Relatório do EP2 de MAC0219

Renan Ryu Kajihara, NUSP: 14605762

19 de novembro de 2024

Resumo

O presente relatório descreve as atividades realizadas no segundo Exercício-Programa da disciplina "Programação Concorrente e Paralela (MAC0219)", que consistiu na paralelização com GPU de um programa que calcula a distribuição de calor em um espaço determinado utilizando CUDA (Compute Unified Device Architecture), verificando as diferenças observadas entre a execução do programa paralelizado e a versão sequencial, principalmente em relação ao tempo de execução.

Conteúdo

1 Introdução

Com o intuito de observar a diferença de tempo de execução entre versões sequenciais e versões paralelizadas com GPU de um mesmo programa, realizou-se a paralelização de um algoritmo que calculava a distribuição de calor em um determinado espaço utilizando CUDA. Além disso, efetuo-se uma série de experimentos para verificar a diferença de performance das duas versões, observando o speedup entre as versões de acordo com os diferentes parâmetros utilizados.

2 Objetivos

O exercício teve como objetivo a implementação de técnicas de paralelização com GPU de códigos na linguagem de programação C, bem como a análise de perfomance entre as versões sequenciais e as versões paralelizadas.

3 Cronograma

O desenvolvimento deste Exercício Programa envolveu algumas etapas-chave, elencadas abaixo:

- (10/11/2024-14/11/2024) - Paralelização do código sequencial, utilizando CUDA.
- (14/11/2024-15/11/2024) - Realização da "Tarefa 3"que consistia em adicionar um corpo com temperatura fixa de 37 graus Celsius no ambiente.
- (15/10/2024-16/11/2024) - Coleta dos dados referentes ao tempo de execução dos programas, com diferentes parâmetros.
- (17/11/2024-18/11/2024) - Análise dos dados e redação do relatório do Exercício Programa.

4 Métodos

4.1 Acesso à um ambiente com GPU

Primeiramente, para conseguir utilizar as ferramentas que permitem a utilização do CUDA, foi necessário instalar o CUDA Toolkit no computador utilizado para depurar e compilar os códigos paralelizados. Além disso, para executar os programas paralelizados, foi utilizado uma máquina RTX3060, que é um GPU desenvolvida pela NVIDIA que conta com 3584 CUDA Cores. O acesso a essa máquina foi feito através da Rede Linux, que é uma rede de computadores presente no Instituto de Matemática e Estatístca da Universidade de São Paulo.

4.2 Tarefa 2 - Implementação em CUDA

A tarefa 2 consistia em paralelizar o programa sequencial utilizado CUDA.

Para realizar a paralelização do programa sequencial utilizando CUDA, ao invés de matrizes bidimensionais, foi utilizado vetores unidimensionais para tratar os pixels da imagem. Nesse sentido, uma posição da matriz h, por exemplo, que é representado como h[i][j], na versão paralelizada é representado como h[i*n+j], com n sendo o número de linhas. Tal mudança foi realizada pois CUDA possui otimizações para vetores unidimensionais, melhorando o desempenho dos cálculos.

Além disso, na versão paralelizada foram utilizados grades e blocos bidimensionais. Uma das dificuldades ao realizar a paralelização foi em relação ao número máximo de threads por bloco. Em

CUDA, cada bloco pode ter no máximo 1024 threads. Dessa forma, quando o programa é executado com mais de 1024 threads por bloco, o programa não produz a saída desejada.

Ademais, para verificar se as imagem produzida pela GPU é igual à produzida pela CPU, foi criado a função "compara_cpu_gpu". É importante observar que, para determinado número de threads, experimentos com um número pequeno de pontos e um número pequeno de iterações produzem saídas diferentes entre a CPU e a GPU, sendo a diferença, na maioria das vezes, presente na primeira ou segunda casa decimal, produzindo imagens praticamente idênticas. Dessa forma, a função "compara_cpu_gpu"considera os pontos iguais quando sua diferença absoluta é menor que 0,5. Tal diferença é praticamente imperceptível na visualização da imagem.

Por fim, para medir o tempo de execução dos processos que eram feitos na GPU e a cópia dos dados do host para o device e vice-versa, foram utilizado os métodos "cudaEventCreate", "cudaEventRecord"e "cudaEventElapsedTime". Tais métodos são disponibilizados pela biblioteca "cuda_runtime".

4.3 Tarefa 3 - Corpo Quente no Quarto

A tarefa 3 consistia em, a partir do código da tarefa 2, adicionar um corpo quente com temperatura fixa de 37 graus no quarto, observando o impacto do corpo na temperatura dos outros pontos do quarto ao passardo tempo.

