1. Implementação sequencial
2. Descreva a abordagem escolhida para fazer a atribuição dos tempos de início das operações.

## Implementação sequencial

A abordagem escolhida para atribuição de tempos de início das operações passou pela utilização de um conjunto de ciclos *for* e utilização de *structs.* As *structs* utilizadas podem ser visualizadas no excerto de código 1.



Código - Estruturadas de dados utilizadas

De modo a receber o número de *jobs* e de máquinas de um ficheiro de *input* recebido no primeiro argumento, após a invocação do script, foi desenvolvido o excerto de código apresentado no código 1.



Código - Leitura do número de jobs e máquinas

Após a leitura do número de jobs e de máquinas, era necessário, também, ler as operações que compunham os jobs. Para tal foi desenvolvido o excerto de código apresentado abaixo, no código 2.



Código - Leitura das operações dos jobs

Como referido anteriormente, foram criadas estruturas de dados para as máquinas, operações e conjuntos de operações (*jobs)*. Posto isto, inicialmente, foram inicializadas todas as máquinas como não-ocupadas e atribuído o seu identificador, utilizando um ciclo *for*.

Seguidamente, foi utilizado um *nested for* com três outros ciclos *for* dentro:

* O exterior iterava sobre as máquinas, que correspondem às linhas do ficheiro.
* O primeiro interior ciclo *for* é responsável pela gestão das operações. Começa por verificar se a máquina pretendida para a realização da operação atual (correspondente à iteração), está ocupada:
  + Caso não esteja, o tempo inicial do job é definido com o tempo máximo e o tempo de finalização da operação é equivalente ao tempo de início da operação atual mais o tempo necessário para finalizar a operação atual. Por fim, a máquina responsável pela operação é marcada como ocupada e o tempo até ao qual a máquina irá estar ocupada é equivalente ao tempo de finalização referido anteriormente.
  + Caso esteja, o tempo inicial da operação é equivalente ao tempo até ao qual a máquina estaria ocupada e o tempo final da operação seria equivalente ao tempo inicial da operação mais a respetiva duração. Por fim, a máquina é marcada como ocupada e o tempo até ao qual a máquina estaria ocupada é igual ao tempo de finalização da operação.
* O segundo ciclo *for* interior é responsável pelo *reset* das propriedades das máquinas, isto é, quando as operações de um certo *job* já foram definidas o estado das máquinas passa a livre e o tempo de finalização das mesmas também.
* O terceiro ciclo *for* interior é responsável por atualizar a variável que guarda o tempo máximo de finalização da operação. Esta variável é utilizada para definir o tempo de início de uma operação quando a máquina está livre.

No excerto de código 1, apresentado abaixo, está presente o código da abordagem descrita anteriormente.



Por fim, de modo a apresentar os resultados foi realizado um mero ciclo *for* que percorria as estruturas pretendidas e apresentava os seus valores. Além disso, como era requisito a criação de um ficheiro de *output* com os resultados, o segundo argumento após a chamada do *script* é o nome do ficheiro onde queremos guardar o resultado.



Código - Apresentação dos resultados

### Problema com a abordagem escolhida

A abordagem apresentada anteriormente tem alguns problemas nomeadamente o não respeito da ordem das tarefas, isto é, não há nada que garanta que a ordem de execução das operações vá ser respeitada. Embora a leitura e procura no ficheiro de input seja feita ordenadamente e a ordem, à partida, esteja garantida, este pode ser um problema a ter em consideração.

## Implementação Paralela

De modo a realizar a paralelização do problema foi utilizada a biblioteca OpenMP tendo em conta que a adaptação de aplicações sequenciais é relativamente fácil utilizando a mesma.

Durante o processo de conversão do problema de sequencial para paralelo tivemos alguns problemas, nomeadamente, de condição de corrida. Para combater os mesmos, utilizamoso *pragma critical* e o *pragma barrier*, mas nem assim foi possível resolver a condição de corrida.



Código - Implementação paralela inicial

De modo a dar a volta a este problema foi decidido que, por defeito, cada *thread* será responsável por um *job*. Caso o utilizador defina um número de *threads* em específico, esse mesmo número será utilizado. Para tal, deve simplesmente introduzir o número de *threads* no terceiro argumento aquando da invocação da *script*.



Código - Definição do número de threads

Relativamente à *thread* responsável por *job*, foi definida uma secção paralela em que o compilador tem conhecimento que o bloco de código subsequente deve ser executado em paralelo com as várias *threads*. Para definição desta secção foi tirado proveito do *pragma omp parallel* com especificação do número de *threads* a utilizar.

Tendo, então, a secção paralela criada, foi procedemos com a utilização da função omp\_get\_thread\_num(), que retorna o identificador da *thread* atual. Posto isto, foi criada uma variável que guarda este identificador que vai atuar como o número do *job.*

De seguida, mantivemos apenas o ciclo *for* que percorre todas as máquinas existentes e o que percorre os *jobs* foi substituído pelo código da figura 7, devido ao problema exposto anteriormente.



Código - Alteração do ciclo for exterior por omp parallel

Findada a execução do ciclo *for* que percorre as máquinas, é então invocada uma barreira, utilizando o *pragma omp barrier*, de modo a sincronizar as *threads*. De seguida, é realizado o *reset* ao estado das máquinas e é calculado o tempo máximo de finalização das operações. Tendo em conta que foi removido um ciclo *for* que percorria os *jobs*, este processo também sofreu uma alteração mínima. Em vez de ter dois ciclos *for*, passou a ter apenas um e recebe o valor da *thread* atual no seu lugar.



Código - Cálculo do tempo máximo de finalização da operaçao - paralelo

Parte 2

Variáveis globais partilhadas:

No excerto de código acima demostrado conseguimos identificar variáveis globais partilhadas:

• Machines – Representa uma Estrutura que armazena as informações sobre as máquinas como id, o estado (ocupada ou liberta) e o tempo que demorou a processar a informação.

• InitialTime - Representa uma matriz bidimensional que armazena os tempos iniciais das operações para cada job e máquina.

• EndTime - Representa uma matriz bidimensional os tempos de finalização das operações para cada job e máquina.

• MaxEndTime - Uma variável inteira que armazena o tempo máximo de término entre todas as operações.

Variáveis locais de cada thread:

• Row – Representa uma variável local que armazena o índice da linha (job) atribuída a thread.

Secções críticas:

A secção crítica acontece quando uma thread verifica a condição se a máquina está ocupada e atualiza os tempos iniciais e de finalização. Esta verificação e atualização ter de ser feita com precaução para evitar condições de corrida. O código #pragma omp critical é usado para criar uma seção crítica ao bloco de código que realiza operações de leitura e escrita. As variáveis envolvidas nessa seção crítica são:

machines[jobs[row].operations[m].machineId].isWorking: a variável isWorking da estrutura Machine que indica se uma máquina está ocupada ou livre.

machines[jobs[row].operations[m].machineId].endTime: a variável endTime da estrutura Machine que armazena o tempo que levou a realizar a última operação na máquina.

Técnicas de exclusão mútua:

A técnica de exclusão mútua utilizada para impedir que múltiplas threads executem secções críticas em simultâneo esta salvaguarda através do código #pragma omp critical. Este garante que apenas uma thread de cada vez pode realizar operações de leitura ou escrita nas variáveis. Isto evita condições de corrida e garante que os tempos finais e iniciais sejam atualizados corretamente.