Faculdade de Ciências da
Universidade do Porto

Computação Paralela

All-Pairs Shortest Path Problem

José Pedro Sousa
201503443

9 de Dezembro de 2020

1 Introdução

Com a evolução da tecnologia, os computadores estão cada vez mais rápidos de acordo com o poder computacional, principalmente nos processadors. Isto está de acordo com a Lei de Moore, em 1965, que: "o número de transistores num microchip duplica a cada dois anos"e, por sua vez, os processadores conseguiram ficar cada vez mais rápidos e resolver cada vez mais problemas. Porém esta Lei de Moore tem vindo, a partir de 2015, a não ser verificável e por isso teve que se optar por outros métodos para resolver problemas de grande escala computacional. A solução foi a computação paralela através de vários *cores* dentro do processador ou até mesmo vários *cores* de diferentes processadores ou *clusters*. Este trabalho tem como objetivo a representação de um algoritmo para um determinado problema em que se recorre à imlementação de computação paralela sobre a linguagem de programação C. No final, também haverá uma comparação de resultados.

2 Desenvolvimento

2.1 Problema

O problema *All-Pairs Shortest Path Problem* tem como objetivo determinar todos os caminhos mais curtos entre pares de nós em um dado grafo.

A ideia é simples: dado um grafo direcionado G = (V, E) em que V é o conjunto de nós (ou vértices) e E é o conjunto de links (ou arestas) entre os nós, o objetivo é determinar para cada par de nós *(vi, vj)* o tamanho do caminho mais curto que começa em *vi* e termina em *vj*. Um caminho é simplesmente uma sequência de links em que o nó final e o nó inicial de dois *links* consecutivos são iguais. Um *link* pode ser visto como um tuplo da forma *(vi, vj, t)* onde *vi* é o nó de origem, *vj* é o nó de destino e *t* é o tamanho do *link* entre os nós. A soma dos tamanhos dos links é o tamanho do caminho entre *vi* e *vj*.

O *input* deste problema é em forma de um *array* 2D de tamanho NxN em que cada coordenada *(vi, vj)* é o caminho do nó de origem *vi* até ao nó de destino *vj*.

O *output* é um *array* 2D de tamanho NxN em que cada coordenada *(vi,vj)* é o caminho minímo do nó de origem *vi* até ao nó de destino *vj*.

2.2 Paralelismo

Porém mesmo sendo um algoritmo e um problema simples para pequenos *inputs*, à medida que o número e o tamanho dos *inputs* aumenta, maior será o tempo de execução. Por isso em vez de o programa executar em apenas um processador, este irá executar em um ou mais processadores. Esta gestão de tarefas entre os processadores é realizada principalmente através do algoritmo Fox.

2.3 Algoritmo Fox

O algoritmo Fox usado para fazer parte na multiplicação de matrizes sendo eficiente num ambiente de computação paralelo. No contexto deste problema, o algoritmo *Fox* foi alterado para realmente calcular os caminhos minímos em vez de multiplicar números. Este algoritmo tem como objetivo dividir a matriz com tamanho N em submatrizes para que cada processador P possa calcular a submatriz solução de tamanho N/Q.

No entanto para que este algoritmo seja possivel aplicar irá ter que obedecer a algumas regras relativamente ao número de processadores e ao tamanho da matriz inserida. Dado um tamanho N da matriz inserida e P número de processadores, existe Q tal que P=Q*Q e N *mod* Q seja igual a zero.

2.4 Implementação

No início da implementação, criou-se um programa sequencial sem paralelismo com o objetivo de solucionar um algoritmo capaz de resolver o problema principal. Nesta implementação foi discutido a maneira de como se ia gerir a gestão de memória das matrizes(estática ou dinâmica) e qual a melhor maneira de alocar na memória.

Depois de estar concluído, o programa sequencial e com alguns resultados prontos, procedeu-se à realização do programa com os algoritmos a executar em ambiente de computação paralelo.

2.4.1 Principais dificuldades

Ao longo da implementação encontraram-se muitos problemas de falha de segmentação em memória devido ao método de alocação da memória para matrizes e o acesso às mesmas. Este problema foi rapidamente resolvido.

Também ao longo da implementação do problema foi usado uma variável global constante que representa o valor infinito para esta ser usada no cálculo dos valores da matrizes ao longo da execução. Esta variável foi atribuída valores altos, em que um deles foi o INTMAX mas teve que ser anulada pois dava problemas na execução e cálculo das matrizes. Foi encontrado o valor SHRTMAX(máximo de um número do tipo *short int*) que resolveu este problema.

