REDUÇÃO DO TEMPO DE EXECUÇÃO DE MÉTODOS NUMÉRICOS UTILIZANDO GNU/OCTAVE E GPGPU

ANDRÉ LUIS TIBOLA, VINÍCIUS MENEZES DE OLIVEIRA

Resumo

No presente trabalho é apresentada uma proposta para solução do problema da diminiuição do tempo de execução de scripts Octave, de forma transparente ao usuário programador. Nesta utilizamos computação em GPU com placas gráficas NVIDIA e o toolkit CUDA, a diminuição do tempo de execução no Octave é obtida pela modificação de bibliotecas utilizadas pelo mesmo, e por substituição de algumas das suas extensões.

Palavras-Chave: BLAS, Computação de Alto Desempenho, CUDA, GPGPU, GNU/Octave, Nvidia

TRANSPARENTLY OPTIMIZATING GNU/OCTAVE WITH GPGPU

Abstract

This paper present an alternative to optimize GNU/Octave Numerical Methods, in a transparent way to the programer user. For this we used GPU computing with a NVIDIA's graphics card through the CUDA toolkit, overlapped the BLAS library and also few OCT extensions.

Key words: BLAS, CUDA, High Performance Computing, GPGPU, GNU/Octave, NVIDIA

1 INTRODUÇÃO

Mesmo com a rápida evolução no desempenho dos processadores convencionais, emprega-se na computação de alto desempenho, arquiteturas alternativas com melhor relação preço por *Gigaflop*, como é o caso mais recentemente do *Cell* e das *GPU*. A utilização de *hardware* dedicado para processamento gráfico é impulsionada pela proeminente demanda computacional das aplicações do gênero, em especial, na indústria do entreterimento. Segundo dados apresentados pela fabricante *NVIDIA* em [1], o aumento observado no poder computacional das GPU, quando comparado as CPU convencionais, é notoriamente maior nos últimos anos. Dado o baixo custo desse tipo de equipamento, logo surgiu o interesse na aplicação do mesmo para computação de propósito geral.

As GPU atuais são compostas por aglomerados de processadores, sendo geralmente integradas numa placa de expansão com memória dedicada de alta velocidade. Podem ser enquadradas pela taxonomia de Flynn (Flynn [2]) como processadores MIMD (*Multiple Instruction Multiple Data*) no entanto, essa fabricante (NVIDIA) as classifica como *Single Instruction Multiple Thread* (SIMT)[1], pois podem expressar paralelismo em nível de *thread*, mas alcança maior performance quando todas *threads* executam o mesmo código num dado instante no tempo. Essa estreita ligação com o processamento paralelo expõe a natureza da tarefa que desempenha, onde conjutos de pixels são processados por algoritimos *data-parallel* (Hillis e Steel [3] discorrem detalhes sobre os algoritimos data-parallel), são esses algoritmos que podem ser executados com melhores resultados em GPU.

Após a adoção da arquitetura de shaders unificados em VGAs, a fabricante NVIDIA lançou um *toolkit* para programação de GPUs denominado CUDA (*Compute Unified Device Architeture*). O advento deste representa uma mudança no modo de programação das placas gráficas, pois anteriormente era necessário o mapeamento do problema para alguma API (*Application Programming Interface*) gráfica, enquanto agora é possível programá-las com código ”regular”, de forma tão conveniente quanto outras tecnologias para HPC.

Uma característica importante das GPU é o grande poder para executar operações de ponto flutuante, o que desperta o interesse em diversas áreas, inclusive básicas como a Álgebra Linear Computacional. Dentro desta área um software amplamente utilizado é o *GNU/Octave* (ou simplesmemnte Octave).

O Octave é um software livre majoritariamente compatível com o *Mathworks Matlab*®, de modo geral, este software interpreta uma linguagem matricial própria e possui diversas rotinas já implementadas para uso em variadas áreas, a disposição em seu ambiente, o que faz ele lograr muitos usuários das Ciências e Engenharias. O Octave pode ser estendido a fim de tornar disponíveis mais funcionalidades em seu ambiente, podem ser desenvolvidas extensões utilizando os *scripts* próprios da linguagem, através de uma sofisticada API e classes C++, e ainda através de arquivos MEX (API em C utilizada pelo Matlab), como mostra Eaton em [4].

Sendo cada vez mais frequente a utilização desse aplicativo em computadores convencionais (desktops e notebooks), é especialmente atrativo buscar soluções que explorem essas capacidades no intuito de beneficiar os usuários de Octave provendo melhoras no desempenho.

