem escalonadas nos processadores, o que pode fazer com que custos entre tarefas escalonadas em um mesmo processador sejam ignorados.

3. Geração de DAGs e Simulação de Escalonamentos de Lista

O estudo das estratégias de escalonamento foi realizado no contexto do projeto Anahy. Anahy propõe a utilização de um modelo de programação concorrente que permita descrever uma aplicação em termos de um grafo de fluxo de dados, composto por tarefas e identificação de dependências de dados entre tarefas, associado a um modelo de execução baseado em *multithreading* [Valiant 1990] habilitado a limitar o número de execuções paralelas de tarefas em função dos recursos da arquitetura. Athreads [Cavalheiro 2006] é uma implementação de Anahy, na qual tarefas são encapsuladas no contexto de threads e, no suporte a execução, estes threads são executados sobre processadores virtuais.

A Figura 1 representa um DAG gerado pela execução de um programa desenvolvido em Athreads. Observa-se que os threads são *containers* de tarefas, desta forma, o núcleo de execução é capaz de manipular um grafo de threads, reduzindo o sobrecusto de manipulação de um grande número de dependências entre tarefas. Como conseqüência, as estratégias de escalonamentos de lista devem ser adaptadas para obter o melhor proveito desta representação. Para compreender e adaptar estes algoritmos para Athreads, foi desenvolvido um simulador de estratégias de escalonamento de lista. Este simulador é composto de duas partes principais, um gerador de grafos e um aplicador de estratégias de escalonamento. Importante observar que o grafo gerado representa a estrutura implementada em Athreads, sendo as tarefas encapsuladas em threads. Neste grafo, o DAG é descrito pelo relacionamento entre tarefas, sendo que dependências entre tarefas de threads distintos representam as criações e sincronizações (por *join*) entre threads.

A geração do grafo requer parametrização, de forma a caracterizar a estrutura de programa a ser avaliada. São quatro os parâmetros necessários: (*i*) profundidade (P), indicando a profundidade, em termos de threads, que o grafo deve possuir, (*ii*) comprimento (K), número de tarefas que cada thread deve conter, (*iii*) custo de computação (w) e (*iv*) custo de comunicação (c), indicando, respectivamente, o tempo necessário para processamento de cada tarefa e para comunicação de seus resultados.

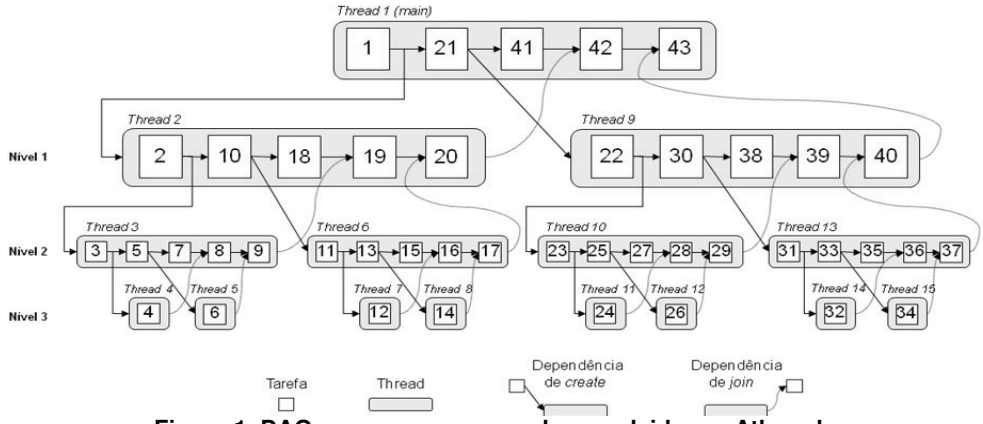

Figura 1. DAG para um programa desenvolvido em Athreads

Observe que o comprimento mínimo de um thread é um (1) e que um thread de comprimento informado *n* terá, de fato, *(2n 1)* tarefas, sendo que *n* tarefas terminam com a construção de um novo thread, outras *n* terminam sincronizando, individualmente, com os threads criados e a tarefa final representa a computação final do thread. Um quinto parâmetro permite indicar diversas organizações para os threads, em função do espaço deste artigo, assume-se uma organização série-paralelo (forks-joins aninhados) dos threads.

A etapa de avaliação inicia valorando cada tarefa do DAG em termos dos respectivos t_level e b_level. Na seqüência são identificados, para cada thread, os valores mínimos e máximos de t_level e b_level obtidos nas tarefas que estas contêm. O passo seguinte consiste em aplicar uma estratégia de escalonamento de lista sobre o DAG e, por fim, indicar, considerando os t_level e b_level mínimos e máximos de cada thread, o escalonamento desejado para os threads.

