Conversão de algoritmos de processamento e análise de dados atmosféricos da linguagem C para CUDA

Bruno N. Zanette

Universidade Federal do Paraná, Dept. de Informática, Brasil

Resumo

Este artigo apresenta o processo de conversão de um algoritmo de processamento de dados atmosféricos, onde cada série é processada de forma independente uma da outra, muito utilizados em laboratórios de física e meteorologia por todo o mundo, da linguagem C para CUDA, com o intuito de mostrar que a relação custo e benefício dessa tarefa pode ser muito boa. Como exemplo usaremos a conversão do algoritmo "Filtro de Lanczos".

Palavras-chave: filtro de lanczos, processamento e análise de dados atmosféricos

Contato dos Autores:
{brunonzanette}@gmail.com

1. Introdução

O algoritmo do "Filtro de Lanczos" descrito por [Duchon 1979] é utilizado, entre outras coisas, para filtrar variações temporais de dados atmosféricos diários. O problema consiste em calcular resultados para um certo vetor de dados, multiplicando-se um vetor de constantes de tamanho menor, WT, previamente calculado, com um pedaço da serie de dados, e armazenando o resultado numa nova série de dados, na posição central, como descrito na fórmula a seguir:

$$Res\left[T+\left|\left(K/2\right)\right|\right]=\sum_{i=0}^{K-1} WT\left[i\right]*Dados\left[T+i\right]$$

Para cada resultado produzido por esse algoritmo é necessário K multiplicações, onde K é o número de pesos usados, definidos [Duchon 1979]. Sabendo-se que cada um desses resultados é relativo a um único valor numa série de dados atmosféricos, para um certo dado contendo NX pontos no eixo X (longitude) e NY no eixo Y (latitude), ou seja NP=NX*NY quadrículas, o número aproximado de multiplicações é de NP*(365)*K para o cálculo de apenas 1 ano de dados. Por isso, mesmo sendo um algoritmo simples, o tempo de execução desse algoritmo sequencial é muito grande. Uma característica comum em algoritmos deste gênero.

Porém, dado ao fato de que o processo de filtragem de cada quadrícula é independente uma da outra, esse algoritmo é altamente paralelizável. E por possuir um número elevado de quadrículas, a plataforma CUDA acaba sendo muito eficiente nesse caso por disponibilizar uma grande quantidade de núcleos de processamento.

2. Implementação

Nesse capítulo iremos analisar as principais etapas da implementação de algoritmos de processamento e análise de dados atmosféricos.

2.1 Estruturas de dados

O primeiro passo para a implementação de qualquer algoritmo é criar uma estrutura de dados que se adeque ao problema, e para este caso não é diferente. Entretanto neste caso específico isso se torna um desafio ainda maior por dois motivos: o tamanho total do arquivo de dados e sua organização original.

Muitos dos dados atmosféricos usados em pesquisas hoje em dia possuem uma enorme quantidade de quadrículas. A maioria desses dados possuem grades que variam de 128x64 (Global, com quadrículas de 2,5°) a até mesmo 360x180 (Global, com quadrículas de 1°). Considerando que esses dados podem ser de valores diários, para 10 anos de dados numa grade de 360x180 quadrículas, por exemplo, o tamanho total do arquivo de entrada seria de aproximadamente 900MB. Na figura 1 essa relação entre tamanho total da série de dados com o tamanho total do arquivo de dados.

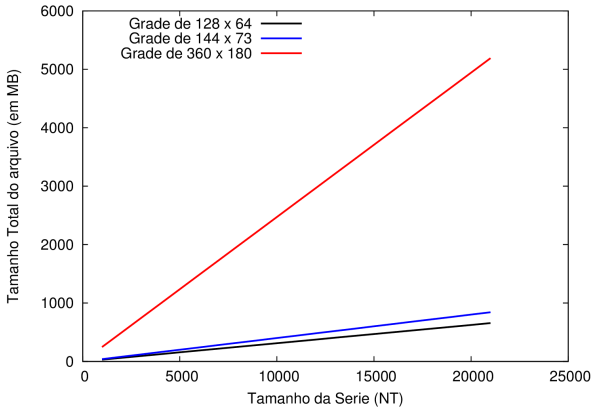

Figura 1: Gráfico contendo a relação entre o tamanho da série de dados e o tamanho total dos arquivos

