SISTEMAS OPERATIVOS

PROJETO 3: IMPLEMENTAÇÃO, COM *THREADS*, DO *CONWAY'S GAME OF LIFE*

Realizado por:

M.ª Inês Coelho, 2004104311

Filipe Batista, 2006124931

CRIAÇÃO DO MUNDO

O projecto apresentado constitui uma implementação do *Conway's Game of Life. A* aplicação começa por pedir ao utilizador para definir a duração, em gerações, do jogo, o tamanho da grelha e a frequência de *snapshots*. Por questões de visualização optamos por limitar o tamanho da grelha entre 20 e 80 células. Posteriormente o utilizador pode definir as quantidades de cada padrão, entre os 5 disponíveis *(single, block, glider, LSS, pulsar)*, que quer colocar em jogo. Para garantir a colocação aleatória dos padrões na grelha, são colocados primeiro os de maior dimensão e o espaço disponível para colocar novos padrões em jogo diminui progressivamente: até metade da grelha pode ser ocupada pelo primeiro padrão a ser colocado em jogo e os restantes padrões podem ocupar até um quarto do espaço livre da grelha. Estas funções encontram-se definidas em *pattern.c*.

PROCESSAMENTO EM PARALELO

O processamento em paralelo é conseguido recorrendo a *threads*. Utilizam-se tantas *threads* quantas as necessárias para dividir a grelha de jogo em conjuntos de três linhas. Quando o tamanho da grelha não é múltiplo de 3, as linhas sobrantes dividem-se entre a primeira e a última *thread*: se sobrar apenas uma linha, a última *thread* fica com ela; se sobrarem 2 linhas, a última *thread* fica com a antepenúltima e a primeira *thread* fica com a última linha. O *array* de estruturas *worker_struct*, contém um *mutex* e uma variável de condição para cada *thread*.

Para calcular as várias gerações, cada *thread* começa por calcular a próxima geração das linhas centrais. De seguida, tenta aceder ao semáforo *mutex* da *thread* anterior e, nesse caso, verifica se a geração calculada é a mesma ou a anterior da que está a tentar calcular. Enquanto esta condição não se verificar, a *thread* espera pelo desbloqueamento da condição e, quando isto acontece, calcula a próxima geração da linha da esquerda. Posteriormente faz o mesmo para a linha da direita: tenta aceder ao mutex da *thread* seguinte, verifica se a sua geração é a mesma ou a anterior da que está a tentar calcular, faz um *wait* enquanto não for e, quando possível, calcula a próxima geração da linha da direita.

O cálculo das gerações é feito, alternadamente, para um dos dois *arrays* disponíveis, de acordo com a paridade da geração que se pretende calcular.

SNAPSHOTS

Para fazer os *snapshots* recorre-se a *memory mapped files*. Por cada *snapshot* é inicializado um novo ficheiro na pasta *snapshots* (que, se não existir no directório do programa, terá que ser criada), cujo nome

correspondente é *AnoMêsDia_HoraMinuto_GeraçãoNúmero*, sendo que a parte do nome correspondente à data e hora é criada no início do programa para ser comum a todos os *snapshots* da mesma execução. As funções necessárias ao manuseamento dos *memory mapped files* encontram-se em *files.c.*

Quando uma geração é múltipla da frequência de *snapshots* definida pelo utilizador, a *thread* executa a função memory_mapped_files(). A escrita em *memory mapped files* é sincronizada, recorrendo-se a um semáforo e a uma variável de condição: só ocorre escrita quando todas as *threads* estão na mesma geração. Após copiar toda a informação para o o *memory mapped files*, a área de memória é desmapeada e o ficheiro é fechado. A última geração e a geração 0 (a grelha após a criação do mundo) são também copiadas para um *snapshot*.

SPEEDUP

O tempo de execução do programa foi calculado fazendo a diferença do tempo obtido, através do método *gettimeofday()*, antes da criação das *threads* e após a sua terminação. Obtiveram-se os seguintes resultados de *speedup*, comparativamente com a versão sequencial fornecida:

Dimensão	Gerações	Versão Sequencial (ms)	Versão paralela (ms)	Speedup
1000	20	36,33	94,44	0,385
5000	20	151,43	420,05	0,36
10000	20	302,64	806,77	0,375
25000	20	737,64	2178,58	0,339
50000	20	1468,09	3907,68	0,376