Tabela 1. Escalonamento do DAG da figura 1 com quatro processadores virtuais

Tempo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Proc1	1:1	1:21	1:41	10:23	10:25	10:27	3:8	3:9	2:19	6:17	2:29	1:42	9:40	1:43
Proc2	-	2:2	2:10	2:18	6:11	6:13	6:15	7:12	6:16	12:26	10:29	9:39	-	-
Proc3	-	-	9:22	9:30	9:38	4:4	5:6	11:24	10:28	14:32	13:36	13:37	-	-
Proc4	-	-	3:3	3:5	3:7	13:31	13:33	13:35	8:14	15:34	-	-	-	-

Enquanto a figura 1 apresenta o grafo gerado pelo simulador, onde os parâmetros utilizados foram: P = 3, K = 2, w = 1 e c = 0, a tabela 1 apresenta o escalonamento obtido para o DAG e para o grafo de threads. Em cada instante de tempo, o processador "Proc*i*", executa um par "thread:tarefa". Para realizar o escalonamento do DAG foi aplicado um algoritmo simples, o algoritmo DLS (*Dynamic Level Scheduling*)[Sih 1993], adaptado para este caso onde não há custos de comunicação entre tarefas. Para o escalonamento dos threads foi aplicado o mesmo algoritmo, considerando os valores mínimos para t_level e b_level.

O resultado foi satisfatório visto que cada processador busca, em um determinado instante de tempo, a tarefa que representa menos custo do ponto de vista do escalonamento dos threads. Por exemplo, nos instantes de tempo 5 e 6, onde há várias tarefas prontas para executar no mesmo instante, o processador Proc1 escolhe executar as tarefas 25 e 27 que possibilitam dar continuidade a execução do thread 10. Como

pode-se observar os processadores Proc2, Proc3 e Proc4 ficam ociosos em períodos de tempo onde não há tarefa pronta para execução.

4. Conclusão

A popularização de arquiteturas multiprocessadas, com o barateamento das configurações baseadas em processadores *multicore*, aumenta a demanda por processamento paralelo. Neste tipo de arquitetura, dotada de memória compartilhada, o modelo de programação mais adequado é o baseado em multiprogramação leve. Athreads é uma ferramenta de programação que representa este modelo de programação e ainda permite a descrição da concorrência em termos de um grafo de fluxo de dados. Resultados de desempenho já documentados deste ambiente mostram sua eficiência, como pode ser visto em Cordeiro (2005). Com o presente trabalho, pretende-se aplicar técnicas de escalonamento de lista dinâmicas no seu núcleo de execução.

O atual estágio de desenvolvimento contempla a introdução de estratégias de escalonamento de lista no simulador. O trabalho prosseguirá com o estudo dos mapeamentos possíveis dos escalonamentos obtidos com estas técnicas para o nível de threads do grafo e, então, na proposição de estratégias dinâmicas de escalonamento de listas para threads e da incorporação destas no núcleo de execução de Athreads.

Referências

Graham, R. L. (1969) "Bounds on Multiprocessing Timing Anomalies", In: SIAM Journal of Applied Mathematics**,** vol. 17, n.2, p.416-429.

Adam, T. L., Chandy, K. and Dickson, J. (1974) "A Comparison of List Scheduling for Parallel Processing Systems," In: Communications of the ACM, vol. 17, n. 12, p.685-690.

Blumofe, R., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E. Randall, K.H. and Zhou, Y. (1995) "Cilk: An Efficient Multithreaded Runtime System", In: *PPOPP '95*, p.207-216, New York, NY, USA. ACM Press.

Cavalheiro, G. G. H., Gaspary, L. P., Cardozo, M. A. and Cordeiro, O. C. "Anahy: A programming environment for cluster computing" In: VECPAR 2006 (LNCS 4395).

Ahmad, I., Kwok, Y. and Wu M. "Performance Comparison of Algorithms for Static Scheduling of DAGs to Multiprocessors", *Proceedings of the 2nd Australian Conference on Parallel and Real-Time Systems*, Sep 1995, p. 185-192.

Valiant, L.G. (1990) "A bridging model for parallel computation" In: *Communications of ACM*, vol. 33, n.8, p.103-111.

Cordeiro, O., Peranconi, D., Villa Real, L., Dall'Agnol, E. and Cavalheiro, G. "Exploiting multithreaded programming on cluster architectures" In: Proc. of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS), Guelph (2005).

Sih, G.C. and Lee E.A. (1993) "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," In: IEEE Trans. On Parallel and Distributed Systems, vol. 4, n.2, p.75-87.