Um estudo de implementação paralela da Eliminação de Gauss usando OpenMP e CUDA

Vitor Milanez[1], Ivan Navas[1], João Migliatti[1], Matheus Miata[1], Hélio Guardia

[1]Departamento de Computação
Universidade Federal de São Carlos (UFSCar) – São Carlos, SP – Brazil

{vmilanez,ivancn,joaomigliatti,matheusymm}@estudante.ufscar.br

Abstract. *With the increase in the degree of parallelism in computer systems, interest in parallel implementations to reduce execution times for the most varied problems also increases. Knowing this, in this article we present two parallel versions for the Gaussian Elimination problem, one in a multi-core environment and the other using a GPU. The results obtained show parallelism is advantageous in this case, with greater gains obtained with GPU.*

Resumo. *Com o aumento do grau de paralelismo nos sistemas computacionais, aumenta-se também o interesse por implementações paralelas para reduzir os tempos de execução dos mais variados problemas. Sabendo disso, neste artigo foram feitas duas versões paralelas para o problema da Eliminação Gaussiana, uma em ambiente multi-core e outra em GPU. Os resultados obtidos mostram que o paralelismo é vantajoso neste caso, com maiores ganhos obtidos com GPU.*

1.Introdução

Sistemas lineares estão entre os mais importantes problemas matemáticos encontrados em aplicações científicas e industriais, sendo o método da eliminação de Gauss uma das mais populares formas para resolvê-los. O sistema é resolvido com sucessivas operações elementares sobre as linhas do sistema, "eliminando" todos os números que ficam nas colunas abaixo da diagonal principal ao zerar estes, o que transforma o sistema original em um de mais fácil resolução. Isso é explicado em detalhes em [Grcar 2011].

Sabendo da importância deste algoritmo, é natural que já existam inúmeras versões de código para automatizar este problema. Dada a complexidade exponencial para sua realização sequencial, contudo, torna-se interessante explorar estratégias para sua execução com menor tempo de execução. Tomando como base o código sequencial em C da *Gauss Elimination*, apresentado na maratona de programação paralela [IEEE, 2010], este artigo apresenta os resultados de um estudo para a paralelização do código com duas estratégias: usando OpenMP em ambiente multiprocessado com memória compartilhada, e usando CUDA para programação em GPU.

2. Aspectos gerais do problema

Para efeitos de sua resolução, um sistema de equações lineares pode ser representado como um conjunto de matrizes em que Ax=B.

O objetivo de resolver esse sistema de equações é encontrar valores para as incógnitas de X, dados os valores de A e de B. Para sua resolução, o algoritmo de Eliminação de Gauss utiliza a característica das equações lineares de que qualquer linha pode ser substituída pela mesma linha somada a outra linha multiplicada por uma

constante, conseguindo assim, zerar todos os elementos de uma coluna abaixo de uma linha de referência com a seguinte propriedade:

$$A_{j,i} = A_{j,i} + A_{i,i} (-A_{j,i}/A_{i,i}) = 0 \quad \text{(Eq 1)}$$

Para minimizar problemas com decimais, esse procedimento é modificado para o chamado pivoteamento parcial, em que uma linha é substituída pela pela linha abaixo dela que possui o maior elemento absoluto na coluna que será usada na eliminação.

2.1. Trabalhos relacionados

Dada a relevância da resolução de sistemas de equações, vários trabalhos já foram realizados procurando paralelizá-lo. Em [McGinn and Shaw 2002], foram apresentadas duas soluções de implementação paralelas para a eliminação de Gauss, a primeira utilizando um ambiente com memória compartilhada, com OpenMP. Nesta, as iterações foram distribuídas de forma *dynamic* e *static* e depois comparadas, chegando à conclusão que a distribuição *dynamic* funciona melhor por conta de sua habilidade de distribuir novas iterações enquanto outras *threads* continuam ocupadas. Neste estudo, também foi utilizada a distribuição dinâmica.

