Enunciado do Trabalho Prático

Este trabalho prático tem como principal objetivo a utilização das capacidades de programação paralela através do modelo de partilha de memória, para a resolução de um problema de cálculo computacional.

Descrição do Problema

Considere o problema de *Job-Shop*, que é a alocação de recursos para a concretização de um trabalho[1]. No problema de *Job-Shop* existem várias máquinas que conseguem realizar operações. Para se produzir um trabalho (*job*) é necessário realizar um conjunto de operações, por sequência, em várias máquinas. Como as máquinas não conseguem realizar todas as operações, cada trabalho é uma sequência de operações, em que cada operação tem de ser feita numa máquina específica.

Adicionalmente, porque cada trabalho é distinto dos restantes, a sequência de máquinas de cada trabalho pode não ser a mesma dos restantes trabalhos, i.e., cada trabalho tem a sua própria sequência de máquinas. Contudo, a ordem das máquinas para cada trabalho tem de ser respeitada, i.e., uma operação de um trabalho que deve ser executada numa máquina específica só pode começar quando a operação anterior desse mesmo trabalho já terminou; assim como a operação seguinte do trabalho só pode começar depois da operação actual terminar.

Finalmente, realizar uma operação numa máquina demora tempo. Cada operação de cada trabalho demora um tempo específico, que pode ser distinto para cada operação. Uma máquina só consegue fazer uma operação de cada vez, assim, quando começa uma operação, só fica livre para novas operações após esta terminar.

Identificam-se assim as duas restrições deste problema:

1) As operações de cada trabalho têm de ser realizadas por ordem. A segunda operação de um trabalho só pode começar depois da primeira operação do mesmo trabalho estar concluída, e assim sucessivamente para todas as operações seguintes do mesmo trabalho;
2) uma máquina só consegue executar uma operação de cada vez, pelo que havendo duas operações para a mesma máquina, uma só pode começar depois da outra ter terminado.

Uma característica do problema, que simplifica as restrições, é que um trabalho não depende dos restantes trabalhos, i.e., os trabalhos são independentes entre si. Assim sendo, as operações de um trabalho não interferem com as operações dos restantes trabalhos, desde que a máquina para onde estão atribuídas esteja livre.

[1] Ver: https://developers.google.com/optimization/scheduling/job_shop

Objectivo

O objectivo do problema é apresentar um escalonamento válido, isto é, definir um tempo de início para cada uma das operações (de todos os trabalhos), de modo a cumprir as restrições, i.e., uma operação só começa após a anterior ter terminado, e uma máquina só está ocupada com uma operação de cada vez.

Após encontrar uma solução válida, o segundo objectivo é optimizar (reduzir) o tempo de execução total do escalonamento, conhecido como Makespan. O tempo de execução total do escalonamento é definido como o tempo necessário para concluir todas as operações do plano de escalonamento.

Um problema pode ter várias soluções distintas que são todas válidas e que podem ter uma duração total do tempo de produção distinta. O pior caso possível pode ser definido como o tempo necessário para executar todas as operações, uma de cada vez, até estarem todas concluídas. O melhor tempo possível pode não ser fácil de encontrar; contudo sabe-se que não poderá nunca ser inferior ao tempo que demora a executar qualquer um dos trabalhos individualmente (porque uma operação não pode nunca começar antes da anterior).

A solução algorítmica deve apresentar um resultado melhor do que a "pior solução possível", mas que não tem de ser o melhor possível.

Dados de Entrada e Resultados

A aplicação recebe como entrada a quantidade de máquinas e de trabalhos, e uma tabela com a identificação das máquinas capazes de executarem cada uma das operações de cada trabalho assim como a sua duração. O formato desta entrada pode ser representado através de um ficheiro de texto com um formato específico[2], considere por exemplo a lista de Fisher and Thompson 6x6[3]. Ver Anexo.

A tabela seguinte apresenta um exemplo dos dados de entrada do problema apresentada na página web da aplicação OR-Tools da Google[4]. Os valores na tabela representam o par (<*máquina, tempo duração*>) para 3 trabalhos com 3 operações cada, em 3 máquinas (identificadores começam no 0).

