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 de forma eficiente tendo como principal objetivo a redução do tempo de execução do programa desenvolvido no trabalho prático anterior, explorando utilizando ambientes de programação paralela alternativos.**

I. Introdução

O enunciado deste trabalho apresentava a tarefa em vários pontos. O primeiro ponto seria corrigir e/ou otimizar a versão OpenMP entregue no trabalho anterior. Além disso , também estava incluído a utilização de métodos alternativos como CUDA e MPI para tentar melhorar o tempo de execução. O segundo ponto seria a realização de testes com maior liberdade , seja para a utilização das máquinas pessoais, seja no número de pontos. Isto teria o objetivo de testar o programa entregue em vários ambientes e condições, sendo sugerida também a utilização de tamanhos de input adequados a cada um dos níveis da hierarquia de memória. Por último, era ainda necessária uma análise dos resultados teóricos obtidos.

II. Trabalho Anterior e OpenMP

A solução apresentada na fase anterior foi testada pelos docentes no que toca a Data Races, sendo que não foram encontrados quaisquer exemplos de tal acontecimento. Assim, o foco do trabalho nesta fase passou para encontrar uma alternativa à solução previamente entregue.

III. Exploração de métodos alternativos de paralelismo

Dados os exemplos fornecidos nas aulas e no enunciado, o grupo tomou a decisão de utilizar o MPI - Message Passing Interface. Para tal foi necessário retirar as primitivas openMP e iniciar o processo de desenvolvimento em MPI. Como na fase anterior, foi necessário identificar os pontos onde seria útil e possível implementar o MPI. Para tal, o grupo baseou-se nas conclusões do trabalho anterior, de que os hotspots estavam localizados na função kmeans bem como na função updateCentroidCoord().

IV. Base MPI

A versão inicial a partir do qual se realizou a implementação do MPI era baseada em Arrays de Structs de Pontos e de Centroids. A decisão de basear a implementação do MPI na versão AoS justifica-se pelo facto de ser necessário fazer Scatter dos dados. Este Scatter está dependente do tamanho dos dados, e caso o grupo estivesse a utilizar um Struct de arrays não se iria verificar uma grande melhoria nos resultados (conclusão do trabalho anterior). Isto deve-se ao facto de que em testes anteriores a utilização de Struct de arrays não melhorava o desempenho num factor significativo. Adicionalmente, a utilização do scatter com Struct de Arrays implicaria a existência de mais instâncias de scatter para distribuir a informação necessária para cada um dos processos.

A solução apresentada divide o conjunto de pontos pelos processos , sendo que cada um dos processos irá ficar responsável por esse conjunto de pontos em toda a execução do programa. Todos os processos vão calcular os novos centroids para cada um dos pontos na sua responsabilidade e depois ocorrerá um somatório das coordenadas dos centrois e o número de pontos a que ele estão agrupados.

V. Implementação

Foram necessárias algumas alterações a nível do kmeans. Para que cada processo pudesse trabalhar com o mesmo conjunto de pontos, é efetuado um scatter dos pontos. Para tal, foi necessário começar por criar 2 data types novos do MPI equivalentes (dt_point e dt_centroid) à struct Ponto e à struct Centroid. O cálculo do buffer enviado para cada um dos processos é feito ao dividir o número de pontos pelo número de processos, assegurando que o programa irá ser capaz de funcionar para quaisquer inputs. Além da necessidade de distribuir conjuntos de pontos pelos processos, é também necessário a distribuição dos centroids através de um broadcast. Através destes novos tipos, vais ser possível enviar as estruturas por mensagens para os restantes processos. Após cada um dos processos processar cada um dos seus novos centroids vai ocorrer uma redução para sincronizar todos as instâncias de centroids para o processo de rank 0. No entanto, não existe uma função existente e adequada para fazer a redução entre as estruturas de centroids, logo foi criada uma manualmente: centroid_reduce_function que vai somar todas as coordenadas x, y e o número de pontos de todos os centroids.