2 MOTIVAÇÃO

Somos constantemente inclidados a utilizar tecnologias imaturas, no entanto, muitas inscursões malfadadas acabam por penalizar os adeptos quando da descontinidade destas tecnologias. Devido essas circunstâncias os usuários tendem a resistir quando se trata da utilização de novas ferramentas. Podemos facilmente reconhecer a necessidade de soluções que permitam a boa utilização da tecnologia sem incorrer em grandes riscos aos adeptos, é com essa motivação que desenvolvemos este trabalho a fim de acelerar de forma transparente ao usuário *scripts* de Octave.

Uma das características das linguagens interpretadas, é a habilidade para desenvolvimento de aplicações, de maneira ágil, e como diferencial do Octave o grande ferramental já disponível o torna excelente para aplicações numéricas. Algumas das operações do Octave são executadas externamente utilizando bibliotecas como BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra PACKage) e FFTW (Fastest Fourier Transform in the West); como consequência em aplicações com grandes volumes de dados por operação (e que utilizem dessas biliotecas), observaremos pouca diferença no tempo total de execução quando comparado com implementações utilizando linguagens compiladas. Dessa forma o Octave mostra-se uma poderosa ferramenta para o tratamento numérico de dados, mesmo em grandes volumes.

3 METODOLOGIA

Mais precisamente, este trabalho consiste em otimizar chamadas com características específicas realizadas pelo Octave durante a execução de scripts. Para isso podemos utilizar fundamentalmente três técnicas: estender os comandos disponíveis no ambiente do Octave (citado anteriormente); modificar o Octave; ou alterar as bibliotecas que ele utiliza.

O maior problema em utilizar extensões do Octave é que não podemos com elas sobrepor operações da linguagem (ja que não é uma mudança transparente ao Octave) como, por exemplo, o operador de multiplicação ”*”sendo necessário uma nova função ”multiplicar(x,y)”para prover essa funcionalidade, disso decorrem dois novos problemas: são necessárias alterações nos scripts existentes para utilizar o novo recurso; e outras funções que utilizem multiplicação (como a inversão de matrizes) não se valeriam dessa otimização.

Modificações no Octave são potencialmente as com maior ganho, pois podem gerenciar a memória em CPU e em GPU e eliminam a necessidade de modificações dos scripts existentes, entretanto é bastante custoso manter o *fork* atualizado em relação ao *branch* principal e mais difícil ainda ter essas modificações incluídas oficialmente. Além disso, os ganhos principais estão em fazer a gerência das transferências de memória entre CPU e GPU, sendo que os ganhos em cenários mais gerais, provavelmente, não são expressivos a ponto de justificar tais alterações.

A modificação nas bibliotecas utilizadas pelo Octave, tem como maior problema a necessidade de copiar-se os dados da memória principal para a memória de vídeo a cada chamada da biblioteca, pois não é possível assim gerenciar a memória operada pelo Octave, o que no caso de não haver modificações nos dados torna-se uma penalidade considerável. Além disso, torna-se oneroso customizar (e manter) diversas bibliotecas para modificar apenas algumas rotinas.

Em nossa proposta optamos por uma abordagem híbrida de modificações nas bibliotecas do Octave e extensões dele. As bibliotecas são escolhidas pela possibilidade de reutilização das mesmas em outros aplicativos e por serem um método totalmente transparente ao usuário para otimizar as aplicações já existentes. As extensões são utilizadas para as funções que são normalmente (i.e. na distribuição padrão do Octave) implementadas em arquivos externos, como o caso das Transformadas de Fourier que são implementadas em uma extensão do tipo ”oct”.

Outra característica da otimização em nível de biblioteca é que ao selecionar um conjunto mais próximo da base na hierarquia das mesmas (ou seja, bibliotecas com rotinas de base), poderemos ter bons ganhos em cenários mais gerais já que serão utilizadas tanto pelo próprio Octave como por outras bibliotecas utilizadas por ele, neste caso, a BLAS é utilizada tanto pelo Octave quanto pelo LAPACK. Poderemos assim ter ganhos mais modestos em cenários mais gerais providos pela otimização das bibliotecas e ganhos mais expressivos para casos mais específicos otimizando as rotinas chave através de extensões.

