1 Introdução

O problema proposto foi criar uma simulação de um mundo onde habitam lobos e esquilos, num terreno com árvores e gelo, e testar o tempo real que demora a processar todas as iterações necessárias, utilizando o paralelismo de memória partilhada. O seguimento temporal deste mundo é feito por intermédio de gerações, que são determinantes na sobrevivência dos animais. São as gerações e o tamanho do mundo que definem a complexidade do problema. Neste pequeno relatório apresentam-se a a decomposição usada na versão serial, para obter a versão paralela, como foi usado o load balancing e os resultados obtidos.

2 Decomposição

Na versão serial do problema decompôs-se o mundo, em cada iteração (leia-se geração), em dois sub-mundos na forma de jogo-das-damas. Cada sub-mundo foi processado à sua vez, sendo o Vermelho o primeiro. A ordem pelo qual cada sub-mundo era processado não alterava o tempo de execução. Também não era importante a ordem pela qual a matriz era processada (embora foi usado linhas e depois colunas). À semelhança da versão serial, a versão paralela também subdividiu o problema numa matriz do tipo jogo-das-damas, mas foi preciso ter cuidado especial com a actualização de células vizinhas. Para evitar conflictos foi utilizado um sistema de locks. A solução consistiu em criar um array de locks onde cada posição do array correspondia a uma célula da matriz. Com esta solução foi possível bloquear o acesso às células vizinhas daquela que estava a ser processada, para evitar actualizações indesejáveis por parte de outra thread.

3 Load Balancing

Para tentar distribuir, de forma mais eficiente, o trabalho (work) entre as threads implementou-se uma estratégia de scheduling do OpenMP. Após vários testes com as diversas opções de scheduling possíveis, sendo elas o static, dynamic e guided decidiu-se usar o static. Isto deve-se ao facto de o tamanho do tabuleiro ser conhecido. Posto isso, é mais eficiente ter a possibilidade de controlar a largura dos chunks (ao contrário do guided) e quantos deles são, à partida, atribuídos a cada thread. Para o chunk size decidiu-se utilizar metade da largura da matriz. Isto deveu-se ao facto de se ter obtido melhores resultados com este valor. Foi registado pelos testes feitos que o chunk size não devia ser muito pequeno pois devido à implementação de locks, nas posições à volta da célula que está a ser processada, as threads eram obrigadas a ficar à espera umas das outras, levando a situações de deadlock.

4 Resultados de Performance

Os testes para obter os resultados de performance foram realizados utilizados os ficheiros de input disponibilizados pelo corpo docente. Os testes foram executados nas versões paralela e serial do problema, sendo que na versão paralela foram usadas respectivamente 2, 4 e 8 threads para executar o programa. De seguida serão apresentados os resultados obtidos nos testes para os diversos inputs e serão apresentadas as devidas conclusões sobre os resultados esperados.
Nos testes realizados com os inputs onde a grelha possui um tamanho pequeno, em geral 10x10, os resultados obtidos mostram que o programa serial é mais eficiente do que a versão paralela, principalmente no caso em que a versão paralela é testada com mais do que com 2 threads,

fazendo com que o tempo de execução aumente consideravelmente, como é mostrado na tabela abaixo.

Number of Threads	Execution Time
1	0.00 s
2	0.00 s
4	0.01 s
8	0.02 s

Nos testes realizados com a grelha de dimensão 100x100, foram usadas apenas 1000 iterações para comparar as versões paralela e serial. Ao observar a tabela abaixo, podemos concluir que a versão em série ainda é mais eficiente do que a paralela, mas também podemos notar que o programa paralelo começa a tornar-se mais eficiente conforme o número de threads que executam o programa crescem.

Number of Threads	Execution Time
1	2.823 s
2	6.540 s
4	6.120 s
8	5.490 s

Por fim foram realizados testes com a grelha de dimensão 1000x1000, onde ambos os programas foram executado durante 10 gerações. Nesta situação o programa paralelo passou a ser mais eficiente do que a versão paralela, como se pode ver na tabela abaixo.

Number of Threads	Execution Time
1	5.726 s
2	4.760 s
4	4.250 s
8	4.500 s

5 Conclusão

Depois de serem efectuados os testes e de se analizar os resultados obtidos podemos concluir que a versão serial do programa é mais eficiente para uma grelha de tamanho reduzido e para testes com um número de iterações não muito grande. Já para o caso em que a dimensão da grelha é muito grande e o número de iterações do programa é elevado, a versão paralela do programa é mais eficiente, devido ao facto de a execução do programa ser distribuída por diversas threads.