Análise de Desempenho de Aplicações Paralelas do Padrão *Pipeline* em Processadores com Múltiplos Núcleos

Giuseppe G. P. Santana[1], **Luís F. W. Goés**[1]

[1]Departamento de Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC-Minas)

giuseppe.santana@sga.pucminas.br, lfwgoes@yahoo.com.br

Abstract. *This paper presents a performance analysis of the pipeline parallel applications over a multi-core architecture, using TBB (Threading Building Blocks Intel) in two different applications, dedup and ferret that are provided in the PARSEC benchmark.*

Resumo. *Este artigo apresenta uma análise de desempenho de aplicações do padrão pipeline sobre uma arquitetura de múltiplos núcleos, utilizando-se o TBB (Intel Threading Building Blocks) em duas aplicações distintas, o dedup e ferret que são disponibilizadas no benchmark PARSEC (Princeton Application Repository for Shared-Memory Computers).*
Palavras-chave: *padrão pipeline; programação paralela; TBB.*

1. Introdução

Atualmente, a melhoria de desempenho na computação está relacionada aos vários métodos de paralelização de aplicações em arquiteturas com múltiplos núcleos. [McCool 2010].

As aplicações paralelas possuem características comuns que podem ser agrupadas em padrões paralelos ou esqueletos algorítmicos. Porém, o programador enfrenta várias dificuldades na programação paralela, tais como, sincronização, comunicação entre *threads*, concorrências de dados e recursos entre outros [McCool 2010].

A programação paralela se concentra principalmente em dois segmentos principais, o desenvolvimento de bibliotecas paralelas otimizadas que encapsulam o paralelismo de dados ou de tarefas, e a criação de novas linguagens de programação paralela [Navarro et al. 2009].

Vários métodos de paralelização foram implementados nos últimos anos para facilitar ao programador sua utilização, tornando explícito a paralelização das aplicações através de diretivas ou métodos, fazendo com que os modelos de programação paralela fiquem em um alto nível de abstração. O *Threading Building Blocks* (TBB) e o OpenMP são exemplos de modelos de programação paralela para múltiplos núcleos, que tornam explícito o uso de paralelismo [Vasconcelos and Maillard 2009].

O TBB foi projetado para suportar paralelismo de dados e tarefas aninhado implementando tarefas para equilibrar a carga entre os núcleos de processamento disponíveis, a fim de aumentar a utilização dos núcleos e, portanto, a escalabilidade das aplicações. O modelo de programação TBB possui em sua biblioteca funções que implementam alguns padrões de programação paralela como *scan*, *stencil*, *reduce* e o *pipeline*.

Especificamente, o padrão de programação paralela *pipeline* é uma técnica utilizada para dividir o problema em uma série de tarefas para aumentar o desempenho das aplicações, uma vez que múltiplas instruções são executadas simultaneamente [Navarro et al. 2009].

Este trabalho tem como objetivo realizar uma análise de desempenho de aplicações paralelas do padrão *pipeline* em processadores com múltiplos núcleos. Para a avaliação de desempenho foi utilizado o **PARSEC** *benchmark* executando as aplicações do padrão *pipeline dedup* e *ferret* com o TBB.

A estrutura do trabalho consiste na seguinte organização: na seção 2 são apresentados os trabalhos relacionados, na seção 3, o referencial teórico, na seção 4, a metodologia de avaliação, na seção 5, resultados experimentais e na seção 6, a conclusão.

2. Trabalhos Relacionados

Em [Bhadauria et al. 2009] é realizado um estudo para compreender o desempenho do PARSEC nos contemporâneos *Chip Multiprocessor* (CMP), onde as aplicações são expostas a diferentes tipos de memória, padrões de compartilhamento de dados e cargas de trabalho. Ao estudar esse comportamento foram identificados gargalos nas aplicações do PARSEC, principalmente nas operações de entrada e saída de dados.

