Redução do Tempo de Execução de Métodos Numéricos Utilizando GNU/Octave e GPGPU[1]

André Luís Tibola[2]; Vinícius Menezes de Oliveira[3]

Introdução

Nos últimos anos popularizou-se o emprego de arquiteturas alternativas, na computação de alto desempenho, com melhor relação de preço por Gigaflop como é o caso mais recentemente do *Cell* e das *GPU*. A taxa de acréscimo no poder computacional das GPU(*Graphics Processing Unit*), quando comparado ao das CPU, tem sido notoriamente maior [1] o que desperta grande interesse na utilização delas para uso geral. As GPU são simplificadamente um conjunto de processadores *MIMD* [2] capazes de executar com melhor desempenho algoritmos *data-parallel* [3], e após a arquitetura de shaders unificados em VGAs, com posterior lançamento do toolkit NVIDIA CUDA para programação de placas gráficas, é possível programá-las de forma tão conveniente, quanto outras tecnologias para processamento paralelo de alto desempenho.

O GNU/Octave (ou simplesmente Octave) é um software de processamento numérico que implementa uma linguagem matricial própria [4], de modo geral compatível com o Mathworks Matlab®, e possui muitas rotinas implementadas para utilização em variadas áreas. Devido à estrutura do Octave, que utiliza bibliotecas para prover parte das funcionalidades e pelas suas possibilidades de extensão [4], podemos utilizar computação em GPU para acelerar o processamento do Octave.

Metodologia

Em nossa aplicação utilizamos computação em GPU para obter ganhos no tempo de execução dos métodos numéricos no Octave, optamos por otimizar as Transformadas de Fourier utilizando extensões do tipo *".oct"* e também acelerar operações de álgebra linear com a criação de uma implementação da BLAS(Basic Linear Algebra Subprograns) para substituir a padrão. A Figura 1 mostra a estruturação do Octave após nossas modificações, os blocos tracejados representam elementos modificados ou adicionados.

Nossa proposta preza pela não criação de dependências por parte do usuário com tecnologia utilizada, também, a abordagem proposta é um método totalmente transparente ao usuário, de otimizar códigos já existentes de Octave, podendo ele utilizar desses recursos sem nenhuma modificação em seus *scripts*.

As medidas apresentadas são tomadas diretamente no Octave a fim de incluir todos os *Overheads* provenientes do ambiente de execução. Foram tomadas 10 amostras para cada tamanho de instância indicado, com matrizes quadradas, variando o tamanho da entrada de 500.000(quinhentos mil) unidades a cada teste, com testes desde 500.000(quinhentos mil) até 45.000.000(quarenta e cinco milhões) de elementos. É apresentado o *SpeedUP* observado das implementações, dado

[1] Projeto 196925/2008

[2] Estudante de Eng. da Computação; Universidade Federal do Rio Grande; e-mail: altibola@gmail.com

[3] Professor Dedicação Exclusiva; Universidade Federal do Rio Grande; e-mail: vinicius@ieee.org

pela fórmula Sp = Ts/Tp, que expressa o número de vezes que uma implementação é mais rápida que outra. A implementação referência é a padrão do Octave.

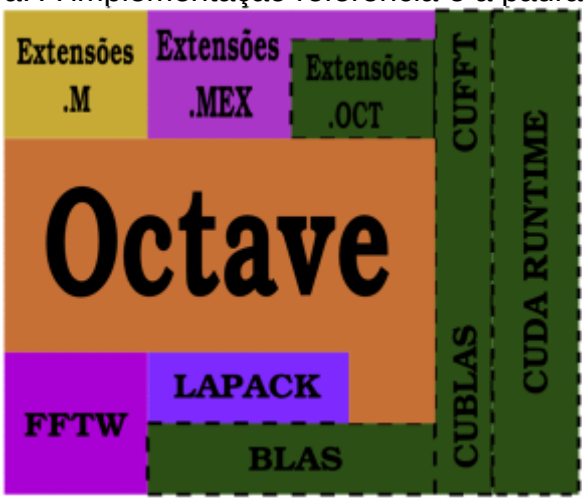

Figura 1: Estrutura do Octave

Resultados e Discussão

A Figura 2 mostra a variação do SpeedUP em função da variação do tamanho da entrada, para a operação de multiplicação de matrizes no Octave. O cenário utilizado para os testes consiste de processador AMD Athlon X2 3800+ e VGA XFX Nvidia Geforce 8800GS. É apresentado o SpeedUP da Blas proposta(GPU BLAS) e da BLAS ATLAS[4]. Podemos notar o Speedup crescente e com pequenas variações negativas, representando baixa degradação da performance com aumento da instância. A baixa variância nas medições demonstra também a confiabilidade para aplicação da solução.