Especificamente neste trabalho visamos à utilização de GPGPU em placas gráficas da NVIDIA com o toolkit disponibilizado pela mesma, o CUDA. O toolkit CUDA é composto por compiladores e bibliotecas, sendo duas voltadas para aplicações específicas: a CUBLAS e a CUFFT que são respectivamente uma implementação parcial da BLAS em espaço de GPU e a implementação de Transformadas de Fourier também em espaço de GPU. Para a arquitetura proposta realizamos a otimização pela substituição de rotinas da BLAS utilizada em CPU por versões com a habilidade de fazer as transferências necessárias para a GPU e executar versões em GPU dessas rotinas; também otimizamos as Transformadas de Fourier utilizando extensões do tipo ”oct”valendo-se da mesma abordagem.

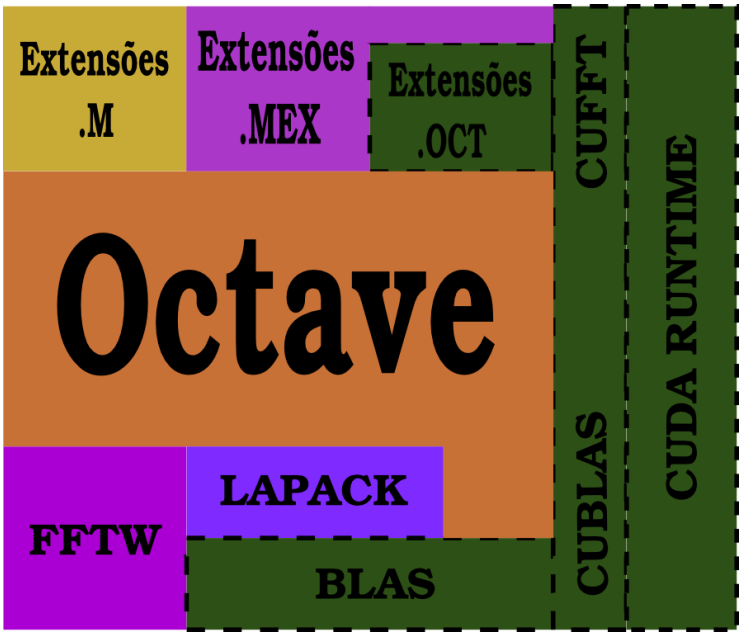

Figura 1: Estrutura do Octave após modificações

Na figura 1 apresentamos a organização do Octave após nossas alterações. Nela

explicitamos as extensões , que rodam no topo do Octave, e a possibilidade das extensões binárias utilizarem GPGPU (as Transformadas de Fourier estão inseridas nas ".oct"). Observamos ainda, bibliotecas pertinentes da base do Octave e destacamos a substituição da BLAS por nossa implementação. Assim, as modificações nas bibliotecas são transparentes ao Octave - como consequência também às extensões - e as alterações como um todo, transparentes ao usuário. Figuramos dessa forma uma implementação que atende aos critérios estabelecidos anteriormente.

Devemos também considerar nas arquiteturas atuais a possibilidade de executar computação apenas em GPU ou simultâneamente em CPU e GPU. A computação híbrida é a forma com maior aproveitamento dos recursos, entretanto exige um delicado controle dos mesmos, demandam mais tempo de desenvolvimento e tem maiores complicações para depuração (acarretando em maiores custos de software); a grande vantagem dessa abordagem é em pequenas instâncias e em rotinas com menor número de cálculos, entretanto os ganhos mais expressivos em cenários com maior carga de cálculos é provido pela GPU. Tomov e Dongarra demonstram em [5] resultados da aplicação de computação híbrida para álgebra linear computacional em sistemas atuais.

4 ANÁLISE DOS RESULTADOS E DESDOBRAMENTOS DO TRABALHO

Os resultados apresentados são tomados diretamente no Octave com o propósito de agregar todos os overheads envolvidos nas operações, tendo um cenário mais próximo da aplicação. Expomos uma comparação da implementação BLAS, as BLAS utilizadas são a reference, a ATLAS (Auto Tuned Linear Algebra System) que é uma BLAS para CPU altamente otimizada, e nossa implementação em GPU.

O cenário utilizado para os testes foi composto por processador AMD Athlon 64 X2 3800+ (SSE3) e VGA XFX Geforce 8800 GS. A figura 2 demonstra o speedup (relativo) observado quando utilizada a operação de multiplicação de matrizes no Octave ("*"), o qual expressa a relação do número de vezes que uma implementação é mais rápida que outra, dado pela da equação 1.

$$S_p = \frac{T_s}{T_p} \tag{1}$$

A base para a comparação é a implementação *reference* compilada com GCC e nível de otimização -O2". A versão utilizada do Octave foi a 3.0.3 e a de ATLAS foi 3.9.3 com suporte a todas instruções SIMD ativadas.