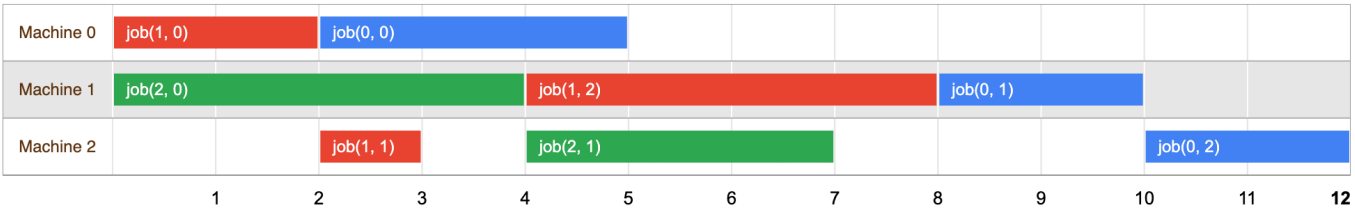
Jobs	**Operation1**	**Operation2**	**Operation3**
job0	(M0, 3)	(M1, 2)	(M2, 2)
job1	(M0, 2)	(M2, 1)	(M1, 4)
job2	(M1, 4)	(M2, 3)	*(M0, 1)*

[2] Ver: https://ptal.github.io/scheduling-data.html#job-shop-scheduling-problem
[3] Ver: https://github.com/ptal/kobe-rcpsp/blob/master/data/jssp/ft/ft06.jss
[4] Ver: https://developers.google.com/optimization/scheduling/job_shop

A aplicação deve produzir como resultado uma lista que identifica o instante de tempo de início de cada operação, para todas as operações de todos os trabalhos. Ver Anexo II. A tabela seguinte mostra o resultado do escalonamento, em que cada operação de cada job/trabalho tem um tempo de início.

Jobs	Operation1	Operation2	Operation3
job0	2	8	10
job1	0	2	4
job2	0	4	7

Esta lista pode ser agrupada por máquina (cf. imagem do sítio Google).



O resultado do escalonamento é uma lista das operações que serão executadas em cada máquina, com o tempo de início de cada uma. Adicionalmente deve ser indicado o tempo máximo de conclusão, no qual todas as operações estão concluídas. Na figura anterior, a última operação termina no tempo 12. O Anexo II mostra o formato de saída da aplicação.

Abordagem para Atribuição dos Tempos de Início

Existem várias possíveis abordagens para esta atribuição. O método de atribuição de tempos de início mais simples de usar é fazer uma travessia sequencial para todas as operações de todos os trabalhos, e atribuir um tempo de início a todas as operações, respeitando a sua duração, independentemente da máquina ou trabalho a que pertençam. Esta abordagem corresponde ao caso sequencial, que também é o mais longo possível, sem intervalos, em que se executa uma operação e apenas uma operação em cada instante de tempo.

Existem outras possíveis abordagens para a atribuição de tempos, válidas, desde que tenham em atenção o cumprimento das restrições do problema, como por exemplo *Branch & Bound* ou *Shifting Bottleneck*.

Uma abordagem que consegue encontrar a melhor solução óptima para o problema, isto é, a combinação que demora menos tempo de todas as possíveis, denomina-se *Branch & Bound*[5][6]. Esta abordagem constrói uma árvore de pesquisa, onde regista as várias soluções encontradas e o seu resultado. No final do algoritmo, é escolhido o nodo da árvore com a melhor solução, i.e., a que tem

[5] Ver Pinedo, 2012, *Job Shop Branch & Bound Slides*, https://web-static.stern.nyu.edu/om/faculty/pinedo/scheduling/shakhlevich/handout11.pdf
[6] Ver Brucker, 1994, *A branch and bound algorithm for the job-shop scheduling problem*, https://www.sciencedirect.com/science/article/pii/0166218X94902046

o tempo de conclusão da última operação mais baixo. Cada nodo da árvore representa uma potencial fonte de paralelismo, dado que os seus cálculos podem ser feitos de modo independente em relação aos cálculos dos restantes nodos da árvore.

Finalmente, foram publicados alguns trabalhos sobre implementações paralelas do método de *Branch & Bound*[7], algumas abordando o problema de job-shop[8][9]. Estes trabalhos podem ser considerados, mas não é obrigatório a sua implementação; devem ser considerados como referências.

Restrições na Implementação de uma Solução

A escolha do algoritmo que o grupo queira usar é livre. A única restrição que se requer para a implementação é a **não** utilização de apontadores nas estruturas dinâmicas. A implementação de grafos ou listas deve ser baseada em arrays (estáticos sem apontadores) e não listas ligadas ou equivalentes (com apontadores). Caso o grupo assim o entenda, pode usar um único apontador inicial para um array, mas não é permitida a utilização de apontadores internos na estrutura. Podem ser usados vários arrays.

Entrega

A entrega deste trabalho prático consistirá num relatório onde deve apresentar uma explicação para cada tópico seguinte, assim como um extrato do código relevante para esse mesmo tópico. O relatório deverá estar no formato PDF.

Todos os ficheiros (relatório + código-fonte das duas aplicações + ficheiros auxiliares) devem ser enviados num arquivo compactado do tipo **ZIP**, com a identificação dos elementos do grupo, número e nome.

A. Implementação Sequencial (5 vals)

1) Descreva o algoritmo escolhido para fazer a atribuição dos tempos de início das operações.
2) Apresente uma implementação sequencial da abordagem.
    - Esta implementação deve receber dois parâmetros na linha de comandos: um ficheiro com os dados de entrada; e um ficheiro para gravar o resultado.
    - A implementação deve cumprir as restrições do problema na solução gerada.