Foi também discutido qual seria o melhor uso na comunicação entre processos, ou seja, comunicações com bloqueio ou sem bloqueio. No final foi decidido o uso de comunicações com bloqueio entre processos através dos metodos *MPI_Sendrecv*. Ainda se procedeu ao uso de funcões sem bloqueio, principalmente o *MPI_Scatter* e o *MPI_Gather* mas originou problemas que não se conseguiram resolver e por sua vez foi descartado o seu uso.

Com o uso do *MPI_Sendrecv* originou alguns problemas nos cálculos dos *outputs* da matriz solução, que foi resolvido por um *sleep(0)* antes da chamada a este método *MPI*. É de salientar que este não mostrou impacto no tempo de execução do programa.

3 Performance

3.1 Preparação

Relativamente à execução do programa, foi testado numa máquina inicial para testes mais simples e com um número de *quatro cores* de processamento. Mais tarde foi usado um *cluster* de computadores do Departamento de Ciência de Computadores com cada um quatro cores, sendo neste um total máximo operacional aproximadamente de cento e quatro cores que nao foi de todo usado na maioria (o teste com mais cores foi com trinta e seis cores).

Para compilar este programa basta executar o comando *$make*. De seguida, para correr o programa, basta executar o comando *$mpirun –hostfile hostfile -np P trab*, em que o P é o número de processadores que pretende executar.

3.2 Resultados

Os testes são realizados por vários *inputs* de diferentes dimensões. É de salientar que a unidade de tempo dos ressultados estão em segundos. Nas seguintes imagens ilustra-se os resultados no cluster:

	1 CPUs	4 CPUs	9 CPUs	16 CPUs	25 CPUs	36 CPUs
INPUT5	0.00017	0	0	0	0.00964	0
INPUT6	0.00023	0.00042	0.00768	0	0	0.01001
INPUT300	2.87943	0.69908	0.64909	0.41665	0.4639	0.44517
INPUT600	27.82541	7.32469	4.63481	2.68548	3.2945	2.96129
INPUT900	108.14271	25.90367	15.27361	8.41042	7.54851	6.14277
INPUT1200	335.91903	72.06311	40.78144	21.86553	17.14525	13.68838

Figura 1: Tabela de resultados.

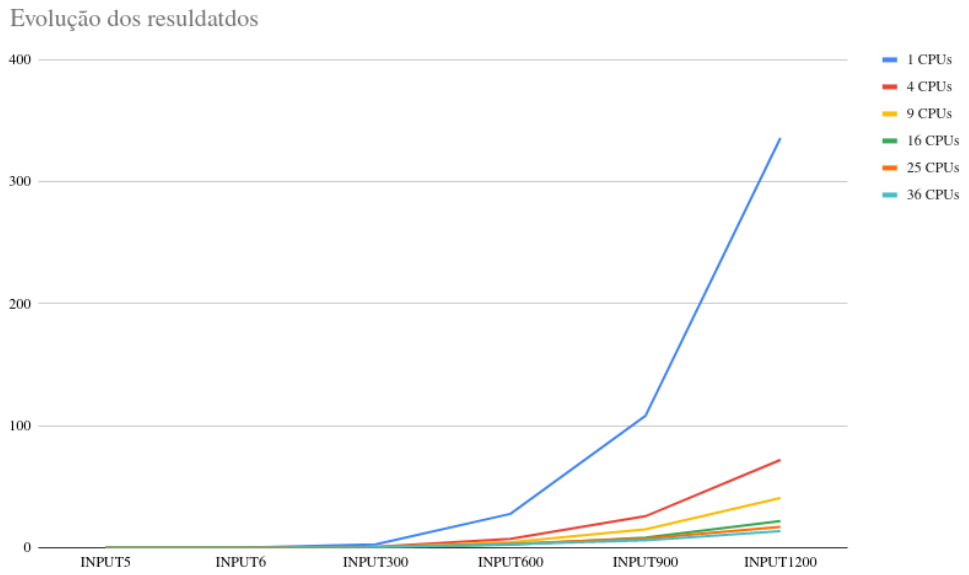

Figura 2: Evolução dos resultados em vários CPUs.

3.3 Observações

Podemos observar pela tabela e com melhor visualização atravás do gráfico da figura 2 que para *inputs* de matrizes de maiores dimenssões, o aumento de número de processadores pode causar uma maior eficência na execução do problema.

4 Conclusão

Em conclusão, acredita-se que este projecto foi um sucesso, visto ter se ultrapassado vários problemas inerentes à implementação, atingindo com êxito a meta final à qual nos tínha-se proposto. Infelizmente não se conseguiu usar métodos de *non-blocking* do *MPI* visto que não consegui-se resolver os problemas associados aos mesmos.

Adicionalmente, foi possível obter uma breve e interessante ideia da computação paralela e explorar algumas das tecnologias da mesma através da linguagem de programação C com a biblioteca *MPI*.