Computação Paralela - Trabalho Prático 2

1st João Vitor Vieira
Universidade do Minho
Braga, Portugal
PG50501

2nd José Pedro Magalhães
Universidade do Minho
Braga, Portugal
PG50528

***Abstract*—Este trabalho tem como objetivo aprimorar o algoritmo k-means realizado anteriormente, fazendo uso de técnicas de paralelismo atráves da interface OpenMP, de forma a melhorar vários aspetos do algoritmo, tais como o seu tempo de execução, número de ciclos, entre outros.**

***Index Terms*—paralelismo, schedule, openmp, escalabilidade, thread, multithreading**

I. Introdução

Para a realização do segundo trabalho prático da cadeira Computação Paralela foi necessária a melhoria do algoritmo k-means, de forma a que este, através de threads, possa realizar paralelismo utilizando as primitivas OpenMP, tornando o algoritmo melhor e mais eficiente. A criação desta nova versão implicou várias mudanças no código já feito, de forma a que o mesmo seja capaz de se tornar mais eficiente e mais compatível à utilização de paralelismo.

II. Alterações no código

Face ao enunciado do trabalho, foram realizadas algumas mudanças no código de forma a que a implementação de paralelismo compense. Primeiramente, foram retiradas as estruturas de dados e criados arrays dinâmicos, para as coordenadas x,y e assim como para as informações dos centroids de cada cluster.

Além disso, foi necessário receber como input o número de threads, clusters e pontos.

Relativamente aos ciclos de tamanho N (número de pontos), utilizou-se loop unrolling manual de tamanho 4 e foram simplificados os cálculos matemáticos da distância de um ponto a um cluster fazendo melhor uso de memória e evitando acessos repetidos a arrays dinâmicos. Foi também substituído o segmento de código:

```
if(dist < menorDist){
    menorDist = dist;
    menorCluster = j;
}
```

pelas seguintes expressões:

```
menorCluster =
dist < menorDist ? j : menorCluster;

menorDist =
dist < menorDist ? dist : menorDist;
```

uma vez que chegámos à conclusão que estas representam um ganho considerável de tempo.

Por fim de modo a evitar zonas críticas no ciclo for que contém as computações mais pesadas (cálculo das distâncias) apenas guardamos o menorCluster num array na posição i, ou seja, na posição associada ao index de cada ponto. Este array é depois iterado de modo a atualizar os valores dos tamanhos dos clusters e os centroids temporários, separando assim os pesados cálculos matemáticos das atualizações de valores que acedem a memória partilhada.

III. Balanceamento de carga

A. OpenMP

O OpenMP é uma interface de programação que trabalha na implementação de multithreading, e foi a ferramenta principal para a implementação de paralelismo.

Inicialmente, o objetivo foi verificar quais os blocos de código com maior peso computacional e, após tal observação, verificou-se em quais blocos seria rentável inserir o paralelismo e em caso afirmativo, qual a melhor alternativa.

B. Multithreading

A primeira tentativa foi realizar multithreading nos ciclos for, utilizando o

```
#pragma omp parallel for num_threads(W)
```

nos blocos de código mais pesados, sendo W o número de threads criadas escrito no input do algoritmo. Este comando irá criar W threads, e indica ao compilador para paralelizar o ciclo escolhido, separando as iterações do ciclo em questão por cada thread.

Foi testado o código acima referido em cada um dos ciclos pesados do algoritmo, porém o código apenas se tornou efetivamente mais rápido quando colocado apenas no ciclo que calcula as distâncias de cada ponto a cada cluster.

C. Zonas Críticas

Na primeira fase do trabalho, o código possuia algumas zonas onde memória teria de ser acedida por diferentes threads ao mesmo tempo. Inicialmente foi testado colocar nessas secções:

```
#pragma omp critical
```

que indica que a secção seguinte só pode ser acedida por uma thread de cada vez.

Após testar a utilização de ambos, o tempo de execução do algoritmo aumentou considerávelmente tal como seria esperado uma vez que as threads teriam de esperar pela sua vez e verificar constantemente por permissões de acesso a esta área. Para eliminar este problema optou-se por mover o bloco de código para fora do ciclo onde existia paralelismo, melhorando de modo considerável o tempo de execução do programa.

