**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.

**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 sub-dividiu 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.

**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.

**Resultados de performance**

–- inserir os resultados dos testes que o prof pôs (world\_10, world\_100 e world\_1000) –-