Por um lado isso é bom, pois permite uma melhor visualização do mesmo e resultados mais precisos. Porém, isso dificulta a sua utilização na plataforma CUDA que possui uma quantidade limitada de

memória, principalmente quando lembramos que ainda é necessário reservar um espaço em memória para armazenar os resultados.

Uma solução para possibilitar o uso desses dados em qualquer plataforma CUDA, independente do tanto de memória disponível, é processar esses dados em ciclos de processamento. Porém os mesmos são originalmente oranizados de forma que os valores dos eixos X e Y fiquem juntos, fazendo com que os valores de uma quadrícula no eixo do tempo fiquem separados, o oposto do que seria o ideal. Isso não é nenhum obstáculo quando pensamos numa execução sequencial do algoritmo onde a quantidade e facilidade de acesso à memória é maior, mas impossibilita a ideia de ciclos de processamentos, pelo fato dos valores estarem separados no eixo do tempo, o que impede que os mesmos sejam repartidos.

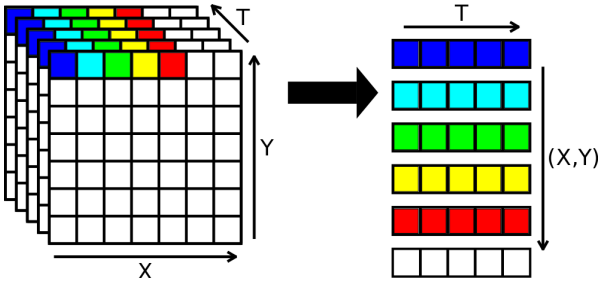

Figura 2: Organização dos dados

Para contornar isso foi utilizado uma função de leitura que armazena os dados na memória do Host de forma que a série de tempo de cada quadrícula seja contínua (figura 2), assim sendo possível copiar pedaços separados desse dado para a GPU. Além disso, ao invés da estrutura de matriz foi usado a estrutura de um vetor comum para armazenar esses valores e funções auxiliares para calcular a posição na memória em que se inicia os dados de cada quadrícula. Essa decisão ainda facilita a cópia de dados entre a memória do Host e do Device.

2.2 Conversão do código

Em teoria essa tarefa seria a mais difícil do processo de implementação do algoritmo em CUDA, porém foi o que menos precisou de ajustes.

Na função que implementa o algoritmo “Filtro de Lanczos” as únicas alterações necessárias foram remover o loop de incremento das quadrículas a serem processadas, necessário na versão sequencial, e adicionar o cálculo que defini qual quadrícula cada um dos núcleos de processamento da GPU será responsável por processar.

```
__global__ void filtro_lanczos( <PARAMETROS > ){

    <DECLARACAO_DE_VARIAVEIS>

    int p=(blockDim.x*blockIdx.x)+threadIdx.x;
    if (p >= total_pos) return;

    for (p=0;p<total_pos;p++){
        ini_seq=(p*NT);
        for (j=fNWL;j<NT-fNWL;j++){
            <CALCULOS>
            s[ini_seq+j]=resultado;
        }
    }
}

//-----------------------------------------------//

int main(int argc, char **argv){

<DECLARACAO_DAS_VARIAVEIS>

<ALOCA OS DADOS DE ENTRADA E SAIDA NO HOST>

<LE OS ARGUMENTOS E OS DADOS DE ENTRADA>

<CALCULA O VETOR "WT">

<CALCULA OS PARAMETROS DE EXECUCAO DO CUDA>

//ALOCA OS DADOS DE ENTRADA E SAIDA NO DEVICE
cudaMalloc((void**)&d_entrada, tam_por_ciclo);
cudaMalloc((void**)&d_saida, tam_por_ciclo);

//ALOCA E COPIA O VETOR "WT" DO HOST PARA O DEVICE
cudaMalloc((void**)&d_wt, (NWT*sizeof(float)));
cudaMemcpy(d_wt, h_wt, (NWT*sizeof(float)),
        cudaMemcpyHostToDevice);

for (ciclo=0;ciclo<total_ciclos;ciclo++){

    pos_entrada=(ciclo*npos_por_ciclo)*NT;

    cudaMemcpy(d_entrada,(h_entrada+pos_entrada),
        tam_por_ciclo,cudaMemcpyHostToDevice);

    filtro_lanczos <<< … >>> ( … );

    pos_saida=(ciclo*npos_por_ciclo)*NT;

    cudaMemcpy((h_saida+pos_saida),d_saida,
        tam_por_ciclo,cudaMemcpyDeviceToHost);
}

salva_arq_saida(param, h_saida);
desaloca_variaveis();
return 0;
}
```