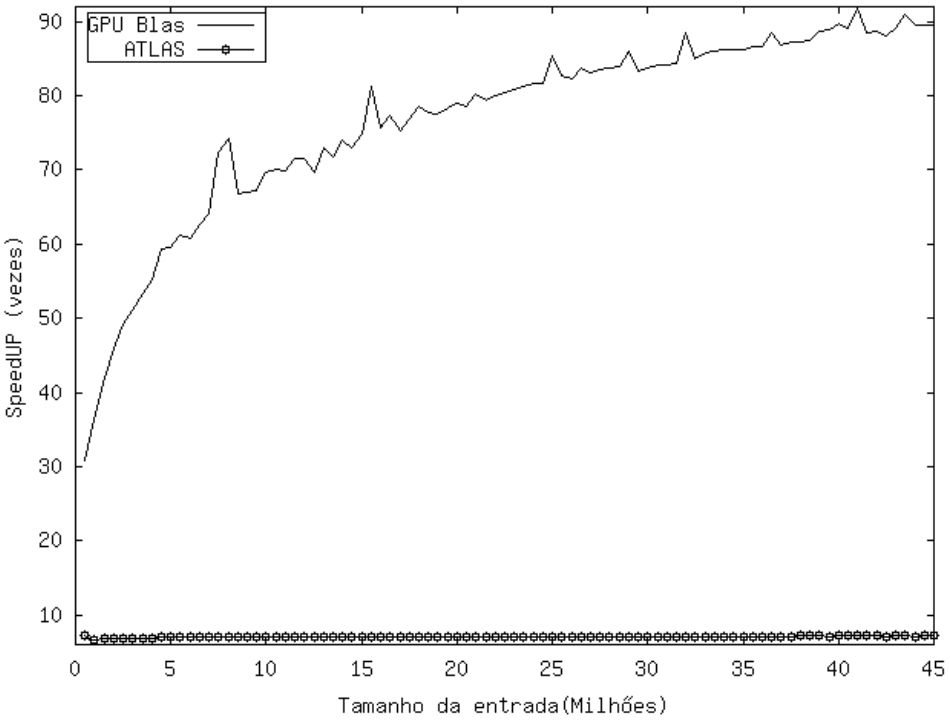

Figura 2: SpeedUP Observado no operador multiplicação

Devemos observar que a multiplicação de matrizes vale-se da operação GEMM da BLAS, apresentada na equação 2 (Sendo A,B e C matrizes e α e β escalares):

$$C = \alpha * A * B + \beta * C \tag{2}$$

Podemos notar pela Equação 2 que a multiplicação de matrizes (α=1, β=0) subutiliza as capacidades da operação, entretanto ela (a GEMM) é o principal bloco de construção e o primeiro ponto a ser focado quando deseja-se uma BLAS com alto desempenho. Temos dessa forma ela como referência, e devido aos overheads envolvidos o speedup atingido na GEMM é possívelmente o máximo speedup alcançável.

A variação no speedup com o crescimento da instância deve-se ao atenuamento dos overheads da implementação. Como observamos pouca degradação de performance, podemos evidenciar a previsibilidade do processo a fim de reforçar a confiabilidade para aplicação da arquitetura mostrada.

A continuidade do trabalho foca consolidar a BLAS proposta e realizar avaliações sistemáticas do desempenho da mesma quando utilizada indiretamente por funções do LAPACK. Como aparato vislumbra-se o auto-tunning da biblioteca afim de utilizar de forma mixada GPU e CPU no intuito de contornar instâncias pequenas nas quais os overheads tornam mais lenta a execução em GPU e nas grandes onde a memória gráfica disponível não é capaz de comportar a computação.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] NVIDIA, NVDIA CUDA Programming Guide, 2009

[2] M. Flynn, Some Computer Organizations and Their Effectiveness, IEEE Trans. Comput., 1792, Volume C-21, 948p.

[3] Daniel W. Hillis, Guy L. Steele, Data Parallel Algorithms, 1986, Communications of the ACM, Volume 29, 1170-1183p.

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

[5] S. Tomov , J. Dongarra, M. Baboulin, Towards Dense Linear Algebra for Hybrid GPU Accelerated Manycore Systems, 2008, University of Tennessee, Computer Science Technical Report, Note 210 UT-CS-08-632

5 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 Luis Tibola.