Relatório do Trabalho Prático 2

Ana Gonçalves
MEI
pg50180

Francisco Toldy
MEI
pg50379

***Abstract*—Este trabalho tem como objetivo a avaliação da capacidade dos alunos para desenvolver programas que explorem paralelismo tendo como principal objetivo a redução do tempo de execução do programa desenvolvido no trabalho prático anterior.**

I. Introdução

O enunciado deste trabalho começava por apresentar a tarefa como continuação do trabalho prático anterior, sendo assim necessário uma base robusta para o mesmo. Tal como no trabalho anterior, foram apresentados vários pontos a seguir sendo estes a identificação dos blocos de código com maior carga, a apresentação e analise de alternativas para exploração de paralelismo, a seleção da alternativa mais viável bem como a implementação da mesma e análise do seu desempenho. Neste documento iremos abordar cada um desses passos. É importante referir ainda que todos os testes foram realizados com 4 centroids e 10 000 000 pontos e um limite máximo de 20 iterações, com exceção do gráfico de escalabilidade grande, onde foi adicionado testes com 32 centroids.

II. Blocos de código com maior carga

Para este ponto, foi executado um pequeno script .sh de forma a realizar os comandos *perf record* e *perf report* no SEARCH para a solução apresentada no trabalho anterior. Seguem-se na imagem seguinte os resultados:

Overhead	Command	Shared Object	Symbol
61.83%	a.out	a.out	[.] kmeans._omp_fn.1
21.41%	a.out	a.out	[.] updateCentroidCoord
4.72%	a.out	libc-2.17.so	[.] __random
3.36%	a.out	libc-2.17.so	[.] __random_r
0.97%	a.out	[unknown]	[k] 0xffffffffa1195377
0.93%	a.out	a.out	[.] init._omp_fn.0
0.48%	a.out	libc-2.17.so	[.] rand
0.34%	a.out	[unknown]	[k] 0xffffffffa0e6d5f8
0.18%	a.out	[unknown]	[k] 0xffffffffa158c48a
0.15%	a.out	[unknown]	[k] 0xffffffffa158c4ef
0.11%	a.out	[unknown]	[k] 0xffffffffa0ee4b37
0.09%	a.out	a.out	[.] rand@plt
0.08%	a.out	[unknown]	[k] 0xffffffffa0e6d5fa

Fig. 1. Excerto do report do perf sem paralelização

Como podemos observar, a função com maior percentagem de tempo gasto foi a função principal, kmeans. Este é o resultado que seria de esperar, uma vez que a função kmeans é responsável por grande parte das operações da solução, nomeadamente a passagem pelos pontos todos (implicando cálculo do centroid novo para cada um deles) em cada uma das 20 iterações.

Em alternativa à vista apresentada no perf report, segue-se a vista em árvore:

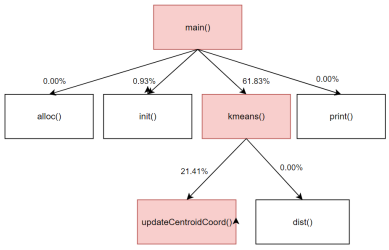

Fig. 2. Tree View sem paralelização

Ficam assinalados na tree view o hotspot já mencionado (kmeans) bem como a segunda função com maior tempo gasto (updateCentroidCoord())

III. Diferentes alternativas para exploração de paralelismo para cada bloco identificado em 1

Dado o bloco identificado anteriormente , foi necessário analisar o código a que este se referia. Assim, olhando para a função kmeans , foi possível identificar imediatamente que o ciclo principal do...while seria a causa para o gasto deste bloco. Dentro do ciclo do...while, foi possível identificar o candidato a paralelização , o ciclo for que analisa cada um dos pontos, invocando depois a função updateCentroidCoord depois de calcular a distância para cada um dos centroids. Este ciclo for seria um bom candidato à aplicação da primitiva **omp parallel for** já que não existe situação de data race.

Após encontrado o ponto onde aplicar a primitiva for, apresentou-se outra alternativa, usar schedule(static), schedule(dynamic) e schedule(guided) nessa mesma primitiva.