Em [Navarro et al. 2009] foi utilizado o paralelismo de tarefas usando o padrão de programação pipeline, abordando duas aplicações do *benchmark* PARSEC, o *dedup* e o *ferret*, onde a primeira aplicação envolve a compressão do fluxo de dados e a segunda aplicação implementa pesquisa de imagens similares. Para a aplicação *ferret* ocorre o gargalo na etapa de entrada de dados nas operações de E/S enquanto o *dedup* apresenta o gargalo na saída de dados nas operações de E/S. Dada essa situação, foi realizada uma modelagem analítica para o padrão de programação paralela pipeline utilizando teoria de filas e também a implementação das aplicações em *Pthreads* e no TBB, onde os dois principais problemas identificados que limitam a escalabilidade foram o desequilíbrio de carga e o gargalo apresentado nas operações de E/S.

Em [Feldman et al. 2013], o trabalho apresenta uma utilização eficaz dos dados *non-blocking* em estruturas de um aplicativo de deduplication, o estudo realizado foi sobre as formas de integração de estruturas de dados *non-blocking* sobre o *benchmark* PARSEC com a aplicação *dedup*. O ganho de desempenho ao utilizar as estruturas de *non-blocking* depende muito da distribuição de operações sobre os dados e a concorrência entre eles, mostrando-se satisfatório quando utilizado estruturas de *hash map*.

Neste trabalho é realizada uma análise de desempenho das aplicações *dedup* e *ferret* do PARSEC *benchmark* utilizando o TBB, estudando o comportamento em cada estágio do *pipeline*.

3. Referencial Teórico

3.1. Padrões de programação paralela

Os padrões de programação são soluções gerais para os diversos problemas encontrados na computação, seja sua execução de maneira sequencial ou paralela.

Os padrões de programação paralela são motivados pelo ganho de desempenho das aplicações, no entanto sua implementação é uma tarefa difícil. Os padrões de

programação seguem a descrição ou a implementação de um padrão de linguagem, que envolve várias abordagens da computação e auxiliam no desenvolvimento de soluções para os problemas e na sua implementação, tornando mais fácil o entendimento de aplicações que são escritas de forma paralela [Griebler et al. 2011].

Com a utilização dos padrões de programação paralela é possível a criação de esqueletos e *frameworks* que facilitam a implementação de aplicações em paralelo. Contudo faz-se necessário o entendimento dos problemas que podem ocorrer no processo de paralelização das aplicações, como a concorrência de dados e a estruturação das tarefas [Griebler et al. 2011]. Os padrões mais comumente encontrados são: *Pipeline*, *Stencil*, *Reduce* entre outros.

O padrão de programação *pipeline* divide a aplicação em uma série de etapas que necessitam ser completadas uma após a outra. Os núcleos de processamento funcionam como estágios para processar as aplicações que foram divididas. Cada estágio opera sobre todo o problema e quando necessário trocam informações com os estágios subsequentes [Schepke 2009]. Assim não é necessário esperar que uma aplicação seja totalmente processada para iniciar o processamento de mais uma aplicação, pois como a aplicação foi dividida em partes essas são executadas paralelamente [Baldo 2007].

3.2. *Threading Building Blocks*

O Threading Building Blocks (TBB) é um modelo de programação que foi projetado para suportar paralelismo aninhado e recursivo. O TBB implementa tarefas para equilibrar a carga entre os núcleos de processamento disponíveis, a fim de aumentar a utilização dos núcleos e, portanto, a escalabilidade das aplicações [Reinders 2007].

As aplicações geralmente possuem comportamentos diferentes em seu processamento e ao se utilizar o TBB, inicialmente a carga é dividida igualmente entre os núcleos dos processadores disponíveis, podendo variar o número de *threads* para a divisão da carga. Como cada divisão de carga tem suas particularidades de processamento, um núcleo pode processar a sua carga e terminar em um menor tempo se comparado com os outros núcleos, podendo ocorrer uma sobrecarga de tarefas em alguns núcleos e outros ficarem ociosos, porém o TBB redistribui a carga do núcleo que está saturado para o núcleo ocioso, ganhando em desempenho de processamento. Essa técnica é chamada de roubo de tarefas (*work-stealing*) [Reinders 2007].