O corpo foi colocado no meio da imagem. Ele ocupa as posições em que $\frac{n}{2} - \frac{n}{10} \leq i \leq \frac{n}{2} + \frac{n}{10}$ e $\frac{n}{2} - \frac{n}{10} \leq j \leq \frac{n}{2} + \frac{n}{10}$. Dessa forma, a função que inicializa os vetores foi modificada, para que as posições que fazem parte do corpo tenham temperatura fixa de 37 graus. Além disso, a função "jacobi_iteration"também foi modificada, para que a temperatura nas dimensões do corpo não fosse calculada.

5 Análise dos resultados experimentais

5.1 Resultado dos experimentos da Tarefa 2

A tabela que representa os diferentes experimentos feitos com o código da Tarefa 2 para diferentes números de pontos, número de threads por bloco e número de blocos por grade pode ser vista a seguir:

n (número de pontos)	t (número de threads por bloco)	b (número de blocos na grade)	número de iterações	Tempo de execução versão CUDA (s)	Tempo de execução versão sequncial (s)
1000	1024	8	1000	0,48	4,99
1000	32	64	1000	0,47	5,01
1000	1024	64	1000	0,24	5
5000	1024	64	1000	12,76	120,72
5000	32	8	1000	72,63	120,57
5000	32	64	1000	10,34	121,26
5000	1024	512	1000	5,55	122,02
7000	1024	512	1000	12,56	236,42
7000	32	8	1000	142,83	239,62
10000	32	8	1000	294,65	480,07
10000	1024	512	1000	30,36	483,57

Figura 1: Tabela que representa os diferentes experimentos feitos pelo código produzido na Tarefa 2.

Primeiramente, evidencia-se que o código paralelizado com CUDA é muito mais rápido do que o código sequencial. Nesse sentido, observa-se que a versão paralelizada com CUDA em relação a versão

sequencial foi de 10 a 20 vezes mais rápida, dependendo do número de threads por bloco e de blocos por grade.

Além disso, é possível observar que, para a máquina GPU usada (RTX3060), os experimentos com maior número de threads por bloco e maior número de blocos por grade resultaram em um cálculo mais rápido da imagem, acelerando o processo.

5.2 Resultado dos experimentos da Tarefa 3

A tabela que representa os diferentes experimentos feitos com o código da Tarefa 3 para diferentes números de pontos, número de threads por bloco e número de blocos por grade pode ser vista a seguir:

n (número de pontos)	t (número de threads por bloco)	b (número de blocos na grade)	número de iterações	Tempo de execução versão CUDA (s)	Tempo de execução versão sequncial (s)
1000	1024	8	1000	0,46	7,91
1000	32	64	1000	0,46	7,93
1000	1024	64	1000	0,23	7,91
5000	1024	64	1000	14,46	198,18
5000	32	8	1000	70,38	198,24
5000	32	64	1000	10,01	197,93
5000	1024	512	1000	5,18	198,81
7000	1024	512	1000	11,82	389,52
7000	32	8	1000	139,18	390,11
10000	32	8	1000	285,44	794,23
10000	1024	512	1000	28,94	795,93

Figura 2: Tabela que representa os diferentes experimentos feitos pelo código produzido na Tarefa 3.

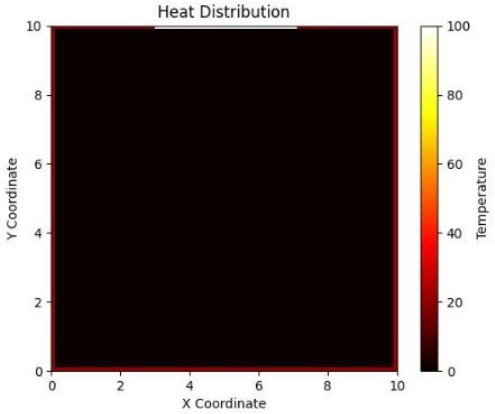
Em primeira análise, é evidente que a versão paralelizada com CUDA sofreu pequenas alterações em seu tempo de processamento, sendo que em alguns experimentos o tempo de execução diminuiu e em outros aumentou.

Nos experimentos que envolvem a versão sequencial, ficou evidente que houbе um aumento de tempo considerável em relação à versão sem a adição do corpo quente fixo. Tal fato pode ter ocorrido porque, apesar da temperatura não ser calculada nas dimensões do corpo (aproximademente 9% da figura), todas as iterações passam por um número maior de comparações para saber se determinada parte da figura deve ter a temperatura calculada, ou não.

