Análise do Uso de Work Stealing no Núcleo de Escalonamento do Ambiente Anahy

Cícero Augusto de S. Camargo*, Gerson Geraldo H. Cavalheiro

[1]Departamento de Informática – Universidade Federal de Pelotas (UFPel)
Caixa Postal 354 – 96.010-900 – Pelotas – RS – Brasil

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

1. Introdução

A técnica de *work stealing* [Burton and Sleep 1981, Halstead Jr 1984], roubo de trabalho, é uma das estratégias mais empregadas pelos algoritmos de lista para fazer o escalonamento de tarefas em um ambiente dinâmico, onde o grafo de descrição de dependências (DAG) é gerado em tempo de execução. Os limites de desempenho de tal técnica considerando arquiteturas multiprocessadas foram amplamente estudados, considerando tanto aspectos teóricos [Graham 1969, Shmoys et al. 1995] quanto práticos [Cordeiro et al. 2005, Blumofe and Leiserson 1994]. O principio desta técnica é garantir que, havendo uma tarefa pronta para ser executada em um DAG, não haverá um processador ocioso e que não deverá existir atraso na execução de tarefas que compõem o caminho crítico deste DAG.

A maior sequência de tarefas em um DAG define o caminho crítico da aplicação. A obtenção de índices de desempenho satisfatórios está relacionada com a habilidade da estratégia de escalonamento implementada em garantir que não haverá atrasos na execução das tarefas que compõem este caminho. No entanto, deve ser evitada a inserção de qualquer sobrecusto de execução sobre as tarefas do caminho crítico, uma vez que qualquer *overhead* introduzido implica em aumento do tempo de processamento deste caminho. Atender a este último critério é particularmente necessário quando se trata da aplicação deste mecanismo de escalonamento em um ambiente de execução dinâmico.

Este trabalho trata do estudo da aplicação do *work stealing* como técnica de escalonamento de tarefas descritas em um DAG gerado de forma dinâmica. Os resultados apontam que o ganho de desempenho pode ser obtido pelo encapsulamento de sequências de tarefas em unidades de execução com maior granularidade, denominadas threads. Para realização deste trabalho foi implementada a ferramenta AKSim, Anahy Kernel Simulator, capaz de simular a aplicação de técnicas de escalonamento de lista sobre DAGs que representem o modelo de execução do ambiente Anahy [Cavalheiro et al. 2006].

2. Simulação do ambiente Anahy

A ferramenta AKSim foi desenvolvida para permitir a análise do comportamento do núcleo de escalonamento de Anahy. Este usa uma variação de algortimo de lista, adaptado para um contexto de execução dinâmico. AKSim permite a aplicação de diferentes algoritmos de escalonamento de lista sobre DAGs gerados de forma parametrizada pelo usuário e a realização do escalonamento equivalente considerando que tarefas encontram-se encapsulados no contexto de threads, como definido no modelo de programação de

*Bolsista PIBITI CNPq

Anahy. No modelo, tarefa entende-se por um trecho de código sequêncial presente entre invocações a operações fork e join. Assim, um thread de comprimento n possui $2 \times n + 1$ tarefas, duas tarefas que terminam com a criação de novos threads, duas que terminam com a sincronização dos threads criados e uma última tarefa de conclusão do thread.

Os parâmetros para AKSim são *comprimento*, *profundidade*, *custo unitário*, *número de processadores* e *overhead*. *Comprimento* e *profundidade* descrevem a estrutura de um programa paralelo desenvolvido sob o modelo de programação multithread com fork/join aninhados (*fully strict*). O parâmetro *comprimento* indica quantos novos threads serão criados para cada thread executado. *Profundidade* limita o tamanho do grafo em termos de threads. *Custo unitário* indica a carga computacional associada a cada tarefa no grafo - para efeitos de simplificação neste texto, assume-se que as comunicações tem custo 0 (zero). *Número de processadores* define o paralelismo da arquitetura a ser considerada no escalonamento da aplicação e *overhead* indica que um taxa de overhead está associada à manipulação do grafo pelo núcleo de escalonamento.