4. Metodologia de Avaliação

O método de avaliação desse trabalho é analisar o desempenho das aplicações *dedup* e *ferret* do *benchmark* PARSEC utilizando o TBB com diferentes números de *threads*, usando entradas de dados sugeridas do próprio *benchmark* PARSEC.

4.1. Ambiente Experimental

Para compor o ambiente experimental foi utilizado um computador com o processador *Intel Core i5 750 2.67 GHz* com quatro núcleos físicos e oito lógicos, uma memória de 12 *GBytes* rodando em um sistema operacional *Linux Ubuntu 13.10*.

4.2. Benchmark PARSEC

O *benchmark* PARSEC foi criado com o objetivo de selecionar aplicações diversificadas para estudos científicos. Essa carga de trabalho foi selecionada a fim de incluir diferentes combinações de modelos e padrões paralelos, entre eles o modelo TBB e o padrão *pipeline*.

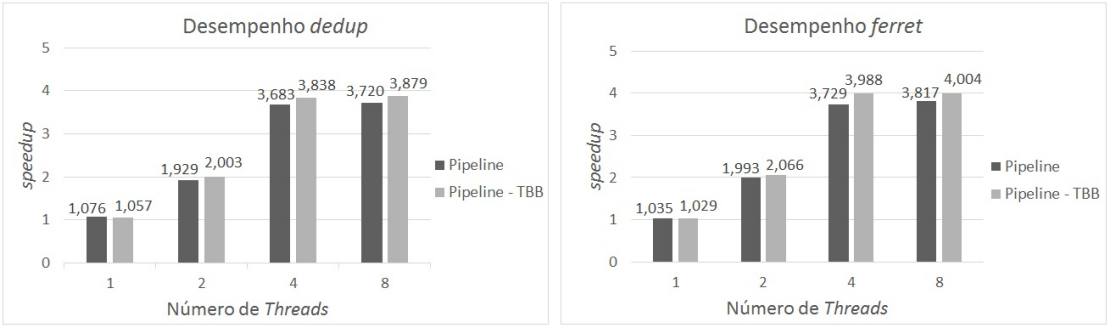
O PARSEC consiste de nove aplicações e três núcleos que divide as aplicações em tópicos e distribui a carga dinamicamente. Todas as aplicações são paralelas e a carga de trabalho foi diversificada para atender a demanda de processamento de mídia, visão computacional, computação financeira, servidores corporativos e animação física, dentre elas é possível destacar as aplicações do padrão *pipeline*, o *dedup* e o *ferret*.

A aplicação *ferret* está inclusa no *benchmark* PARSEC e sua execução baseia-se na pesquisa de similaridade pelo conteúdo de dados como imagens, áudio, vídeos formas 3D entre outros, a partir de um conjunto de dados na entrada. O *ferret* é composto em seis estágios de *pipeline*, onde o primeiro e último estágio são processados sequencialmente [Navarro et al. 2009].

O *dedup* é um kernel que utiliza uma compressão de dados de gerações próximas, esse método é chamado de ”deduplication”. O método combina compressão de dados local e global para alcançar melhores taxas de compressão. O *dedup* é um método padrão para sistemas de armazenamento de backup e para largura de banda otimizada em dispositivos de rede. As entradas da aplicação são executadas através de cinco estágios de *pipeline* sendo as três intermediárias paralelas [Bienia et al. 2008].