Finalmente, vai ser feito um reduceall final para sincronizar a flag de continuação do ciclo entre todos os processos. Este reduce é diferente do anterior no sentido em que é necessário alterar todas as flags para o mesmo valor. Isto acontece apenas porque o ciclo só pode ser finalizado quando nenhum dos pontos altera, logo, se apenas um alterar, todos os outros processos com os restantes pontos têm de se manter no ciclo também.

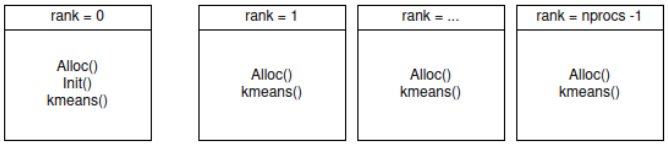

Fig. 1. Diagrama geral de funcionamento entre processos

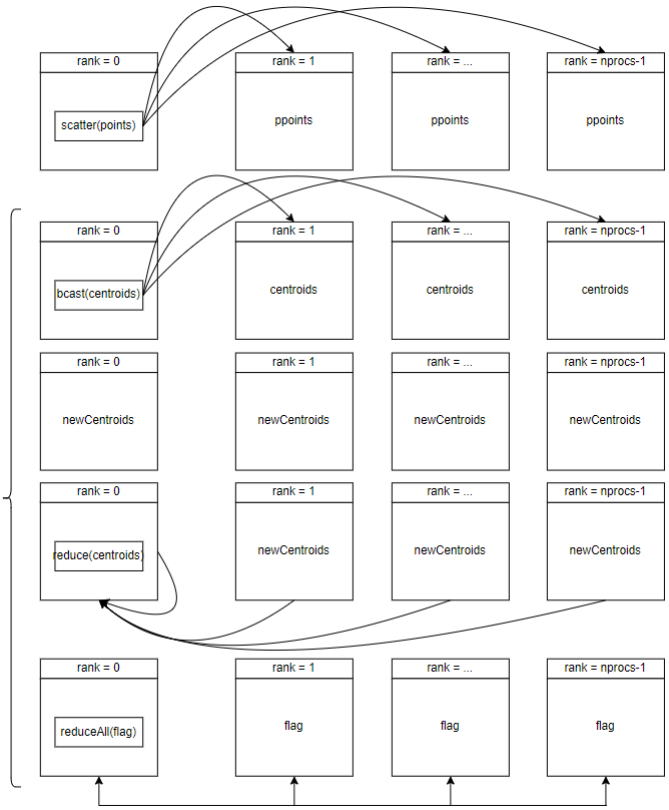

Fig. 2. Diagrama de funcionamento entre processos

VI. Ambiente de Exploração e Análise

Devido à possibilidade de realizar a exploração e análise de performance noutros ambientes, o grupo optou por escolher uma máquina pessoal devido a restrições existentes no ambiente Search relativamente à disponibilidade de eventos na ferramenta PAPI.

Assim sendo, foi utilizada a ferramenta PAPI com base nos eventos que estavam disponíveis na máquina pessoal:

- Número de instruções
- Número de Clock Cycles
- Número de L2 cache misses

Em relação à máquina pessoal, é de notar algumas das suas características que são importantes para melhor entender os resultados da secção seguinte:

- Processador: AMD Ryzen 4700U
- Gráfica: Radeon graphics
- Cache L1i: 256kiB
- Cache L1d: 256kiB
- Cache L2: 4MiB
- Cache L3: 8MiB

VII. Exploração e Análise

Criada uma solução baseada na implementação do MPI, passou-se então para a fase de testes. Era recomendado no enunciado a utilização de inputs de dados adequados ao tamanho das caches, no entanto, não conseguimos fazer a análise do programa relativamente a L1 cache misses nem a L3. Logo, a análise relativamente à L2 cache misses é a única que vai ser analisada neste relatório.