O escalonamento é feito, inicialmente, no nível de tarefas. Faz a valoração das tarefas do DAG considerando alguma política de prioridade e aplica um algoritmo de escalonamento de lista ótimo sobre o DAG. Em um segundo momento é realizado um novo escalonamento considerando o nível de threads. Este escalonamento, a cada passo, não utiliza o grafo completo, partindo do princípio que a execução é dinâmica e a criação de threads respeita a ordem temporal representada. O overhead, caso exista, é aplicado apenas à primeira e à última tarefa do thread, uma vez que apenas estas são, de fato, manipulados pelo escalonador. O algoritmo aplicado considera a estrutura fork/join aninhada e assume as seguintes regras:

1. O processador 0 (zero) inicia executando a primeira tarefa do primeiro thread;
2. Um processador quando ocioso busca a partir da raiz do grafo o thread criado a mais tempo;
3. Um processador que executa uma operação join sobre um thread que ainda não tenha sido executado irá interromper a execução do thread corrente, executar o thread sincronizado e, após o término deste, retomar a execução interrompida;
4. Um processador que executa uma operação join sobre um thread que tenha sido iniciado e ainda não tenha terminado busca no grafo o thread mais antigo ainda não executado criado na porção do grafo que possui como raíz o thread sincronizado. Caso nenhum thread esteja disponível nesta região do grafo, o processador entra no modo ocioso.

Esta estratégia assume que o caminho crítico encontra-se nos threads mais antigos, aqueles representados nas regiões superiores do grafo representado na Figura 1. De forma equivalente, caminhos secundários também podem ser identificados assumindo a mesma estratégia em regiões mais internas do grafo. Assim, busca-se priorizar a localidade de referência dos processadores, uma vez que cada processador prioriza a execução de threads definidos em uma mesma região do grafo.

AKSim produz como resultado a representação gráfica do DAG gerado em arquivo em formato UDG, compatível com o software uDraw(Graph), e o escalonamento realizado, saída esta em formato CSV, compatível com editores de planilhas. A Figura 1 apresenta o grafo obtido pela execução de AKSim onde os parâmetros são comprimento 2 e profundidade 2. Nesta figura, as tarefas são representadas por círculos e aquelas

em destaque identificam o caminho crítico. Os threads neste grafo são identificados por sequências horizontais de tarefas. Os números no interior de cada tarefa indicam o thread ao qual ela pertence e um identificador único de tarefa no contexto deste thread.

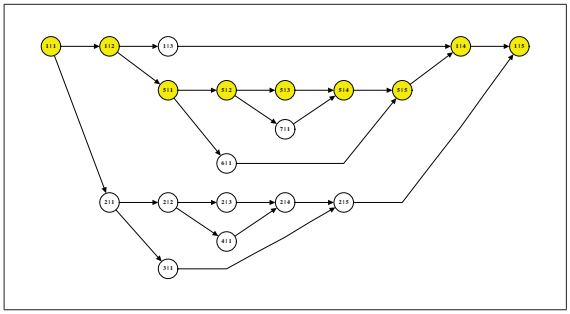

Figura 1. Grafo gerado por AKSim: Profundidade = 2, Comprimento = 2.

3. Resultados

Os resultados que seguem provêm da análise dos escalonamentos obtidos a partir de simulações feitas em AKSim. Estas fazem o escalonamento um grafo de comprimento 3, profundidade 3, custo unitário de 10 u.t. por tarefa, realizado sob 1, 2, 4, 8, 12, 16, 20 e 24 processadores. São apresentados os tempos de escalonamento sem e com overhead. Para estes experimentos assumimos que o overhead tem um custo de 10% do custo de computação da tarefa manipulada no grafo. Os dados para análise estão na Tabela 3. Esta tabela também apresenta o *speedup* obtido pelas execuções paralelas sob o tempo de execução com 1 (um) processador e a relação de ganho percentual atingida pelo uso da abstração de thread como unidade de escalonamento.

