Atividade 8b - Execução do algoritmo FDTD-2d - SMP, UDP e MPI

Gabriel Thiago Henrique Dos Santos <ra107774@uem.br>

***Abstract*—O estudo de paralelismo tende a ser importante para diversos cenários, como em momentos nos quais há a necessidade e possibilidade de executar o mesmo serviço com diferentes ambientes, no nosso caso é utilizado na execução do algoritmo *fdtd-2d* utilizando as abordagens *SMP*. *UDP* e *MPI*. Para a análise dos experimentos, fora desenvolvido entradas com parâmetro a partir da quantidade de memória na máquina, sendo abordado desde memória física quanto virtual, para todas as abordagens.**

***Index Terms*—Latex template, IEEE, Latin America Transactions, guidelines for authors.**

I. Introdução

EM diversas aplicações, o uso de execuções simultâneas podem ser aplicados, e com isso se tem o objetivo principal de melhorar o desempenho e aproveitamento do hardware [1].

Para este trabalho utilizamos do algoritmo *fdtd-2d* (*Finite Different Time Domain Kernel*), uma abordagem de domínio de tempo de diferença finita, utilizando a técnica 2d, ou seja, é uma técnica de análise numérica usada para modelar eletrodinâmica computacional, utilizado para encontrar soluções aproximadas para o sistema associado de equações diferenciais.

Para isso a implementação fora utilizado como base a disponibilizada pela *PolyBench*, em que é possível definir o tamanho de entradas dos *arrays* em questão utilizadas para multiplicações, realizado alterações necessárias para o melhor funcionamento do teste que serão descritas no decorrer do trabalho. Como auxilio fora utiliza das documentações e exemplos em [2] e os tutoriais de [3] e [4].

O seguinte relatório está organizado por sessões, no qual a II descreve sobre o problema e seus resultados esperados, em seguida é abordado sobre o desenvolvimento e suas devidas explicações. Em IV há a abordagem realizada para em seguida houve a análise sobre os resultados. Ao final, temos a conclusão seguido pelas referencias utilizadas para o desenvolvimento do trabalho.

II. Objetivos

O principal objetivo deste trabalho é ganhar experiência com as abordagens de paralelismo para arquitetura, incluindo exposição a conceitos de alterações de variáveis de ambiente.

III. Descrição do Problema

O Método simplificado de domínio de tempo de diferença finita para dados 2D, que modela campos elétricos e magnéticos com base nas equações de Maxwell. Em particular, a polarização usada aqui é TE^z; Transversal elétrico na direção z. É um estêncil envolvendo três variáveis, Ex, Ey e Hz. Ex e Ey são campos elétricos que variam nos eixos x e y, onde Hz é o campo magnético ao longo do eixo z. Os campos ao longo de outros eixos são zero ou estáticos e não são modelados.

Fig. 1: Equações

$$\mathtt{Hz}^t_{(i,j)} = C_{hzh}\mathtt{Hz}^{t-1}_{(i,j)} + C_{hze}(\mathtt{Ex}^{t-1}_{(i,j+1)} - \mathtt{Ex}^{t-1}_{(i,j)} - \mathtt{Ey}^{t-1}_{(i+1,j)} - \mathtt{Ey}^{t-1}_{(i,j)})$$
$$\mathtt{Ex}^t_{(i,j)} = C_{exe}\mathtt{Ex}^{t-1}_{(i,j)} + C_{exh}(\mathtt{Hz}^t_{(i,j)} - \mathtt{Hz}^t_{(i,j-1)})$$
$$\mathtt{Ey}^t_{(i,j)} = C_{eye}\mathtt{Ey}^{t-1}_{(i,j)} + C_{eyh}(\mathtt{Hz}^t_{(i,j)} - \mathtt{Hz}^t_{(i-1,j)})$$

Em 2 as variáveis Cxxx são coeficientes que podem ser diferentes dependendo da localização dentro do espaço descretizado. No *PolyBench*, ele é simplificado como coeficientes escalares.

IV. Desenvolvimento

Utilizamos da linguagem C para nossa implementação, para o modo sequencial houve a disponibilização da plataforma *PolyBench*, então fora adaptado para o uso do programa UPCC, variando em suas abordagens de uso de paralelismos.

a seguir apresentamos algumas escolhas e estruturas utilizadas no código.

- três tipos de mensagem:

Para a compilação do trabalho:

1) Vá a raiz do projeto e execute:
2) *make* <tipo>;

Temos para o tipo:

- seq: compilação do código sequencial;
- smp: compilação do código com *smp*;
- udp: compiação do código em UDP;
- mpi: compilação do código com *mpi*;
- all: compilação e geração para todos tipos citados acima;
- clean: limpar os arquivos executáveis criado.