→ Executado nas duas versões

→ Exclusivo da versão Sequencial

→ Exclusivo da versão CUDA

Figura 3: Diferenças entre o código C e CUDA

Foi a função “Main” do código em C que sofreu as maiores mudanças. Porém, grande parte dessas alterações foram apenas para adicionar as premissas básicas de todo programa, como alocação e inicialização de memória, no contexto do CUDA. As outras modificações foram a adição das funções de cópia de memória entre Host e Device e um loop para controlar os ciclos.

Esses ciclos foram pensados para o caso da memória total necessária para armazenar os dados de entrada e saída for maior do que o total de memória disponível na GPU. Para que esse método funcione, além da organização dos dados previamente descrita, foi criado um pequeno algoritmo para dividir igualmente o processamento entre os vários ciclos. Dentro de cada um desses ciclos as seguinte ações são executadas, em ordem:

1. Cálculo da posição de início da cópia dos dados de entrada;
2. Cópia dos dados de entrada do Host para o Device;
3. Execução do algoritmo “Filtro de Lanczos”;
4. Cálculo da posição de início da cópia dos dados de saída;
5. Cópia dos dados de saída do Device para o Host;

Todas essas ações possuem relações diretas com as ações consideradas praticamente obrigatórias em todas os programas escritos em C. São elas: leitura dos dados de entrada, execução do algoritmo de processamento, escrita dos dados de saída.

Portanto, com exceção das alterações obrigatórias para adaptar o programa à plataforma CUDA, não foi feito nenhuma modificação ou otimização no código que implementa o algoritmo do “Filtro de Lanczos” em si.

3. Resultados

Ao analisarmos as etapas da conversão do algoritmo “Filtro de Lanczos” da linguagem C para CUDA percebemos que grande parte das alterações são apenas a adição de premissas básicas de programas escritos em CUDA, e algumas outras poucas para contornar os possíveis problemas de falta de memória da GPU. Ainda assim, ao analisarmos os resultados obtidos percebemos que não há perda de precisão nos resultados e há uma drástica redução do tempo de execução.

3.1 Autenticação dos resultados

Algo imprescindível em algoritmos de processamento e análise de dados para que possam ser efetivamente usados em pesquisas é a confiabilidade dos resultados obtidos. Nem sempre é possível, apenas observando os resultados, saber se o mesmo esta certo ou errado. Portanto é preciso ter total confiança de que os dados obtidos da execução do algoritmo estão corretos.

Por esse motivo, o pequeno número de alterações feitas no código é tão importante, pois evita que erros sejam cometidos no processo de conversão. Além disso, em todos os testes realizados, os resultados obtidos da versão CUDA foram comparados com os do sequencial, e em todos os casos os mesmos foram idênticos, comprovando que os resultados estão corretos.

3.1 Avaliação do tempo de execução

Contudo, a principal motivação da conversão do algoritmo para CUDA é a redução do tempo de execução e verificar se tal melhora faz valer todo o esforço.

