Analisando algoritmos de lista no escalonamento de programas Aiyra

CAMARGO, Cícero Augusto de Souza
CAVALHEIRO, Gerson Geraldo Homrich
Universidade Federal de Pelotas

1 INTRODUÇÃO

Ferramentas que fornecem recursos de programação paralela ganharam atenção com a popularização das arquiteturas *multicore*. Em geral, tais ferramentas consistem em bibliotecas que são ligadas ao código do programa para permitir que o mesmo possua múltiplos fluxos de execução.

A linguagem Aiyra provê uma interface de programação *multithreaded*, disponibilizando, por meio de operações do tipo *create* e *join*, recursos para manipulação de threads. O suporte de execução de Ayra é implementado segundo especificado pelo modelo Anahy (Cavalheiro,2006). A interface de programação oferecida permite que programas paralelos possam ser escritos independente da quantidade e capacidade de recursos de processamento disponíveis. Já o suporte de execução busca explorar eficientemente os recursos disponíveis pelo uso de estratégias de escalonamento. As estratégias implementadas são baseadas em algoritmos de lista (Adam,1974).

Algoritmos de lista são conhecidos por sua comprovada eficiência no escalonamento de programas descritos estaticamente em um grafo de tarefas (Graham,1966). Neste tipo de ambiente a aplicação a ser escalonada é conhecida antes do início da execução e o conceito de caminho crítico pode ser utilizado para estabelecer os limites de desempenho da aplicação. Estratégias de escalonamento de lista podem também ser aplicadas em ambientes dinâmicos. Exemplos encontram-se em Cilk (Blumofe,1995) e Athreads (Cavalheiro,2006) aplicam com sucesso estratégias baseadas em algoritmos de lista no escalonamento de aplicações desenvolvidas sob interface de programação *multithreaded*, porém não fornecem uma faixa de desempenho esperado.

O presente trabalho busca compreender como boa performance pode ser obtida aplicando estratégias de escalonamento de lista sobre aplicações desenvolvidas em Aiyra. Para isso, exemplos apresentados em (Graham,1976) foram mapeados para o contexto de Aiyra e cenários de escalonamento estático e dinâmico foram comparados. O resultado mostrou que, para os exemplos estudados, o núcleo de escalonamento dinâmico de threads de Aiyra mantém a mesma faixa de desempenho esperado para um algoritmo de lista estático.

2 APLICAÇÃO DE ALGORITMOS DE LISTA EM APLICAÇÕES MULTITHREAD

A estratégia de escalonamento aplicada sobre as atividades de um programa paralelo é um ponto chave para o desempenho de execução deste programa sobre uma determinada arquitetura. Esta estratégia é responsável por distribuir o trabalho gerado pelo programa sobre os recursos de processamento disponíveis. Uma estratégia inadequada a um determinado programa pode resultar em um aumento significativo no tempo total de execução. Este trabalho aborda, em particular, estratégias de escalonamento que minimizem o tempo total de execução de um programa paralelo.

Estratégias básicas de escalonamento de lista tem como unidade de escalonamento a tarefa: trecho de código sequencial delimitado por uma entrada e saída de dados. Os algoritmos de lista recebem como entrada um grafo dirigido de tarefas (DAG) que descreve o programa a ser executado. Como saída, o mesmo DAG tem as tarefas valoradas de acordo com alguma política de prioridade. Quando da execução do programa, tarefas são consumidas a medida em que processadores ficam ociosos, respeitando a prioridade das tarefas. Graham (1966) define o *caminho crítico* de um DAG como a seqüência de tarefas do DAG que possui a maior soma de custos de comunicação e processamento. Algumas estratégias priorizam a execução de tarefas neste caminho. Este conceito também permite o estabelecimento dos limites do tempo necessário para escalonar um DAG sobre *m* processadores idênticos.

Aiyra possui um modelo de execução *multithreaded* e sua interface de programação é baseada no uso de duas primitivas de sincronização entre threads: *create* e *join*. Tais primitivas permitem ao programador criar novos threads e a forçar que um thread aguarde o término de um outro thread, ao mesmo tempo que permite a comunicação de dados entre os threads sincronizados. Neste trabalho, um thread representa uma unidade de execução que encapsula sequências de tarefas – trechos de código sequencial entre dois pontos de sincronização.

O núcleo de execução de Aiyra mantém a descrição do relacionamento entre threads como um grafo cíclico de threads (DCG). Este DCG embute o DAG que descreve o fluxo de dados entre tarefas definidas pelo programador. Em Aiyra, a unidade de escalonamento é o thread, ao invés da tarefa. O ambiente de execução implementa o modelo Anahy (Cavalheiro,2006). O algoritmo de escalonamento manipula threads do DCG em listas dinâmicas. Contudo, devido à falta de espaço, a estratégia não será descrita em detalhes neste documento.