	Sem overhead					Com overhead				
Procs	Tarefa	Speedup	Thread	Speedup	Ganho	Tarefa	Speedup	Thread	Speedup	Ganho
1	1180		1180		0	1298		1233		5,01
2	610	1,93	630	1,87	-3,28	671	1,93	659	1,87	1,79
4	340	3,47	360	3,28	-5,88	374	3,47	378	3,26	-1,07
8	230	5,13	230	5,13	0	253	5,13	241	5,12	4,74
12	200	5,9	230	5,13	-15	220	5,9	207	5,96	5,91
16	190	6,21	190	6,21	0	209	6,21	197	6,26	5,74
20	190	6,21	190	6,21	0	209	6,21	197	6,26	5,74
24	190	6,21	190	6,21	0	209	6,21	197	6,26	5,74

Tabela 1. Tempos de escalonamento obtidos por AKSim.

Os resultados indicam que a estratégia adotada por Anahy para o escalonamento dinâmico de threads de uma aplicação descrita sob estrutura de fork/join aninhado é satisfatória. Observa-se que quando o overhead de escalonamento não é considerado, o maior grau de paralelismo observado em nível de tarefa é melhor explorado, permitindo uma melhor utilização dos recursos computacionais disponíveis; em particular verifica-se que os tempos obtidos para o escalonamento sem overhead contando com 8 e 12 processadores não resultou em melhora de desempenho, ocorrendo, inclusive, uma perda de desempenho de 15% pela aplicação do escalonamento em nível de thread.

Já quando o overhead de escalonamento é considerado, a maioria dos casos apresenta um ganho de desempenho pela utilização de threads como unidade de escalonamento. O ganho obtido não é grande, nunca superior a 6% nos experimentos realizados, mas deve ser considerado que este ganho é relacionado ao escalonamento ótimo do DAG. Também foram observados casos onde houve perda de desempenho: uma análise do caso em questão, escalonamento sobre 4 processadores, indica que a impossibilidade de migração de threads já iniciadas resulta em perda de paralelismo de execução e, consequentemente, tempo desperdiçado em processadores.

4. Conclusão

Os algoritmos de lista são considerados eficientes para escalonamento de tarefas descritas em um DAG, sendo seus limites de desempenho conhecidos e comprovados em diversos estudos teóricos e práticos. Shmoys et al [Shmoys et al. 1995] mostrou que estes limites de desempenho não podem ser melhorados, considerando as usuais técnicas de escalonamento, como preempção e migração de tarefas. Uma das técnicas mais populares utilizadas nos algoritmos de lista é o *work stealing*, em particular quando o DAG em questão é composto de tarefas com granulosidade fina.

Este trabalho discutiu a incorporação da componente overhead (sobrecusto de execução) nos custos de escalonamento e a redução do impacto deste sobrecusto na execução de um programa paralelo descrito por um DAG gerado dinamicamente. A metodologia para obtenção destes resultados empregou o uso de AKSim, uma ferramenta de simulação de decisões de escalonamento de DAGs utilizando algorimos de lista. Os resultados obtidos indicam que a utilização de unidades de trabalho de maior granulosidade, threads Anahy, permitem reduzir o número de operações de escalonamento sem perda de desempenho final de execução. Futuramente pretende-se implementar no núcleo de Anahy as estratégias de escalonamento que obtiverem melhores índices em AKSim e fazer a medição dos tempos reais de overhead do ambiente.

Referências

Blumofe, R. D. and Leiserson, C. E. (1994). Scheduling multithreaded computations by work stealing. In *FOCS*.

Burton, F. W. and Sleep, M. R. (1981). Executing functional programs on a virtual tree of processors. In *Proc. of the Conf. on Funct. Program. Lang. and Comp. Architecture*.

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

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

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429.

Halstead Jr, R. H. (1984). Implementation of multilisp: Lisp on a multiprocessor. In *Proc. of the Symp. on LISP and Funct. Program.*

Shmoys, D., Wein, J., and Williamson, D. (1995). Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313–1331.