A compilação e execução é feita com o uso do programa UPC, sendo para compilar o UPCC definido com os parâmetros:

- -T: quantidade de *threads*;
- -network: tipo de paralelismo para a execução (smp, mpi ou udp);
- -o: nome do executável a ser criado;

Já na execução, o sudo do UPCRUN, tendo parâmetros:

- –localhost: para o udp sendo necessário a definição para se ocorrer a execução;
- file: executável definido na compilação;

V. Metodologia

Fora criados *scripts* para realização de cada momento de teste, realizado para cada abordagem o uso de 2, 4, 6 e 8 *threads*, mais a sequencial ao inicio para ser comparada também. Neste *script* então utiliza-se em todas a entrada modificada para poder visualizar melhor seus resultados, tendo então a entrada de tamanho:

- TMAX: 2000
- NX: 3000
- NY: 3700

Com estes arquivos foram analisado com o do código sequencial e com os uso de *Threads* a maneira/quesito do tempo de cada processo, utilizando a função *time* do *bash*, no qual foi colocado no início de sua execução, tendo então os seus tempos.

TABLE I: Configurações do notebook

S.O	Debian 11
Processador	AMD Ryzen 5-3500U CPU @ 2.1GHz x 4 - 64 bits
Memória RAM	8 GB's
Memória SSD	300 GB's
Placa de Vídeo	AMD Radeon Rx Vega 8

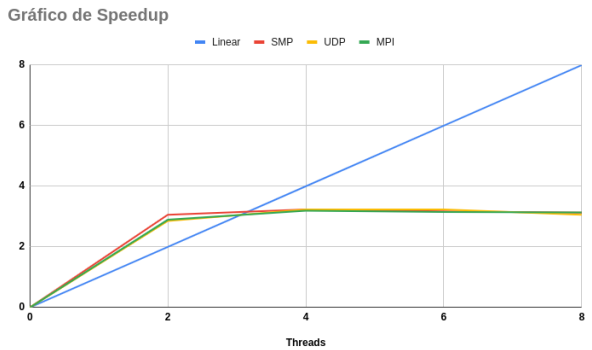
VI. Resultados

Iniciamos os resultados das análises. Tendo o uso do programa UPCC para a compilação e o UPCrun para a execução, para o estudo fora montado um *script* de teste o qual é executado sete vezes e pego a média de tais tempos para a futura realização do gráfico de *Speedup*. Para as *threads*, fora selecionado de 2 a 8, em passo de dois.

TABLE II: Resultados

	Sequencial	SMP	UDP	MPI
Sequencial	5m36,612			
2 threads		1m46,847s	1m54,474s	1m52,891s
4 threads		1m41,059s	1m41,044s	1m42,332s
6 threads		1m43,001s	1m41,066s	1m43,669s
8 threads		1m44,340s	1m41,739s	1m44,223s

Fig. 2: Gráfico de Speedup



VII. Análise e Discussão

Temos que para duas thread houve o speedup superlinear em todos as técnicas de paralelismo, tendo a aproximação quando feito com quatro *thread*, porém a partir então não houve melhora, isto ocorreu principalmente em causa do hardware utilizado para os testes, o qual possui apenas quatro slot de memória com núcleo físico e com isso quatro virtual, causando o desempenho limitado para a eficiência.

Qual a diferneça entre estas 3 versões paralelas, no desempenho.

Agora, focando na diferença entre os tipos de paralelismo, temos que o SMP se manteve melhor entre os três tipos, tendo o seu diferencial ao uso de duas threads, sabendo de que o OpenMP é uma forma de programar em dispositivos de memória compartilhada. Isso significa que o paralelismo ocorre onde cada thread paralela tem acesso a todos os seus dados. Já com o MPI por exemplo, se mantém na forma de programar com a memória distribuída. Isso significa que o paralelismo ocorre onde cada processo paralelo está trabalhando em seu próprio espaço de memória, isolado dos demais. Tendo a construção de ambos desenvolvido em maior parte pelo uso do UPCC, já que apesar de realizar o código com as devidas métricas de paralelismo, o uso para construção de cada network fora utilizado, e com isso então a diferença apesar de ser pouco poderia ser melhor distribuído se houvesse a realização com foco em cada, porém o estudo de momento permanece a este meio.

Conclusão

Finalizamos tendo que o projeto colaborou na experiência e elaboração de códigos com paralelismo com e sem a distribuição de memória, desenvolvendo melhor os conceito aprendidos em sala de aula e em documentações referenciadas. O estudo do paralelismo no algoritmo fdtd-2d em si elaborado atingiu as expectativas e há o bom desempenho, porém a análise entre as três versões se manteve próximas o qual as conclusões de diferencial a partir disto acaba por ser dificultadas.

References

[1] L. Calvin; SNYDER, "Snyder: Parallel programming," vol. 24, 2008.

[2] RookieHPC, "MPI documentation." https://www.rookiehpc.com/mpi/docs/index.php, 2019 - 2021. Online; acessado em 6 Novembro 2021.

[3] W. Kendall, "A Comprehensive MPI Tutorial Resource." https://mpitutorial.com/, 2021. Online; acessado em 6 Novembro 2021.

[4] B. Barney and L. L. N. Laboratory, "Message Passing Interface (MPI)." https://hpc-tutorials.llnl.gov/mpi/, 2021. Online; acessado em 6 Novembro 2021.