A análise feita neste trabalho se dá comparando os ecalonamentos de DAGs estáticos obtidos por Graham (1976) com o tempo obtido escalonando os mesmos programas no modelados em Aiyra. Uma vez que o trabalho de Graham fornece uma boa base para comparação, pois apresenta tempos de escalonamento ótimo para todos os exemplos estudados, bem como estabelece o limite de desempenho de seu algoritmo com base nesses tempos, a comparação serve para mensurar a eficiência da estratégia de escalonamento de Aiyra.

Para tornar a comparação possível, é proposto um conjunto de padrões para transformação de DAGs em DCGs. A Figura 1 mostra os padrões estruturais encontrados em DAGs e as estruturas correspondentes a serem construídas no DCG, para representar o mesmo programa. Círculos representam tarefas e retângulos tracejados os threads gerados. Em cada padrão a tarefa cinza é a que está sendo avaliada. Uma tarefa hachurada já foi parcialmente avaliada, enquanto uma tarefa preta já foi totalmente avaliada.

A transformação é feita em largura no DAG, ou seja, da esquerda para a direita, em níveis. Primeiramente são aplicados os padrões de pré-processamento `Begin`, `End` e `Bunch`. A seguir é iniciado um processamento iterativo, aplicando os padrões `Create`, `Sequence`, `Join`, `Spawn` e `Broadcast`. Esta etapa acaba quando todas as tarefas possuem suas dependências analisadas. O pós-processamento apenas identifica alguma tarefa que casou com o padrão `join` e não possuía nenhuma outra dependência a ser resolvida.

Uma vez que a transformação é feita em largura no DAG, a tendência é de que threads mais à esquerda no DCG contenham mais trabalho. Baseado nisto, uma heurística pode ser a aplicada para determinar que o caminho crítico do DCG se encontra nos threads mais perto da raiz do grafo e mais à esquerda.

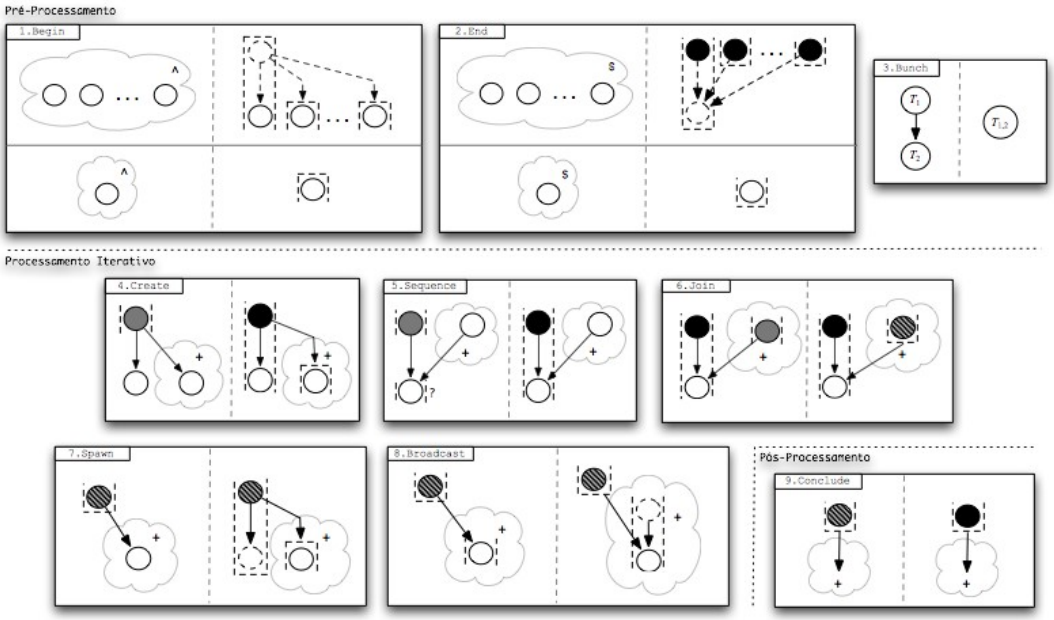

Figura 1: Padrões em um DAG e o segmento correspondente em um DCG.