A. Análise de Escalabilidade Forte

De modo a analisar a escalabilidade forte do programa, foi agrupado um conjunto de valores relativamente aos tempos de execução. Mais especificamente, com a variação do número de processos utilizados para um tamanho fixo de números a agrupar pelo k-means. Assim apresentamos um gráfico da relação do speed-up com o número de processos:

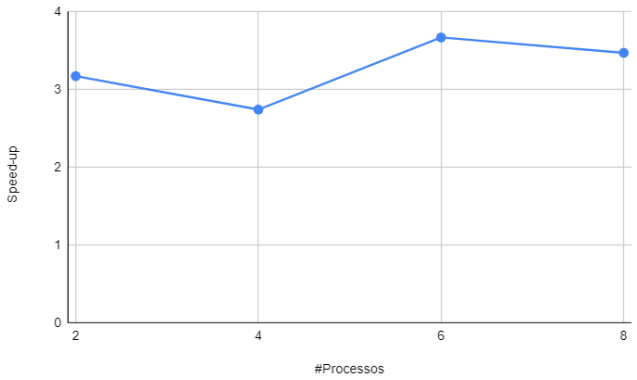

Fig. 3. Diagrama de funcionamento entre processos

Como podemos verificar, existe um ligeiro aumento no speed-up com o aumento do número de processos. No entanto, era esperado pelo grupo uma variação maior, de modo a que o speed-up tivesse um aumento ligeiramente maior. Isto pode-se dever ao facto da existência de um overhead sobre o MPI, visto que se trata da troca de mensagens entre processos, que pode vir a ser mais custosa do que por exemplo a comunicação entre threads.

B. Análise de Escalabilidade Fraca

No que toca à escalabilidade fraca do programa desenvolvido, foram feitos testes aumentando a quantidade de dados de forma proporcional ao número de processos do MPI. Assim, obtivemos dados para 2 processos e 5 milhões de pontos, 4 processos e 10 milhões de pontos e 8 processos e 20 milhões de pontos.

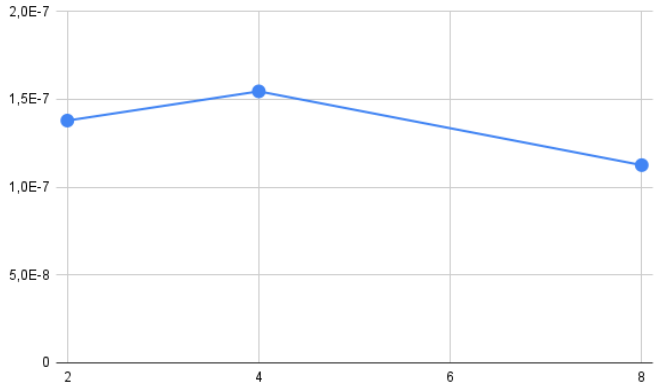

Fig. 4. Variação de Tempo/Pontos com o aumento dos processos

Como se pode observar no gráfico seguinte, o valor de Tempo dividido pelo número de pontos variou de forma pouco significativa (entre 1,1E-7 e 1,54E-7) nas situações estudadas. Olhando puramente para o tempo médio de execução, podemos observar que este aumentou de forma semelhante ao aumento de pontos e aumento de processos.

Apesar destas medições, dado a dificuldade de utilização do perf em conjunção com um programa MPI, não pudemos comprovar a diminuição da fração de trabalho serializado com o aumento do problema, tal como mencionado nas aulas teóricas da unidade curricular.

C. Hierarquia de Memória

Numa primeira análise é de notar imediatamente que existe uma descrepância entre a quantidade de cache misses no processo de rank 0 comparativamente aos outros.

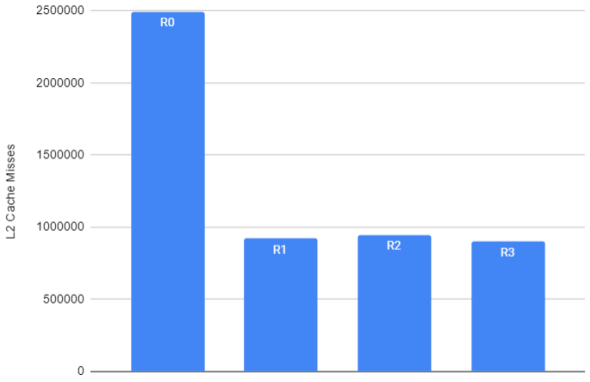

Fig. 5. L2 cache misses em cada rank com 4 processos

TABLE I
MÉDIA DE CACHE MISSES EM CADA RANK PARA 4 PROCESSOS E 10 MILHÕES DE PONTOS

L2 DCM Total	Rank 0	Rank 1	Rank 2	Rank 3
5261042,75	2494021	920807,75	945730	899484