[7] Ver: Barreto & Bauer (2010), Parallel Branch and Bound Algorithm - A comparison between serial, OpenMP and MPI implementations
https://doi.org/10.1088/1742-6596/256/1/012018
[8] Ver: Perregaard & Clausen (1998), Parallel branch-and-bound methods for the job-shop scheduling problem,
https://link.springer.com/article/10.1023/A:1018903912673
[9] Ver: Steinhofel et al. (2002), Fast parallel heuristics for the job shop scheduling problem,
https://www.sciencedirect.com/science/article/pii/S0305054800000630

B. Implementação Paralela com Partilha de Memória (11 vals)

1) Seguindo a metodologia de Foster, apresente uma proposta de particionamento, uma divisão do problema em grupos de instruções e dados, tendo em conta a abordagem usada.
    - O algoritmo a usar deve ser identificado e ser preferencialmente o mesmo da implementação sequencial.
    - Identifique o grupo de instruções que será executado por cada uma das threads.
    - Descreva o padrão de comunicação usado entre as threads para a troca de informação.

2) Apresente uma implementação paralela usando partilha de memória da abordagem.
    - Esta implementação deve receber dois parâmetros na linha de comandos: um ficheiro com os dados de entrada; e um ficheiro para gravar o resultado.
    - Adicionalmente, deve receber como parâmetro de entrada o número de *threads*.
    - Compare e comente o resultado da simulação entre a versão paralela e a versão sequencial, nomeadamente os tempos de início das operações e o tempo de conclusão da última operação executada.
    - O relatório deve identificar:
        i. estrutura de dados usada no programa;
        ii. código inicial de arranque das threads do programa;
        iii. código executado pelas várias threads;
        iv. código final do programa.

3) Identifique as seguintes características do programa paralelo:
    - quais as variáveis globais partilhadas (só de leitura e de leitura/escrita), e as variáveis locais de cada *thread*;
    - as secções críticas (secções de código no programa paralelo), que podem criar situações de condição de corrida, e as variáveis envolvidas;
    - quais as técnicas de exclusão mútua utilizadas para garantir a correcção e determinismo do programa.

C. Análise do Desempenho (4 vals)

1) De modo a medir o desempenho das implementações anteriores, acrescente no código um mecanismo de medição do tempo de execução.
    - Durante a medição do tempo de execução não deve haver qualquer output ou input para a consola. Os dados podem ser obtidos por um ficheiro de texto externo.
    - O cálculo do tempo de execução na versão paralela deverá incluir a criação e término das *threads*.
    - A recolha do tempo de execução de cada aplicação deve ser uma média ou soma de várias execuções, configurável (por exemplo: 10 repetições), em vez de uma única execução.

2) Registe numa tabela o tempo de execução do programa, para os vários nºs de *threads* (assuma como exemplo: SEQ, PC1, PC2, PC4, PC8, PC16, PC32).
    - O ficheiro de entrada deve ser escolhido de modo a gerar uma carga de computação com pelo menos 1 minuto de execução na versão sequencial.
    - Use os mesmos dados de entrada para a versão concorrente. Compare as listas de resultados, nomeadamente, a ordem de execução das operações no tempo e o maior tempo de término das últimas operações.
    - Apresente um gráfico em que usa para o eixo dos XX o nº de threads e para o eixo dos YY o tempo de execução.

3) Compare o desempenho da execução concorrente com a execução sequencial, calculando o valor de *Speedup* (*S*) (rácio entre tempo de execução sequencial T1, e o tempo de execução paralela T_p para *p* threads).
    - Indique o nº de processadores presentes na máquina usada para os resultados.
    - Construa um gráfico com a evolução de *S* em função do valor de *p* (para os eixos YY e XX respectivamente).
    - Comente a variação do valor de *S* e apresente possíveis motivos.

Anexo I – Exemplos de ficheiros de entrada

Exemplo ficheiro: gg03.jss

```
3 3
0  3  1  2  2  2
0  2  2  1  1  4
1  4  2  3  0  1
```

Exemplo ficheiro: ft06.jss

```
6 6
2  1  0  3  1  6  3  7  5  3  4  6
1  8  2  5  4 10  5 10  0 10  3  4
2  5  3  4  5  8  0  9  1  1  4  7
1  5  0  5  2  5  3  3  4  8  5  9
2  9  1  3  4  5  5  4  0  3  3  1
1  3  3  3  5  9  0 10  4  4  2  1
```

Anexo II – Exemplo de ficheiro de saída

Exemplo de ficheiro saída: gg03.output para ficheiro entrada: gg03.jss

```
12
2  8 10
0  2  4
0  4  7
```