ENGENHARIA AEROESPACIAL E MECÂNICA
MECATRÔNICA
CE-265 PROCESSAMENTO PARALELO

Docente: Jairo Panetta
Discente: Lucas Kriesel Sperotto

Exercício 4 – RELATÓRIO

Paralelização por OpenMP do Produto “Matriz Esparsa X Vetor Denso” no Sistema “CROW”

Iniciado o trabalho com a implementação do proposto pelo professor, escolhido a linguagem C pois tenho mas afinidade. Para a implementação, precisei entender o funcionamento do método de armazenagem de matrizes esparsas, encontrei referências sobre o modelo Compressed Row Storage (CRS) e encontrei também um algoritmo que efetuava a multiplicação da matriz por um vetor denso utilizando esse modelo de representação da matriz esparsa.

O próximo passo era adequar a matriz no formato passado pelo professor para o formato CRS, lendo o arquivo armazenei os valores dois índices das linhas em um vetor, os índices das colunas em outro vetor, e por fim um vetor com os valores dos elementos não nulos. Para converter essa representação para o formato pedido, criei a função “ConvCRS()” que itera todas as posições do vetor com os índices das linhas e verifica as variações desses índices, dessa forma é possível reconhecer o primeiro elemento não nulo de cada linha, para as linhas não nulas um laço itera o numero das variações armazenando o índice do próximo não nulo.

O problema residia no fato de que o algoritmo encontrado para a multiplicação, necessitava que o vetor de índices contivesse, caso a ultima linha fosse não nula e contivesse mais de um elemento não nulo, de uma posição a mais para armazenar o índice mais um do ultimo elemento do vetor de elementos não nulos. Isso feito por maus um laço que percorre as ultimas posições do vetor de índices do primeiro elemento não nulo e armazena o ultimo índice do vetor de elementos não nulos acrescidos de um.

Após a implementação, foi executado com uma matriz padrão 4X4, multipliquei ela por um vetor denso unitário 3 vezes, fiz o calculo manualmente e comparei com o software cada uma das iterações (com printf) até rastrear os erros. Foi executada uma matriz identidade de 4X4 para verificar que o vetor não se alterava, e também adicionado mais duas linhas e colunas nulas as matrizes de teste de forma a ver a influência da ultima linha nula no calculo. Tendo comprovada a correta execução até essa parte, executado com as matrizes enviadas pelo professor no sistema “CROW”. Para executar no sistema “CROW” foram feitas alterações no “makefile” e no “script” de execução. Também foi criado

as variáveis para passar os argumentos ao software (argv). Mais detalhes podem ser visto no código fonte, comentei bastante e creio ter dado uma boa ideia do funcionamento através dos comentários.

A paralelização foi feita de varias formas, mas não conseguia encontrar uma logica correta, segui o conselho dado em aula de que deveríamos fazer com que cada thread executasse J iterações, bom, iniciei a região paralela antes desse laço, mas não dividi o trabalho ali. Um bom palpite era sincronizar criar barreiras, até mesmo "reduction" para o somatório, tive um relapso, a calculo do i-esimo elemento do vetor não depende de elementos anteriores dentro do laço, apenas as J iterações são dependentes. Como o vetor é calculado de elemento em elemento então a divisão de trabalho por schedule pareceu mais obvia por separar as i-iterações do laço de forma que uma parte do laço não acessa os, índices de memoria da outra parte do laço, coloquei dois schedules pois usei um vetor auxiliar para guardar o resultado da multiplicação e depois de tudo multiplicado esses valores são copiados para o vetor original e dar continuidade na recursão.

O vetor temporário para o calculo da multiplicação era privado, criado dentro da função que estava inserida na região paralela, retirando da região paralela e passando por parâmetro, percebi que o tempo diminuiu e muito, penso que por não ter de alocar varias vezes um vetor que terá apenas uma parte utilizada.

Os resultados para o primeiro arquivo não foram muito bons, mantive tempo decrescente mas o speed-up não cresceu muito com o aumento das threads, creio que pelo problema ser pequeno, a matriz não era muito grande, apesar te ter muitas iterações, a parte paralelizada não foi as iterações mas sim a multiplicação dos elementos da matriz, o que garantiu maior rendimento nas matrizes maiores.

Tabela 1 - Esparsa_1.txr - 20000 Iterações

Nº Threads	Tempo	Speed-Up	Eficiência
1	0,6593120	1	100
2	0,3590490	1,836273044	91,81365
3	0,3017790	2,184751093	72,82504
4	0,2511570	2,625099042	65,62748
5	0,2138760	3,082683424	61,65367
6	0,1864980	3,535222898	58,92038
7	0,1928440	3,418887806	48,84125
8	0,1785960	3,69163923	46,14549

Tabela 2 - Esparsa_2.txt - 1300 Iterações

Nº Threads	Tempo	Speed-Up	Eficiência
1	0,6362040	1	100
2	0,3182970	1,998774729	99,93874
3	0,2170930	2,930559714	97,68532
4	0,1645030	3,867430989	96,68577
5	0,1321640	4,813746557	96,27493
6	0,1131190	5,624201063	93,73668
7	0,0965750	6,587667616	94,10954
8	0,0874290	7,276807467	90,96009

Podemos ver comparando as tabelas que quanto mais aumentamos o tamanho da matriz maior é o ganho de paralelização. Como comentado anteriormente pelo aumento do problema, já que perdemos eficiência coma alocação e processos relacionados a divisão das threads.

Considerei a paralelização correta por ter executado o software sequencial com as matrizes dadas e o valor da norma2 não se alterou quando alterava o numero de threads, não houve tempo para testar com as matrizes criadas por mim para ter uma segurança maior.

Tabela 3 - Esparça_3.txt - 130 Iterações

Nº Threads	Tempo	Speed-Up	Eficiência
1	0,6324860	1	100
2	0,3165970	1,997763719	99,88819
3	0,2173610	2,909841232	96,99471
4	0,1653050	3,826175857	95,6544
5	0,1315700	4,807220491	96,14441
6	0,1101160	5,743815613	95,73026
7	0,0947540	6,675032189	95,3576
8	0,0818600	7,726435377	96,58044

Para concluir, gostei muito desse trabalho, pelo fato de ter que levar o nosso pensamento a entender o funcionamento de cada thread, como que os dados serão tratados dentro da região paralela e fora dela. Acredito que aprendi muito nesse ponto, já que tive de abstrair esses conceitos mentalmente e verificar as diretivas de divisão de trabalho que mais se adequavam ao problema.