# Projeto de Sistemas Operativos 2017-18
# `heatSim`
# Exercício 3

LEIC-A / LEIC-T / LETI
IST

**Resumo**

Este documento constitui um guia para a realização do terceiro exercício do projeto da disciplina de Sistemas Operativos. Este exercício consiste no desenvolvimento uma outra versão paralela, desta vez recorrendo ao paradigma da memória partilhada. Estenderá cada versão com uma condição de paragem dinâmica. Ambas as soluções (troca de mensagens e memória partilhada) poderão ser comparadas experimentalmente.

## 1 Introdução

Os alunos devem ler primeiro todos os guias anteriores.

No exercício 1 foi solicitado o desenvolvimento de uma versão paralela da propagação do calor usando troca de mensagens. Neste exercício farão a mesma coisa mas, desta vez, usando o paradigma de memória partilhada. O objetivo é os alunos poderem comparar as vantagens e desvantagens dos dois paradigmas, não só do ponto de vista do esforço de programação, mas também do desempenho do código resultante.

## 2 Paralelização Usando Memória Partilhada

Para resolver o exercício 3, os alunos deverão partir da solução sequencial desenvolvida nos laboratórios iniciais do semestre. A estrutura geral do código será bastante semelhante à do exercício 1: existirá uma tarefa principal que cria as tarefas trabalhadoras e lhes indica qual a fatia que devem calcular. Existem, no entanto, duas diferenças principais entre as duas soluções, que se descrevem de seguida.

A primeira diferença é que todas as tarefas podem aceder a variáveis globais e a memória alocada dinamicamente. Apenas as variáveis que são alocadas na pilha são privadas a cada tarefa. Desta forma é possível manter apenas duas matrizes (tal como na versão sequencial) e as tarefas trabalhadoras leem e escrevem diretamente nessas matrizes (nas fatias respetivas). Assim, evita-se a transferência explícita de informação entre tarefas no final de cada iteração. O mestre também não precisa enviar o conteúdo da matriz para as trabalhadoras, tem apenas de passar os ponteiros para as matrizes globais que são partilhadas por todas as tarefas.

A segunda diferença é que será necessário desenvolver um mecanismo de sincronização que impeça uma tarefa de começar a iteração $i+1$ antes das tarefas vizinhas acabarem a iteração $i$. [1] Este mecanismo de sincronização designa-se por uma *barreira*. Numa barreira, cada tarefa invoca uma primitiva designada por *esperar-por-todos*, bloqueando-se até que todas as outras tarefas invoquem essa mesma primitiva.

[1] Note-se que, na versão com troca de mensagens esta sincronização estava implícita, pois uma tarefa tinha de receber os valores dos seus vizinhos para continuar.

O número de tarefas que participa na barreira é conhecido à partida. Quando a última tarefa invoca *esperar-por-todos*, todos os processos bloqueados são libertados.

Apesar de cada tarefa trabalhadora só precisar dos resultados dos seus vizinhos para iniciar uma nova iteração, sugere-se que no final de cada iteração todas as trabalhadoras se sincronizem na mesma barreira. Isto é, nenhuma trabalhadora começa a iteração $i + 1$ enquanto existirem trabalhadoras ainda a calcular a iteração $i$.

Compete aos alunos implementarem este mecanismo. Para tal, devem recorrer exclusivamente ao suporte das *pthreads* para trincos (*mutex*) e variáveis de condição.

# 3 Terminação Dinâmica

Até agora, todas as versões executam um número de iterações fixo, definido quando o programa é lançado. O número de iterações depende, naturalmente, do tamanho da superfície e da diferença entre os valores iniciais e finais. No caso genérico pode ser difícil estimar quantas iterações são necessárias, e utilizar um número excessivamente alto de iterações é um desperdício de recursos.

Pretende-se que os alunos alterem o código para terminar automaticamente a simulação quando se verificar que, entre a iteração anterior e a atual, já não existem alterações significativas aos valores da matriz. Em cada iteração, cada tarefa trabalhadora deve calcular para cada ponto interior da sua fatia qual a diferença entre o valor anterior e o novo valor. Caso se verifique que, na mesma iteração, a maior diferença observada na matriz é inferior a um limiar de paragem `maxD`, então a simulação é dada como terminada, mesmo que o número máximo de iterações `iter` não tenha sido atingido. Se a condição de convergência não se verificar, a simulação deve terminar após o número máximo de iterações. Naturalmente, serão desvalorizadas soluções que limitem o paralelismo.

O valor de `maxD` é passado como argumento adicional (obrigatório) na linha de comandos quando o `heatSim` é lançado.

# 4 Experimente

Sugere-se que os alunos comparem o desempenho da solução com troca de mensagens e da solução baseada em memória partilhada para diversas dimensões da matriz.

# 5 Entrega e avaliação

Os alunos devem submeter um ficheiro no formato zip com o código fonte e o ficheiro *Makefile*. O exercício deve obrigatoriamente compilar e executar nos computadores dos laboratórios.

A data limite para a entrega do terceiro exercício é 13/11/2017 até às 23h59m. A submissão é feita através do Fénix.

Após a entrega, o corpo docente disponibilizará a respetiva solução, que pode ser usada pelos alunos para desenvolverem os exercícios seguintes.