Para comparar o desempenho das implementações em C e CUDA foram feitos diversos testes. Em todos os testes foi utilizado dado com 144x73 quadrículas como entrada e a cada execução o tamanho total da série de dados foi incrementado, variando de 1000 valores por série (40MB de dados) até 22000 (882MB de dados). Nesses testes foram utilizados as seguintes máquinas:

- [CPU] Intel Dual-Core E5200 - 2.5GHz
- [CUDA] 9600GT - 512MB / 900MHz
- [CUDA] GTX480 - 1.5GB / 1.4GHz.

Além disso, na versão CUDA foi utilizado como parâmetro de execução 8 threads por bloco. O número total de blocos e ciclos foram calculados diretamente pelo programa em relação ao tamanho do dado de entrada. Sendo assim, é perceptível que não foi feita nenhuma calibração mais otimizada. Tal decisão foi feita propositalmente, em vista que tais ajustes podem variar de uma máquina para outra, ou até mesmo de uma entrada para outra, e assumindo que tais programas deverão ser executados por pesquisadores e que os mesmos não teriam tempo e/ou conhecimento para fazer tais ajustes.

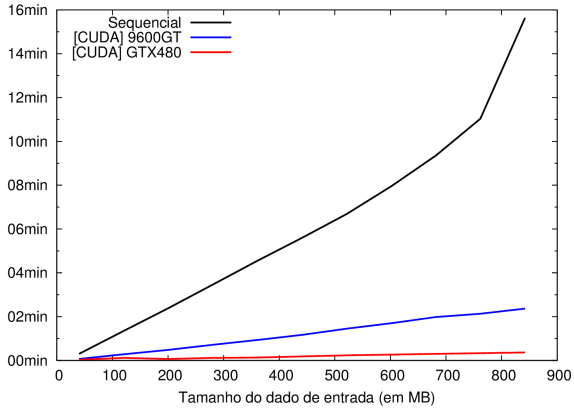

Figura 4: Gráfico comparando o tempo de execução com o tamanho do dado de entrada

Os tempos foram marcados utilizando o comando "time" do Linux, e portanto são referentes a execução completa do programa, e não só apenas da execução do algoritmo de processamento. Os tempos obtidos estão na figura 4.

Como podemos ver, mesmo sem nenhuma otimização tanto no código quanto nos parâmetros de execução do Kernel, os tempos de execução da versão CUDA são significantemente menores, apresentando tempos 6 vezes menores quando usado a GPU 9600GT, e de até 40 vezes com a GPU GTX480 nos testes feitos. Além disso, é perceptível pelas curvas apresentadas no gráfico, que essa redução será ainda maior a medida que o tamanho da entrada aumentar, comprovando o poder de processamento das GPUs Nvidia com a tecnologia CUDA.

3. Conclusão

Podemos concluir com esse trabalho que com apenas algumas modificações, necessárias para incluir as premissas básicas para que o programa execute em GPUs Nvidia capacitadas com a tecnologia CUDA, podemos reduzir drasticamente o tempo de execução de algoritmos de processamento e análise de dados atmosféricos sem abrir mão da confiabilidade dos resultados. Além disso, o trabalho demonstra que tal método de conversão possui um provável potencial de se adaptar a outros algoritmos que possuem características parecidas com o demonstrado, o que facilita ainda mais a conversão destes.

Agradecimentos

Gostaria de primeiramente agradecer toda a minha família pelo suporte em todos os momentos. A Profa. Alice M. Grimm (Dept. De Física da UFPR) por ceder os algoritmos usados como base para esse trabalho, e me ensinar e incentivar a estudar a aplicação de computação em estudos de física e meteorologia. E o Prof. Fabiano Silva (Dept. De Informática da UFPR) por lecionar a matéria de Programação Paralela, a qual me levou a iniciar os meus estudos na área da tecnologia Nvidia CUDA, além de me auxiliar e incentivar a escrever esse artigo. Obrigado!

Referências

DUCHON, C.E., 1979. Lanczos Filtering in One and Two Dimensions