3 RESULTADOS E DISCUSSÕES

O conjunto de padrões de transformação foram aplicados nos DAGs escalonados nos exemplos 1, 2, 3, 4, 5, 6, 7, 9 e 10, retirados do trabalho de Graham (1976). Chamaremos estes DAGs por G_1, G_2, ..., G_9, e os DCGs obtidos pela transformação destes DAGs por C_1, C_2, ..., C_9. No escalonamento dos DCGs é usado o mesmo número de processadores dos exemplos de Graham e o tempo total de execução pode estar dimensionado em termos do número m de processadores da arquitetura ou em termos de , representando um custo de processamento muito pequeno.

O desempenho dos estudos de caso é apresentado na Tabela 1 em termos de tamanho do escalonamento, ou seja, o tempo total de execução da aplicação correspondente. A primeira coluna indica o exemplo considerado e a segunda, o número m de processadores utilizado. A terceira coluna apresenta o tempo de escalonamento ótimo, apresentado em (Graham,1976), e a quarta o tempo obtido pela estratégia adotada por Aiyra no escalonamento do DCG correspondente. A quinta coluna mostra o tempo de escalonamento do algoritmo de caminho crítico apresentado em (Graham,1976). A última coluna apresenta o tempo obtido por Aiyra sobre o mesmo programa rescrito de forma a tirar vantagem da heurística de escalonamento empregada pelo ambiente.

Os resultados indicam que a estratégia de escalonamento adotada por Aiyra atinge o tempo ótimo para C_1, C_2, C_5 e C_6. Graham (1976) apresenta $|S'(G_8)|$ e $|S'(G_9)|$, tempos de escalonamento de um algoritmo estático que prioriza tarefas no caminho crítico. Assim o tempo obtido para os DCGs C_8 e C_9 é o melhor possível para uma heurística como a empregada em Aiyra, que leva em conta o caminho crítico. Já os grafos C_3, C_4 e C_7 podem ser rescritos sem prejuízo das relações de

precedência entre tarefas, resultando nos grafos C_3', C_4' e C_7'. Estes últimos, quando escalonados pela estratégia de Aiyra, alcançam desempenho ótimo.

Tabela 1: Tempos de Aiyra e Graham para os mesmos programas

DAG/DCG	# m	$\|S(G_i)\|$	$\|S(C_i)\|$	$\|S'(G_i)\|$	$\|S(C_i')\|$
G_1/C_1	3	12	12	-	-
G_2/C_2	2	19	19	-	-
G_3/C_3	m	m	$2m$ 1	-	m
G_4/C_4	m	$m + 2$	$2m$ $1 + 2$	-	$m + 2$
G_5/C_5	m	$m +$	$m +$	-	-
G_6/C_6	m	$m' + 2$	$m' + 2$	-	-
G_7/C_7	m	m	$2m$ 1	-	m
G_8/C_8	m	$(m + 1)(1 +$)	$2m +$	$2m +$	-
G_9/C_9	m	m	$2m$ 1 2	$2m$ 1 2	-

4 CONCLUSÕES

Aiyra é uma linguagem de programação em desenvolvimento que embute recursos de programação paralela e um ambiente de suporte a execução *multithreadeded*. Este trabalho apresentou uma análise sobre o desempenho da estratégia de escalonamento empregada pelo ambiente de execução de Aiyra, que usa uma variação de algoritmo de lista. Foi apresentado, também, um conjunto de padrões que permitem a transformação de programas descritos em grafos acíclicos de tarefas em grafos cíclicos de threads.

Os resultados indicam que a estratégia de escalonamento de listas dinâmicas de threads, empregada por Aiyra, pode alcançar um desempenho tão bom quanto estratégias de escalonamento de listas de tarefas, utilizadas em ambientes estáticos. Para tanto, o programador deve estar ciente da heurística de escalonamento utilizada e descrever a aplicação de modo a tirar proveito disto.

O estado atual do projeto contempla o desenvolvimento de uma ferramenta de simulação que implementa os padrões de transformação apresentados. O objetivo é automatizar a transformação de DAGs em DCGs para facilitar a comparação da estratégia de Aiyra com estratégias que manipulam tarefas. Com tais análises pretende-se fornecer base para um estudo teórico que possa estabelecer os limites de desempenho para programas escritos em Aiyra.

5 REFERÊNCIAS

ADAM, T. L.; CHANDY, K.; DICKSON, J. “A Comparison of List Scheduling for Parallel Processing Systems” In: **Communications of the ACM**, vol. 17, n. 12 1974

BLUMOFE, R.; JOERG, C.F.; KUSZMAU, B.C.; LEISERSON, C.E.; RANDALL, K.H. and ZHOU, Y. “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; CORDEIRO, O. C. “Anahy: A programming environment for cluster computing” In: **VECPAR 2006** (LNCS 4395).

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

GRAHAM, R. L.; Bounds on the Performance of Scheduling Algorithms, chapter 5. John Wiley & Sons, 1976.