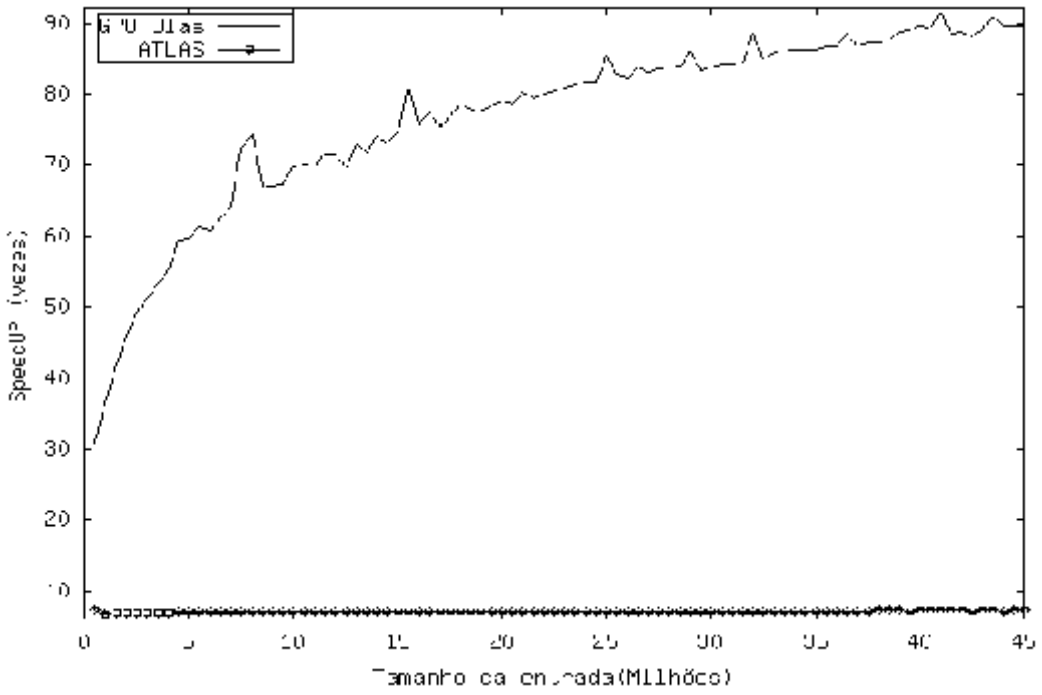

Figura 2: SpeedUP Observado

[4] Automatically Tuned Linear Algebra Software – Uma implementação otimizada para CPU

A GEMM(*General Matrix Multiply*) é o principal bloco de construção das rotinas de álgebra linear computacional, sendo ponto fundamental em qualquer distribuição BLAS. Explicitamos dessa forma, a viabilidade de otimizar o GNU/Octave utilizando-se GPGPU, com ganhos satisfatórios através de nossa abordagem.

Agradecimentos

Agradecemos ao Programa Institucional de Bolsas de Iniciação Científica da Universidade Federal do Rio Grande – PROBIC-FURG, financiadora da bolsa de pesquisa do aluno André Luís Tibola.

Referências

[1] NVIDIA, NVIDIA CUDA Programming Guide, 26/06/2009, Version 2.2.1.

[2] Flynn, M., “Some Computer Organizations and Their Effectiveness”, IEEE Trans. Comput., Vol. C-21, pp. 948, 1972.

[3] Hillis, W. Daniel and Steele, Guy L., “Data Parallel Algorithms”. Communications of the ACM, vol. 29, no. 12, Dec, 1986, pp 1170-1183.

[4] Eaton, John W., GNU Octave Manual, Network Theory Limited, 2002.