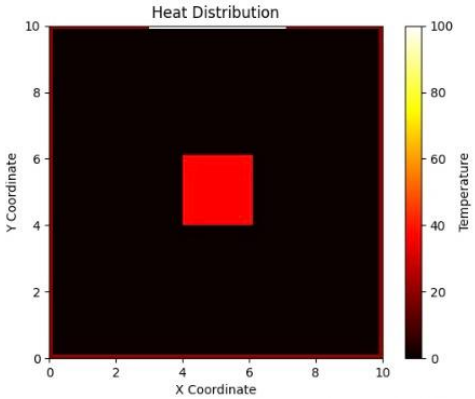
Dessa forma, pelo fato de que a versão sequencial teve um aumento considerável no tempo de processamento e a versão paralelizada com CUDA não sofreu grandes alterações em seu tempo de execução, o speedup aumentou com a adição do corpo fixo. Nesse contexto, a versão paralelizada com CUDA chegou a ficar mais de 30 vezes mais rápida do que a versão sequencial para alguns experimentos.

5.3 Comparação das imagens produzidas com e sem o corpo quente no quarto

A configuração inicial de um quarto com 10000 pixels, com e sem o corpo quente pode ser visto nas seguintes imagens:



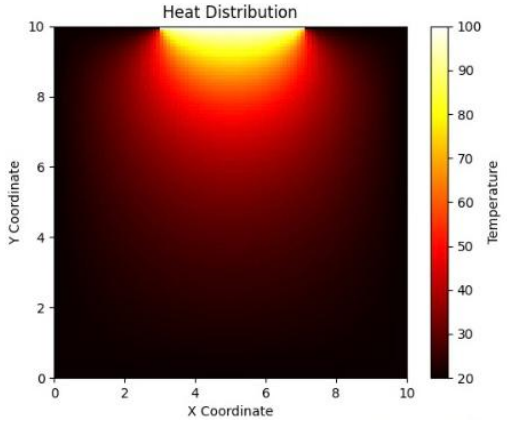
(a) Configuração inicial do quarto sem o corpo quente.



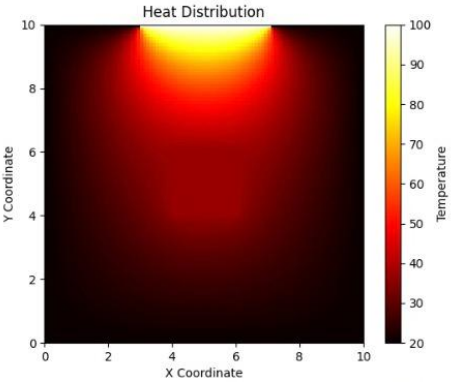
(b) Configuração inicial do quarto com o corpo quente.

Nessas imagens, é possível observar que o corpo quente está exatamente no meio da figura, ocupando aproximadamente 9% do total dela.

A configuração do quarto com e sem o corpo quente após 10000 iterações pode ser visto nas seguintes imagens:



(a) Configuração do quarto sem o corpo quente após 10000 iterações.



(b) Configuração do quarto com o corpo quente após 10000 iterações.

Nessas imagens, é possível observar que o corpo quente está com posição e temperatura fixa. Além disso, é notório que o calor se espalhou mais rápido no quarto em que o corpo quente está presente, uma vez que a temperatura dele, que é maior que a do ambiente, aumenta a temperatura dos locais que estão próximos a ele, a cada iteração.

6 Conclusão

Primeiramente, é notório que a programação com GPUs é muito útil para a programação de alta perfomance, uma vez que elas possuem muitas unidades de processamento e são planejadas para workloads em paralelo. Entretanto, é perceptível que o acesso a uma máquina GPU pode ser difícil

para alguns programadores, já que tais máquinas valem um alto valor financeiro no mercado. Além disso, ao realizar a paralelização do código com CUDA, notou-se que a programação com CUDA pode ser complexa e difícil, uma vez que as bibliotecas possuem uma série de detalhes (como o número de threads por bloco, por exemplo), colocando uma responsabilidade maior no programador, fazendo com que, se a programação não for feita de maneira correta, o programa apresentará uma série de erros. Ademais, vale acentuar que para computações pequenas nãosão indicadas para serem feitas em paralelo em uma GPU, porque a transferência de dados entre a host e o device pode demorar mais do que a própria computação.

Além disso, evidenciou-se que na máquina GPU utilizada, quanto mais threads e blocos forem utilizados, maior o desempenho. Tal fato ocorre porque, ao utilizar um maior número de threads, a computação é distribuída em mais unidades de processamento, resultando em um cálculo mais rápido da matriz.