5. Resultados Experimentais

A métrica utilizada neste trabalho é o ganho de desempenho (*speedup*), onde é calculado pelo tempo de execução sequencial dividido pelo tempo de execução paralela. As aplicações foram executadas com 1, 2, 4 e 8 *threads*. As aplicações *dedup* [Chen 2011] e *ferret* foram executadas 10 vezes e calculada a média aritmética do tempo de execução em segundos. Foi utilizada uma base de dados de tamanho 672 MB para a aplicação *dedup* e uma base de dados contendo 59.695 imagens distintas com 3.500 imagens de consulta com o resultado de 50 imagens mais semelhantes. O ganho de desempenho da aplicação *dedup* é apresentado na Figura 1(a) e o ganho de desempenho da aplicação *ferret* é apresentado na Figura 1(b).



(a) (b)

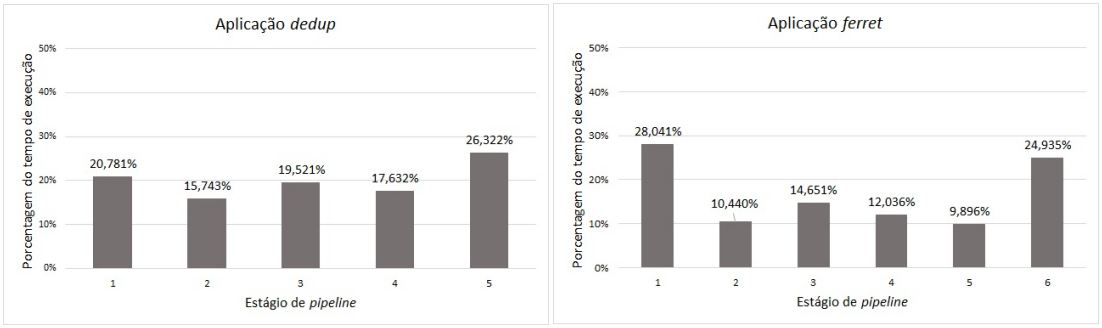
Figura 1. *Speedup* das aplicações *dedup* (a) e *ferret* (b) variando-se o número de *threads*.

Através da Figura 1(a) é possível notar que a aplicação alcançou um *speedup* máximo de 3,879x utilizando 8 *threads* com a implementação do TBB. Para a aplicação sendo executada utilizando *Pthreads* e 8 *threads* o ganho de desempenho máximo foi de 3,720x.

O TBB apresenta, na média, uma tendência a possuir um melhor resultado que a versão *Pthreads*, devido ao roubo de tarefas que é oferecido pela biblioteca, pois para realizar a compressão de dados pela a aplicação *dedup* as cargas entre *threads* são desbalanceadas.

Através da Figura 1(b) é possível notar que a aplicação alcançou um *speedup* máximo de 4,004x utilizando 8 *threads* sendo implementado no TBB. Para a mesma aplicação sendo executada somente com *Pthreads* e 8 *threads* o ganho de desempenho máximo foi de 3,817x.

Como a aplicação *ferret* analisa a semelhança de imagens a partir da comparação entre pixels, uma certa imagem pode levar mais tempo que outra para ser determinado sua semelhança ou não, gerando assim uma carga de trabalho desbalanceada entre as *threads*, assim o TBB apresenta melhores resultados devido ao roubo de tarefas, reduzindo o tempo ocioso entre as *threads*.



(a) (b)

Figura 2. Porcentagem do tempo de execução das aplicações *dedup* (a) e *ferret* (b) para cada estágio do *pipeline*.

Pode-se observar nas Figura 2(a) e 2(b) que a maior parte do tempo de execução das aplicações decorre no primeiro e ultimo estágio do *pipeline* realizando as operações de E/S que são executadas de maneira sequencial, por este motivo consomem maior tempo nas duas aplicações mesmo utilizando o TBB e *Pthreads*, tornando-se o gargalo.

6. Conclusão

Este trabalho apresentou uma análise comparativa das aplicações do padrão de programação paralela *pipeline*, sobre uma arquitetura de múltiplos núcleos utilizando o TBB, em duas aplicações do Benchmark PARSEC *dedup*, utilizado para compressão de dados e o *ferret*, utilizado para busca por similaridade. Os resultados mostram que, ambas as aplicações apresentam melhor desempenho ao utilizar o padrão de programação paralela *pipeline*, comparado com a versão sequencial.

