Relatório CPD 1ª entrega

Esta primeira entrega do projeto revelou-se bastante desafiante e complexa pois no decorrer do seu desenvolvimento, como já era esperado, deparámo-nos com uma série de problemas que foram de difícil resolução.

Problemas encontrados:

1. Conseguir gerir o decorrer das sub gerações fazendo com que o resultado final de cada geração fosse o esperado ;
2. Na versão paralela , conseguir ter sincronização nas threads para que estas corram independentemente umas das outras fazendo o trabalho em paralelo mas sem estragar ou ver variáveis com valores errados – evitar conflitos por ter threads concorrentes ;
3. Na versão paralela, conseguir que as threads a correr , na eventualidade de algumas estarem sobrecarregadas de trabalho não estarem outras idle à espera que estas terminem – evitar load imbalancing.

Solução adoptada para os problemas encontrados:

1. Para resolver o problema das sub gerações optámos por alocar duas matrizes inicialmente ambas com o mesmo conteúdo (old\_world e new\_world) . No decorrer de cada geração corremos a sub geração vermelha seguida da sub geração preta, e o que fazemos é em cada sub geração fazer verificações no old\_world e escritas no new\_world copiando o conteúdo de new para old cada vez que se faz alterações no new. Isto evita que o cálculo da posição destino seja atrapalhada pelo movimento de outros animais.
2. Para evitar conflitos entre threads concorrentes optámos por alocar uma matriz extra . Esta matriz tem em cada posição um lock que bloqueia threads que pretendam aceder àquela posição da matriz quando estiverem a tentar movimentar animais. Inicialmente usámos o pragma critical fazendo uma secção critica grande em torno de todas as instruções que alteravam as matrizes , mas optámos pela matriz de locks por esta solução apontada anteriormente ser muito limitante uma vez que só permitiria o acesso à matriz uma thread de cada vez transformando a versão paralela em sequencial.
3. Quanto ao load imbalancing, optámos por usar o scheduling por defeito (static) para todos os ciclos paralelizados excepto um ,porque para um mapa gigante gerado uniformemente todas as threads têm, em média, o mesmo número de condições a testar, demorando o mesmo tempo. O único ciclo no qual optámos por ter um scheduling diferente possui algumas caracteristicas que o distingue dos outros. Este ciclo tem uma maior probabilidade de para diferentes células, ter diferentes trabalhos influenciando o tempo que cada thread demora a processar cada célula, potenciando o load imbalancing. Usámos então schedule dynamic neste ciclo para evitar que threads que não tenham de eliminar nenhum lobo fiquem idle enquanto outras ainda estão a processar uma determinada linha.
4. Medimos a performance do nosso programa para um número de threads variável , nomeadamente 1 2 4 e 8 .

Obtivémos diferentes resultados:

* Versão sequêncial = 320.924 sec;
* Versão paralela com 2 threads = 231.340 sec --> S =1,387 ;
* Versão paralela com 4 threads = 194.100 sec --> S =1,653;
* Versão paralela com 8 threads = 202.257 sec --> S =1,587;

Em relação aos resultados obtidos, esperariamos que um computador com 8 cores correce a versão paralela com 8 threads mais rápidamente que a versão paralela com 4 threads. No entanto, esta diferença entre os dois resultados não é tão significativa quando comparado com o resultado da execução com 2 e 1 threads. No caso da execução com 2 threads, o resultado foi o esperado, mais rápida que a versão com uma thread e mais lenta que as versões com mais threads.

Para ambas as versões e todas as execuções obtemos output igual.