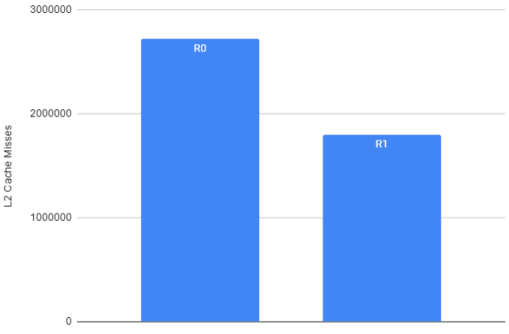

Fig. 6. L2 cache misses em cada rank com 2 processos

Isto pode ser justificado principalmente pela função init() que é executada apenas pelo rank 0 .

D. Número de Instruções e Número de Ciclos de Relógio

Na extração de dados relativamente a número de instruções e clock cycles, verificamos que ambos seguem um padrão muito semelhante. Isto é, com o aumento de processos, existe uma redução significativa em ambos. O número de clock cycles diminuir pode estar relacionado com a distribuição balanceada da carga pelos processos, isto é, vai permitir uma melhor utilização de recursos. Adicionalmente, ao distribuir a carga pelos processos, cada um deles vai estar atribuido um conjunto menor de pontos, que podem ser guardados em maior quantidade nas suas caches. Logo, não vão existir tantas caches misses e consequente menos clock cycles.

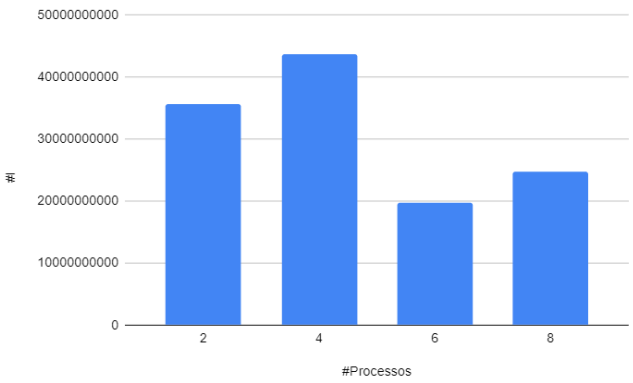

Fig. 7. Número de instruções pelo número de processos

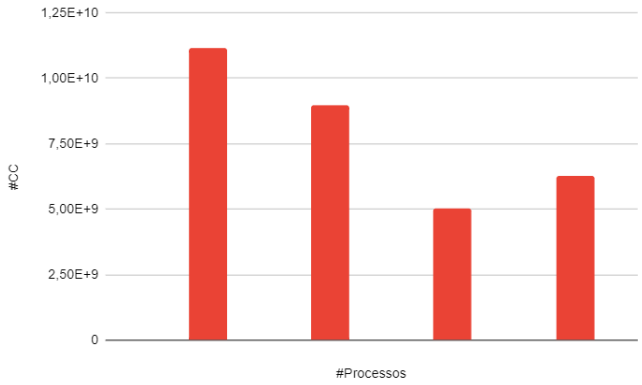

Fig. 8. Número de clock cycles pelo número de processos

VIII. Conclusão

No fim do tempo disponibilizado para desenvolver este trabalho, o grupo considera que foram cumpridos os objetivos propostos, sendo implementado paralelismo recorrendo ao MPI (escolhido pelo grupo), possibilitando a observação do impacto da alteração de vários parâmetros envolvidos. Permitiu também obter um conhecimento mais profundo sobre ferramentas de paralelização como também todos os riscos que lhe acompanham.