Processamento Paralelo Fazendo Uso Do OpenMP

Rogerio Lazzari Junior, Carlos Alberto Gasperin

Uniftec Caxias do Sul – Engenharia de Computação

geiolaz@gmail.com, carlos\_gasperin@acad.ftec.com.br

**Resumo**: Partindo de técnicas de programação paralela fazendo uso do framework OpenMP, este trabalho aborda de forma prática e simples o processamento e trabalho com imagens em uma plataforma Windows com múltiplos núcleos de processamento usando threads. Com o uso da biblioteca OpenCV, o desempenho de um programa modificador de imagens foi medido com a utilização de quantias variadas de threads e analisado.

**Palavras-Chave**: Paralelismo, C++, OpenMP

**1 Introdução**

Para a análise de desempenho gerada pelo processamento paralelo, um programa foi utilizado e quantias diferentes de threads foram utilizadas.

Os testes foram executados utilizando um servidor equipado com processador Intel Xeon E5620 de 4 núcleos físicos e 4 núcleos lógicos, totalizando 8 unidades de processamento que serão responsáveis por executar os diferentes threads criados pela aplicação. O servidor conta também com 16GB de RAM e no momento de testes roda Windows server 2012 R2 Standard.

A biblioteca utilizada para a paralelização foi o OpenMP, biblioteca desenvolvida pela Intel visada para facilitar a programação paralela.

“[…] most of the high-level, portable and/or standard parallel programming models are designed for distributed memory systems. This has resulted in a serious disconnect between the state of the hardware and the software APIs to support them. The goal of OpenMP is to provide a standard and portable API for writing shared memory parallel programs.”Chandra.

**2 Processamento Paralelo Fazendo Uso Do OpenMP**

A análise do diferencial de desempenho de programação paralela se baseia em um programa de autoria própria que varre uma imagem de tamanho qualquer, dividindo a imagem por linhas de pixels e distribuindo as linhas de pixels aos threads criados. Uma simples multiplicação é feita nos valores RGB pixel a pixel pelos threads e uma vez que o mesmo chega ao final da linha, ele inicia o processamento em outra linha ainda não atribuída a nenhum thread.

Uma vez que a execução do código paralelo inicia, o tempo de execução é registrado pelo programa e extraído para análise e comparação.

“[…] parallel computing can have an enormous impact on application performance, and OpenMP facilitates access to this enhanced performance. Can any application be altered to provide such impressive performance gains and scalability over so many processors? It very likely can.” Chandra.

Os dados coletados da execução do programa fazendo uso de quantias de 1 a 32 threads se apresentam no Quadro 1.

|  |  |
| --- | --- |
| Threads | Tempo de execução |
| 1 | 80296 ms |
| 2 | 40232 ms |
| 4 | 22450 ms |
| 8 | 17727 ms |
| 16 | 18727 ms |
| 32 | 18824 ms |

Quadro 1: Dados de execução

O programa exemplo utilizado tem grande benefício ao se fazer uso da programação paralela, pois por natureza, o tratamento de imagens é extremamente paralelizável. Fazendo uso da CPU, o tratamento de imagens se apresenta de forma de desempenho máximo utilizando todos os núcleos do processador, mas em um ambiente de produção real, a utilização de GPUs seria ideal, pela contagem superior de núcleos (mesmo esses sendo “mais fracos” do que os de uma CPU. <Skuhersky, Página 44>

**3 Discussão**

Os dados coletados a partir da execução do programa exemplo se apresentam plotados na Figura 1. Os dados coletados apresentam grande fidelidade com a teoria, sendo que a utilização de mais threads condiz com um ganho imensurável de desempenho até o momento que se tem mais threads do que núcleos de processamento, momento que o desempenho volta a cair.



Figura 1: Threads X Tempo de execução

A perda de desempenho notada utilizando 32 e 16 threads em comparação com 8 threads se dá ao fato que o computador utilizado para a captura de dados contém 4 núcleos fisicos e 4 núcleos lógicos, totalizando 8 cores. A utilização de um número maior de threads do que núcleos reais na máquina começa a trazer uma perda de desempenho, pois o sistema operacional começa a fazer o swap entre threads ativos em execução no processador por outros threads ativos, antes da finalização dos mesmos.

**4 Conclusão**

Através dos testes realizados, fica claro que a utilização de múltiplas threads reduz significantemente o tempo de processamento (de códigos com natureza paralelizável), porém apenas até certo ponto. Uma vez que o processador está “saturado”, a utilização de mais threads não é mais benéfica e se torna prejudicial.

Levando em consideração que nem todo código é paralelizável (todo programa contém partes seriais), o openMP é um ótimo candidato para melhorar o desempenho dos setores paralelizados de programas. Seu uso correto traz resultados notáveis e não é de difícil implementação.

**5 Referências Bibliográficas**

**Parallel Programming in OpenMP** Por Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Ramesh Menon, Jeff McDonald.

**Introduction to Parallel Computing** Por Michael Skuhersky.