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

Bruno Nocera Zanette
Fabiano Silva

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Introdução e Objetivos do trabalho

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:

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

Para cada resultado produzido por esse algoritmo é necessário K multiplicações, onde K é o número de pesos usados. Sabendo-se que cada um desses resultados é relativo a um único valor numa série de dados atmosféricos, para um certo dado contendo 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. Porém, o processamento de cada quadrícula é independente o que faz com que o algoritmo seja altamente paralelizável.

Essas características são comuns em algoritmos desse gênero e por isso o algortimo Filtro de Lanczos foi usado como base para o trabalho. Porém, o principal objetivo desse estudo é verificar as dificuldades e benefícios da implementação de algoritmos de processamento e análise de dados na plataforma CUDA, especialmente em relação as estruturas de dados usadas, e tentar achar soluções que possam ser utilizadas para a conversão de outros algoritmos, facilitando a conversão dos mesmos e aumentando ainda mais a relação de custo e benefício desse processo.

Conversão do código

```
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;
}

__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;
        }
    }
}
```

Executado nas duas versões

Exclusivo da versão sequencial

Exclusivo da versão CUDA

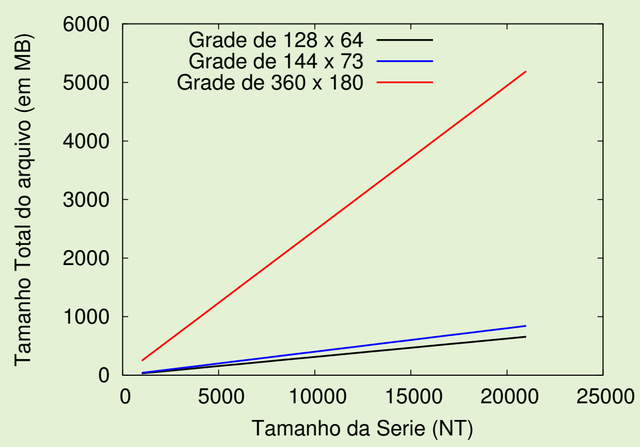
Na função que implementa o algoritmo “Filtro de Lanczos” as únicas alterações necessárias foram remover o loop de incremento das quadrículas a serem processadas, necessário na versão sequencial, e adicionar o cálculo que defini qual quadrícula cada um dos núcleos de processamento da GPU será responsável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

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.

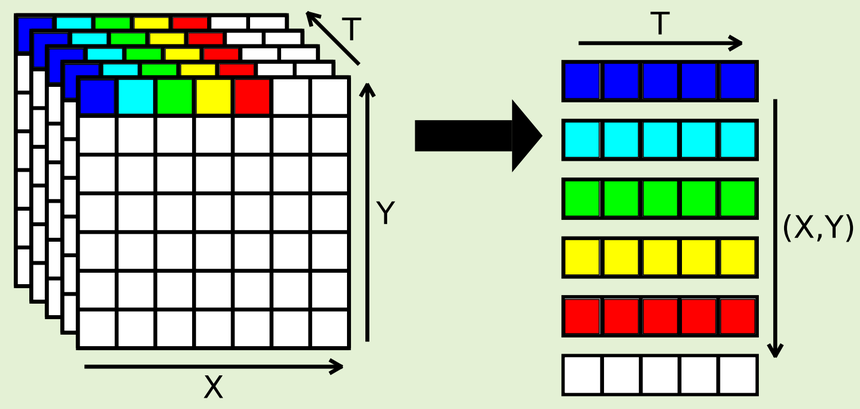
Esses ciclos foram pensados para o caso da memória total necessária para armazenar os dados de entrada e saída for maior do que o total de memória disponível na GPU.

Estrutura de dados

Muitos dos dados atmosféricos usados em pesquisas hoje em dia possuem uma enorme quantidade de quadrículas, para permitir uma melhor visualização do mesmo e resultados mais precisos. Porém, isso faz com que o tamanho dos dados sejam muito grandes, dificultando a sua utilização na plataforma CUDA, que possui uma quantidade limitada de memória.



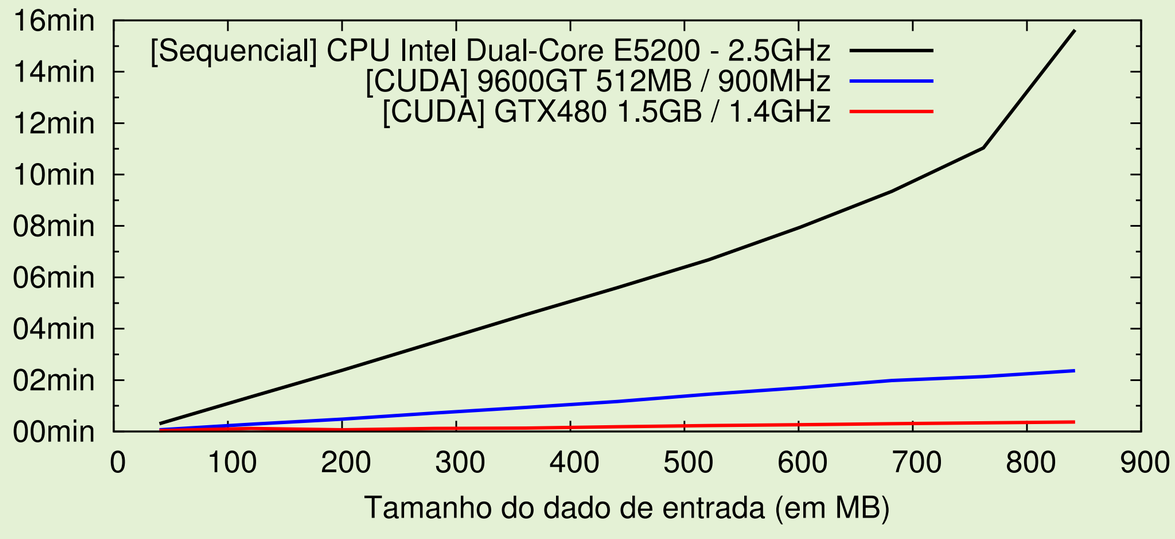
Uma solução para possibilitar o uso desses dados em qualquer plataforma CUDA é processar esses dados em ciclos de processamento, em que cada ciclo seja processado apenas uma fração das quadrículas. Porém esses dados são originalmente organizados de forma que os valores dos eixos X e Y fiquem juntos, fazendo com que os valores de uma quadrícula no eixo do tempo fiquem separados, o que impede que o dado seja repartido. Para contornar isso foi utilizado uma função de leitura que armazena os dados de forma que a série de tempo de cada quadrícula seja contínua assim sendo possível copiar pedaços separados desse dado para a GPU, relativo a apenas as quadriculas que serão processadas em cada ciclo.



Avaliação do desempenho e Conclusões

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).

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.



No gráfico de comparação dos tempos de execução percebemos que utilizando a GPU GTX480 obtivemos uma redução de até 40 vezes do tempo total de exeucução, o que nos leva a concluir que com apenas algumas modificações, necessárias para incluir as premissas básicas para que o programa execute em GPUs Nvidia capacitadas com a tecnologia CUDA, e utilizando uma estrutura de dados adequada, podemos reduzir drasticamente o tempo de execução de algoritmos de processamento e análise de dados atmosféricos sem abrir mão da confiabilidade dos resultados, uma vez que o pequeno número de modificações no código diminui a chance de erros durante a conversão.