Já em [Facci and Gonçalves 2011], o objetivo foi implementar e analisar algoritmos paralelos que resolvem sistemas de equações lineares. Foram feitos 2 experimentos, com os algoritmos de eliminação de Gauss e relaxamento Gauss-Seidel. A estratégia de paralelização utilizada para a eliminação de Gauss foi a de obter os dados do sistema de equações (coeficientes e resultados) e os alocar em uma matriz, para depois conseguir o número de *threads* e as criar. Cada *thread* se responsabiliza por um subconjunto das equações a processar, associado ao seu número de identificação. Ao começar uma iteração, a *thread* verifica se a equação pivô que será usada está pronta, para isso verificando o valor da variável barreira; caso contrário, a equação ainda não foi processada ao ponto de poder ser usada como pivô, fazendo com que esta *thread* espere na barreira até que a *thread* responsável por aquela equação termine de processá- la.

3. Metodologia

O trecho em que será trabalhada a paralelização do algoritmo de eliminação de Gauss utilizado como referência é apresentado no Algoritmo 1, com a variável ***a*** sendo a matriz de coeficientes, ***b*** o vetor de resultados da equação, ***n*** o tamanho da matriz e ***m*** a constante que será calculada para zerar as linhas.

Algoritmo 1 *Gauss elimination*

```
1: for i = 0 to n - 1 do
2:    for j = i + 1 to n do
3:       m ← a[ j ][ i ]/ a[ i ][ i ]
4:       for k = i to n do
5:          a[ j ][ k ] ← a[ j ][ k ] -  a[ i ][ k ] * m
6:       end for
7:    b[ j ] ← b[ j ] -  b[ i ] * m
8:    end for
9: end for
```

Versão em OpenMP[1]:A estratégia de paralelização utilizada com *OpenMP* foi a distribuição do cálculo de cada linha da matriz de coeficientes e de cada posição do resultado das equações do sistema linear para uma *thread* do time. Nesse sentido, foi atribuída uma forma de distribuição das iterações do tipo *dynamic*, uma vez que a carga de trabalho em cada linha abaixo da linha atual pode variar em função dos dados presentes. Considerando as dependências de dados, o paralelismo adotado neste caso consiste em utilizar múltiplas *threads* no tratamento de cada coluna, zerando todos os índices nesta posição em todas as linhas abaixo da linha atual. Para tanto, cria-se um time de *threads* exclusivamente para dividir as iterações do segundo *for* no Algoritmo 1, entre as linhas 1 e 2, como apresentado no trecho de código a seguir. Vale ressaltar a necessidade de que as variáveis ***k*** e ***m*** sejam privadas para cada *thread,* para evitar manipulações indesejadas que podem resultar em erros de cálculo.

#pragma omp parallel for private *(k, m) schedule(dynamic)*

Versão em CUDA[1]:A estratégia de paralelização com *CUDA* foi semelhante à escolhida com *OpenMP*. Com isso, o trecho de código responsável por esse cálculo foi adaptado para ser compatível com o ambiente de *GPU* de modo que a matriz de coeficientes e o vetor de resultados foram alocados em espaço de memória da *GPU* e foi definido um *grid* e *block* de *threads 1D*. O trecho referente à paralelização, à função do *kernel* e sua invocação, é apresentado no Algoritmo 2, com as variáveis ***d_a*** e ***d_b*** servindo para alocar ***a*** e ***b***, respectivamente, no espaço de memória da GPU.

Devido à dependência de dados nos ajustes dos valores das colunas, o paralelismo possível está associado ao número de linhas restantes abaixo da linha atual. Caberá a cada *thread* usar um valor linearizado do seu índice lógico (composto pela identificação do bloco e da *thread* neste bloco) para determinar sua linha de atuação. Outro aspecto importante é manter os dados manipulados na memória da GPU.

Algoritmo 2 Eliminação de Gauss em CUDA

```
1: __global__ void gElim(double* a, double* b, int n, int i) { // Função em kernel
2:    row ← (i+1) + blockIdx.x * blockDim.x + threadIdx.x;
3:    if (row < n ) then
      … //  Mesmo que o Algoritmo 1, utilizando row para calcular a posição do vetor
4:   for i = 0 to n - 1 do    // Invocação da função kernel na função main
5:       gElim <<< grid, block >>> (d_a, d_b, n, i);    // block = 64; grid = n / block
         …
6: end for
```

4. Resultados e discussões

Testes foram realizados para obter o tempo de execução das versões paralelas e sequencial, com uma matriz quadrada de tamanho 4000. Um computador com 32 CPUs

[1] Disponível em: https://github.com/VMila/GaussElimination.git

foi usado para testar a versão *multi-core*. Já a versão em GPU, foi testada em ambiente de *Google Collab*, usando uma GPU *Tesla* T4.