Como as aplicações realizam sua resolução em etapas, à medida em que aumenta-se o número de *threads*, o desempenho das aplicações tende a aumentar mais utilizando o TBB, pois as aplicações se mostraram desbalanceadas em sua execução, contudo o ganho de desempenho não foi proporcional para 8 *threads* pois o processador possui somente 4 núcleos físicos e não consegue escalonar as *threads* para obter o máximo de desempenho, ou seja, o ganho máximo é dado pela quantidade de núcleos que possui o processador para a criação das *threads*. Em geral, o ganho de desempenho do TBB está relacionado a

sua capacidade de roubo de tarefas, que reduz o tempo em que cada *thread* pode ficar em tempo ocioso aumentando o desempenho.

Para ambas as aplicações é utilizada a técnica de *pipeline* dividindo as etapas para obter ganho de desempenho, porém como nos estágios de *pipeline*, os tempos de execução são distintos as *threads* tendem a ficar ociosas, o mecanismo de roubo de tarefas do TBB ajuda na melhoria do desempenho reduzindo esse tempo, pois as mesmas possuem uma característica de cargas desbalanceadas, no *dedup* por tamanhos de arquivos diferentes e no *ferret* por encontrar imagens semelhantes varrendo a imagem total ou parcialmente.

Durante a execução das aplicações *dedup* e *ferret* é perceptível que as operações de E/S demandam maior parte do tempo para executar tornando-se o gargalo das aplicações, pois esses estágios são executados sequencialmente.

Para trabalhos futuros pretende-se utilizar o tempo de execução de cada estágio do *pipeline* para atribuir dinamicamente a quantidade de *threads* necessárias para maximizar a execução das aplicações, sendo diretamente proporcional a alocação de *threads* para cada estágio em que a aplicação consumir maior quantidade de tempo.

Referências

Baldo, L. J. (2007). Predição de desempenho de aplicações paralelas para máquinas agregadas utilizando modelos estocásticos. In *Dissertação de Mestrado, em: UFRGS - Instituto de Informática Programa de Pós-Graduação em Computação*.

Bhadauria, M., Weaver, V. M., and McKee, S. A. (2009). Understanding parsec performance on contemporary cmps. In *IISWC 2009. IEEE International Symposium on*, pages 98–107. IEEE.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The parsec benchmark suite: Characterization and architectural implications. In *PACT'08*.

Chen, N. (2011). Parsec pipeline parallelism partial implementation of dedup. URL: https://github.com/vazexqi/ParsecPipelineParallelism/. Acesso em: 17 mar 2014.

Feldman, S. D., Bhat, A., LaBorde, P., Yi, Q., and Dechev, D. (2013). Effective use of non-blocking data structures in a deduplication application. In *SPLASH '13 Conference on Systems, Programming, & Applications*, pages 133–142. ACM.

Griebler, D., Raeder, M., and Fernandes, L. G. (2011). Padrões e frameworks de programação paralela em arquiteturas multi-core. In *SBC*.

McCool, D. M. (2010). Structured parallel programming with deterministic patterns. In *HotPar-2nd USENIX Workshop on Hot Topics in Parallelism*.

Navarro, A., Asenjo, R., Tabik, S., and Cascaval, C. (2009). Analytical modeling of pipeline parallelism. In *PACT'09.*, pages 281–290. IEEE.

Reinders, J. (2007). Intel threading building blocks: Multi-core parallelism for c++ programming. In *O'Reilly*.

Schepke, C. (2009). Ambiente de programação paralela. In *Dissertação de Mestrado, em: UFRGS - Instituto de Informática Programa de Pós-Graduação em Computação*.

Vasconcelos, F. B. and Maillard, N. (junho, 2009). Programando com gpu's: Paralelizando o método lattice-boltzmann com cuda. In *Trabalho de Conclusão de curso: Monografia. Em: UFRGS, Porto Alegre*.