Outros pontos considerados como possíveis candidatos a implementação de paralelização foi a função **init** e também a função **updateCentroid**

IV. Selecionar a alternativa mais viável justificando com uma análise de escalabilidade

Relativamente às alternativas apresentadas na secção anterior, as primeiras a serem descartadas foi a paralelização na função **init** e na função **updateCentroid**. A primeira das funções não foi paralelizada na versão final uma vez que os resultados finais mostram uma variação grande comparada à versão original. A segunda das funções não foi paralelizada uma vez que causava uma situação de datarace cuja resolução implica a criação de uma secção sequêncial, e por isso, o grupo decidiu não se focar nessa paralelização. Assim,

a última decisão a tomar seria recorrer ar **schedule(static)** ou **schedule(dynamic)** ou **schedule(guided)** sendo que foi selecionada a opção guided. Isto deveu-se ao facto de, ao executarmos 3 tentativas com 16 threads com cada uma das 3 primitivas, as suas médias serem as seguintes :

static	guided	dynamic
1.308	1.23	13.128

Estas medições devem-se ao facto do dynamic fazer um melhor balanceamento em algoritmos com muita diferença computacional em cada iteração. No entanto, KMeans não é um desses casos, visto que em cada iteração, a carga computacional é razoavelmente a mesma. Logo, o uso de dynamic scheduling implicaria apenas adição de um overhead, que resulta neste aumento no tempo de execução

V. Analisar o desempenho da proposta implementada

Após a implementação da versão mais adequada de paralelização, podemos verificar os resultados em anexo relativamente ao report e a tree view resultantes. Como podemos verificar, ocorreu uma paralelização satisfatória, visto que conseguimos observar que o tempo de execução nas funções que anteriormente ocupavam uma grande percentagem de tempo, diminuíram, juntamente com o aumento das restantes funções. Isto decorre da diminuição efetiva de tempo gasto nessas funções, que resulta neste efeito em termos relativos.

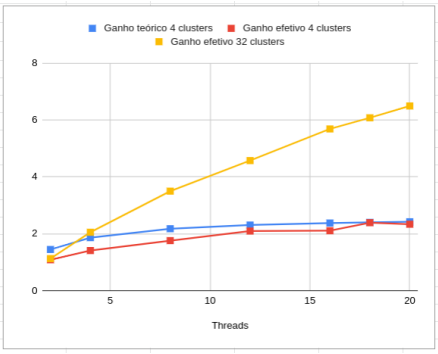

Fig. 3. Gráfico de Escalabilidade Grande

Numa análise de escalabilidade grande, verificamos que existe alguma semelhança entre os resultados obtidos e os previstos na seguinte figura, em que estes últimos foram obtidos a partir de uma aproximação de percentagem de código paralelizado (100%- 61.83% = 38.2%)

Como podemos observar, existe um aumento aproximadamente linear até 12 threads, como era de esperar, como também a esperada estabilização das 12 para as 16 threads.

No desenvolver desta análise, procuramos também saber o porque deste aumento não ser como esperado e exploramos as seguintes hipóteses:

A. *Limitação da Largura de Banda da Memória*

Para verificar se existe uma limitação na largura de banda da memória foi verificado o aumento do CPI com o aumento do número de threads. Assim obteu-se os seguintes resultados:

Número de Threads	CPI
1	0,435
2	0,671
4	0,689
8	0,719
16	0,763
18	0,781
20	0,847

Verificamos efetivamente que existe um ligeiro aumento de CPI com o aumento do número de threads, concluindo então que os resultados podem não ser ideais possivelmente devido ao limite da largura de banda da memória ou à cache. Apesar de na primeira fase já ter sido realizado otimizações para melhorar a localidade espacial (arrays de estruturas), o que reduz os caches misses e consequentemente redução de número de acessos à memória, pode existir ainda uma limitação nos acessos à memória. Isto deve-se ao facto de que com a paralelização existe um maior número de cache misses e acessos à memória

B. *Paralelismo de granulação fina*

Com o objetivo de confirmar se está a ocorrer paralelismo com a granulação demasiado fina verificamos se existia uma variação muito grande entre o número de instruções na versão sequencial e na versão paralelizada. Após essa verificação, concluímos que não existe uma variação significativa , o que vai de acordo com o esperado, visto que é criado apenas uma secção com uma equipa de threads.

C. *Excesso de Sincronização de Paralelização*

O facto de não não existir variáveis partilhadas na área paralelizada, não foi criado uma sincronização visto que não interferia com a correção dos resultados. Logo, esta possibilidade torna-se impossível para a justificação da não igualdade dos resultados teóricos e os reais.

D. *Mau Balanceamento da Carga*

Finalmente, a última hipótese é o mau balanceamento da carga por cada uma das threads. Logo, os tempos de execução foram comparados com os diferentes scheduling, e verificou-se que o static ou guided tiveram resultados bastante semelhantes e satisfatórios, em comparação com o dynamic. O que implica que o balanceamento da carga não era um problema para este algoritmo.

VI. Conclusões

No fim do tempo disponibilizado para desenvolver este trabalho, o grupo considera que foram cumpridos os objetivos propostos, sendo implementado paralelismo recorrendo ao OpenMP, possibilitando a observação de melhorias significartivas do tempo de execução do trabalho entregue anteriormente. Além disso foi feita uma análise dos resultados práticos apoiada pelo conhecimento transmitido nas aulas teóricas e práticas desta UC.