D. Scheduling

Na situação anteriormente referida, utilizando o paralelismo de forma normal, é vísivel que o mesmo separa o número de iterações de um ciclo de maneira igual por cada thread.Assim sendo, o schedule é utilizado para melhorar a maneira como as iterações são separadas, pois há vários tipos de situação onde é melhor ter threads a correr mais iterações do que outras.

Para o algoritmo em questão foram testados os três tipos de schedule, static, dynamic e guided. A diferença principal entre ambos é que o schedule(static) separa as iterações entre as threads antes do primeiro ciclo começar, com as informações inicias, enquanto que o schedule(dynamic) altera o número de iterações para cada thread enquanto as mesmas estão a ser iteradas. Já o schedule(guided) é semelhante ao dynamic porém o tamanho do bloco que cada thread recebe começa grande e vai diminuindo cada vez mais para lidar com o desiquilibrio.

Incluindo os três schedules separadamente, foi possível verificar que nenhum deles melhorava significativamente o tempo de execução do código o que também era esperado pelo grupo uma vez que o ciclo for em que estes foram utilizados executam sempre as mesmas instruções e na mesma quantidade, como tal os benefícios suportados por este método não superam o overhead do seu funcionamento.

IV. Análise de escalabilidade

O algoritmo é capaz de correr sequencialmente ou através de várias threads, utilizando o comando runseq ou runpar, respetivamente. No que toca a escalabilidade, é preferivel utilizar o algoritmo particionando as iterações por cada thread. Desta forma, é possível aumentar o número de clusters e o aumento do tempo de execução ser bastante reduzido.

Por outro lado, na eventualidade de um acréscimo no número de clusters correndo de forma sequencial o aumento do tempo de execução é bastante significativo, pois o bloco de código mais significativo são 2 ciclos, de N(número de pontos) e K(número de clusters), isto é, o número de iterações realizadas são N x K, logo, com o aumento do número de clusters o tempo de execução sobe linearmente.

Resumidamente, apesar de nas duas alternativas, com o acréscimo do número de clusters, o tempo de execução aumentar linearmente, correr o algoritmo paralelamente, cada thread apenas irá iterar (N(número de pontos)*K(número de clusters))/W(número de threads), que é significativamente menor do que de forma sequencial, que é como foi dito, N*K.

V. Medição do perfil de execução

Como é possível verificar nas tabelas em baixo, correr o algoritmo utilizando o multhreading torna-o notavalmente mais eficiente. Porém, a partir de 8 threads não há melhorias de tempo significativas. Isto deve-se ao facto de que, a partir desee número, a implementação de mais threads não compensa o tempo que as mesmas encurtam, pois não há carga de trabalho que as torne necessárias.

TABLE I
Resultados com 32 Threads em paralelo

	4 Clusters	32 Clusters
Time Elapsed	0.88 seconds	2.21 seconds
Instructions	21,903,581,152	118,384,617,505
Cycles	8,463,040,188	43,226,910,152
Insn per Cycle	2.22	2.22
CPI	0.4	0.4

TABLE II
Resultados com 8 Threads em paralelo

	4 Clusters	32 Clusters
Time Elapsed	0.89 seconds	2.31 seconds
Instructions	20,169,496,957	116,801,828,371
Cycles	8,346,583,719	40,515,505,988
Insn per Cycle	2.42	2.88
CPI	0.4	0.3

TABLE III
Resultados com 1 Thread sequencial

	4 Clusters	32 Clusters
Time Elapsed	2.43 seconds	22.19 seconds
Instructions	19,694,795,232	116,339,290,187
Cycles	7,295,311,265	39,752,901,391
Insn per Cycle	2.70	2.93
CPI	0.4	0.3

VI. Conclusão

A realização do presente trabalho permitiu estudar e compreender melhor as primitivas OpenMP, no sentido em que se revelou bastante útil no estudo de paralelismo, nomeadamente, multithreading, scheduling, variáveis privadas, entre outros.

Através do estudo desenvolvido pode-se entender o impacto e a importância da utilização de threads, neste caso no algoritmo em questão.

References

1 - https://www.openmp.org/resources/refguides/
2 - https://gcc.gnu.org/onlinedocs/