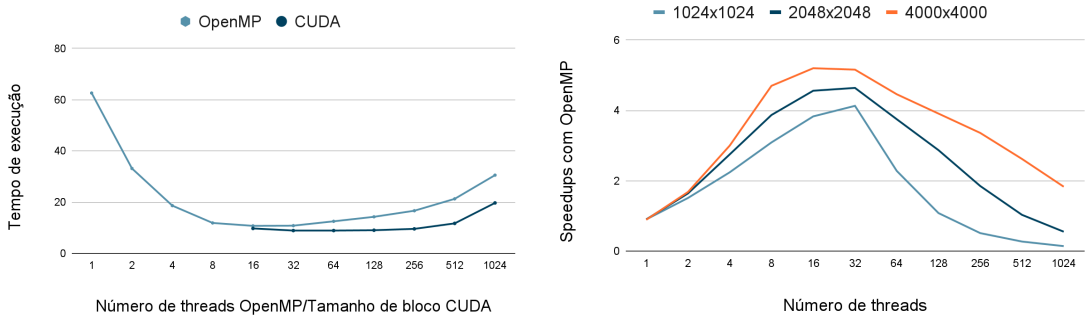

Figura 1. Tempos de execução e *Speedup* com OpenMP

A Figura 1, à esquerda, mostra a comparação de tempo em segundos de cada versão, sendo que 10 execuções foram feitas para chegar a média de tempo de cada tamanho diferente. É possível ver que, na versão em OpenMP, o número de *threads* que obteve melhor resultado foi com 16. Enquanto isso, na versão em CUDA o tempo de execução foi mínimo quando o tamanho de bloco era igual a 32. A Figura 1, à direita, mostra os *speedups* obtidos para os diferentes números de *threads* com OpenMP. A Tabela 1 mostra as acelerações (*speedups*) com CUDA, usando blocos com 32 *threads*.

Tabela 1: *Speedup* obtido com CUDA, usando 32 *threads* por bloco.

Speedup / Tamanho	1024 x 1024	2048 x 2048	4000 x 4000
CUDA	4.49	7.84	6.27

5. Conclusões

Este trabalho investigou a paralelização do algoritmo de eliminação de Gauss para a resolução de um sistema de equações lineares. Utilizando dois ambientes diferentes, multiprocessador com memória compartilhada e GPU, foi possível obter ganhos de desempenho com o paralelismo. Os testes mostraram que ambas as implementações paralelas são vantajosas quando comparadas à sequencial, mas a execução em CUDA demonstrou melhor desempenho. Quando comparados aos trabalhos de [McGinn and Shaw 2002] e [Facci and Gonçalves 2011], percebe-se que os resultados obtidos foram melhores, já que nesses trabalhos, os *speedups* relatados foram todos inferiores a 5.

6. Referências

S.F.MCGINN; R.E.SHAW. Parallel Gaussian Elimination Using OpenMP and MPI. 16Th Annual International Symposium On High Performance Computing Systems And Applications. Saint John, p. 1-5. jun. 2002.

FACCI, Henrique B.; GONÇALVES, Ronaldo A. de L. EXECUÇÃO DE SISTEMAS. DE EQUAÇÕES LINEARES EM PROCESSADORES MULTI-CORE. IV Epac: Encontro Paranaense de Computação. Maringá, p. 1-9. maio 2011.

GRCAR, Joseph F.. Mathematicians of Gaussian Elimination. Notices Of The Ams. Providence, Ri, p. 782-792. jul. 2011.

IEEE (org.). 5th Marathon of Parallel Programming. **SBAC-PAD'2010.** Petrópolis, Rio de Janeiro, p. 5-6. 28 out. 2010.