Programação Paralela Baseada em Tarefas em Arquiteturas com Memória Distribuída

Bruno Gallina Apel[1], **Nicolas Maillard**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{bgapel,nicolas}@inf.ufrgs.br

1. Introdução

Duas perspectivas bem difundidas para obtenção de paralelismo em códigos fonte são o paralelismo de dados, ou *data parallelism* e o paralelismo de tarefas, *task parallelism*. O primeiro tem um forte enfoque em laços e iterações, pois consiste em executar uma mesma operação sobre uma larga quantia de dados, concorrentemente [Hillis and Steele 1986]. Já o paralelismo de tarefas é definido como atribuir uma tarefa, diferente ou não, para os nodos de processamento disponíveis executarem paralelamente [Foster 1994].

Procurando facilitar a criação de programas paralelos, atualmente, o foco é a utilização de *tarefas*, ou *tasks*, como unidade de carga de trabalho paralelo. Este conceito é útil para expressar paralelismo não estruturado e com a possibilidade de criação dinâmica de unidades de trabalho paralelo, em tempo de execução.

2. Conceitos e Trabalhos Relacionados

Inicialmente, deve-se entender o conceito de *tarefa* lógica. Uma tarefa pode ser entendida como uma sequência de código que não pode ser paralelizado, possui uma entrada e retorna algum resultado. Quando não há dependência entre as tarefas, estas podem ser executadas paralelamente, e, se há mais de uma aguardando execução, teremos um ”*pool*”de tarefas a gerenciar.

Este conceito de tarefa permitiu a criação de algoritmos que praticam o ”roubo de tarefas”(*work stealing*). Estes são conhecidos pelo fato de que os nodos de processamento ociosos procuram ”roubar”a carga daqueles que estão sobrecarregados. Para que o roubo seja realizado, cada nodo de processamento possui uma pilha para armazenamento das tarefas criadas, a qual os nodos de processamento ociosos verificam para identificar a existência de tarefas em espera de execução. Após, estas são migradas para os nodos de processamento ociosos, tomando-se cuidado para que não haja nenhuma incoerência com os resultados esperados, atualizando apontamentos e identificadores necessários.

Dentre as propostas e ferramentas existentes na literatura, convêm citar CILK, Intel Threading Building Blocks, OpenMP e MPI, que possuem ao menos uma característica a ser analisada, como o uso de tarefas genéricas, do algoritmo de roubo de tarefas ou a comunicação em arquiteturas de memória distribuída.

A primeira, CILK [Blumofe et al. 1995], pode ser definido como um pré-compilador da linguagem C que emprega um algoritmo de escalonamento de roubo de *threads*, aleatorizado, eficiente em tempo e ótimo em espaço. Funciona de maneira adequada em ambientes com processadores heterogêneos, mas não foi projetado para ambientes que utilizem memória distribuída.

Completamente baseado no conceito de tarefa lógica, o Threading Building Blocks (TBB) [Reinders 2007], da Intel, é uma biblioteca de alto nível para o modelo de memória compartilhada baseada em *templates* de C++. Apresenta suporte para decomposição de dados e controle de granularidade automática, além de balancear a carga ao utilizar um escalonador de tarefas, integrado com a técnica de roubo de tarefas.

Uma das interface mais difundidas, e efetivamente utilizadas, para comunicação em memória compartilhada é o OpenMP [Board 2008]. Inicialmente, possuía foco em paralelização estruturada, ou seja, laços, porém, após a disponibilização de sua terceira versão, esta foi acrescida com a possibilidade de utilização de paralelismo aninhado e criação dinâmica de tarefas.

Dentro do ambiente de programação em memória distribuída, o padrão *de facto* para comunicação por troca de mensagens, hoje, é MPI [Forum 2008]. Sua extensão, MPI-2, possibilita a criação dinâmica de processos. A forma de escalonamento dos processos criados dinamicamente não é abordada em sua norma e, também, ainda não há suporte a utilização de tarefas de forma nativa.

3. Objetivos e Trabalhos Futuros

Pretende-se mostrar que se pode usar paralelismo de tarefas em ambientes com memória distribuída, com desempenho no mínimo tão bom como programas MPI tradicionais, e possibilitando a paralelização de algoritmos irregulares. Para possibilitar tal paralelismo, serão definidas primitivas de comunicação entre as tarefas.

Para alcançar este objetivo será desenvolvida uma API, com ênfase em facilidade de programação, utilizando orientação a objetos, C++ em conjunto com primitivas MPI, visando facilitar a utilização com algoritmos não estruturados, com domínio de iteração não regular. Além de esconder do usuário final questões técnicas mais específicas a comunicação entre nodos, como serialização e migração de tarefas. E, para critérios de validação, alguns problemas tradicionais a programação paralela serão utilizados como benchmarks.

Referências

Blumofe, R. D., 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: Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 207–216, New York, NY, USA. ACM.

Board, O. A. R. (2008). Openmp application program interface - version 3.0. Technical report.

Forum, M. P. I. (2008). MPI: A Message-Passing Interface Standard ver. 1.3. Technical report, Message Passing Interface Forum.

Foster, I. (1994). Task parallelism and high-performance languages. *Parallel Distributed Technology: Systems Applications, IEEE*, 2(3):27.

Hillis, W. D. and Steele, Jr., G. L. (1986). Data parallel algorithms. *Commun. ACM*, 29(12):1170–1183.

Reinders, J. (2007). *Intel threading building blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition.