TÉCNICO LISBOA

CPD

Wolves and Squirrels - Sequêncial e OMP

Tiago Soares	63421	MEEC
Rodrigo Lourenço	65946	MEIC
Daniel Gonçalves	68126	MEIC

1. Decomposição

O nosso algoritmo divide-se em duas fases i) identificação dos movimentos das entidades dinâmicas e ii) resolução de conflitos.

Na primeira fase, são calculados os movimentos de todas as entidades dinâmicas, movimentos esses que são armazenados numa lista de conflitos. Para além disso esta fase encontra-se ainda dividida em duas, através da implementação de um esquema *red-black* que permite dividir uma geração em duas sub-gerações independentes.

Na segunda fase, é processada a lista de conflitos de cada célula da matriz, de maneira a identificar, de todas as entidades que se deslocaram para essa célula, qual a final.

Todas as iterações (gerações) dependem da iteração anterior e a segunda fase depende a primeira fase. Assim a primeira fase é executada como duas secções paralelas, a *red* e a *black* e cada secção é paralelizada num *for*, esta paralelização deve ser sincronizada, pois podem existir múltiplas *threads* e escreverem na matriz de conflitos. As leituras por outro lado não precisam de ser sincronizadas, uma vez que a matriz mundo não é alterada nesta fase.

Uma vez terminada a primeira fase, então poderá ser executada a segunda. Esta foi paralelizada num *for* e como cada célula é independente das restantes, não foi necessário implementar qualquer mecanismo de sincronização.

2. Balanceamento de Carga

Em termos de balanceamento de carga, realizamos testes a dois tipos de *schedueling*, a *static* e a *dynamic*.

Balanceamento *static*, permite um menor *overhead* na distribuição de trabalho, no entanto não permite grande balanceamento de carga. Por outro lado, *dynamic* já permite um *melhor* balanceamento de carga, possuíndo no entanto um maior *overhead*, *overhead* esse que depende do tamanho do *chunk,* pelo que escolhemos um *chunk* que contemplasse as

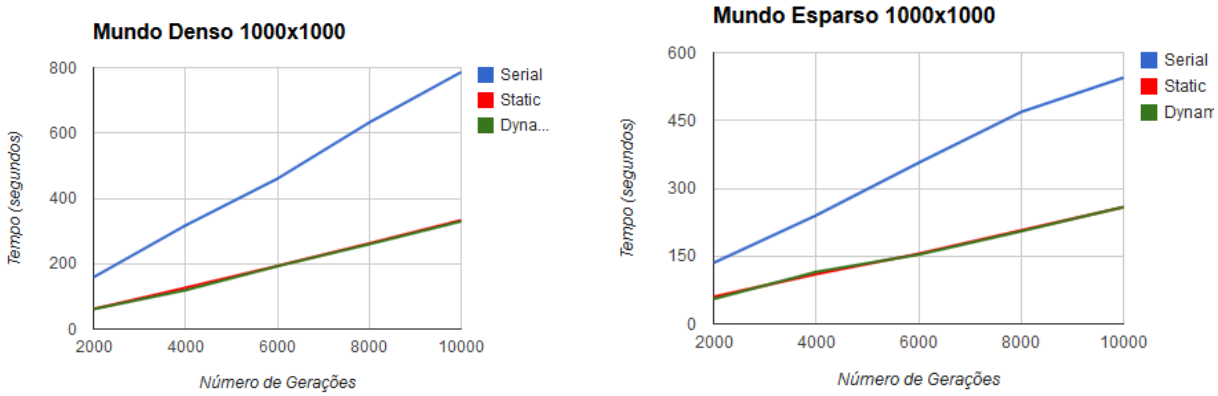
características do mundo e não fosse demasiado pequeno:

chunk = world_size / (proc * 2)

Consideramos que *schedueling dynamic* seria a escolha mais apropriada, uma vez que permite um melhor balanceamento de carga, principalmente para matrizes mais esparsas.

3. Testes de Sistema

Todos os sistemas foram realizados no *cluster* da RNL, num sistema Linux 3.8.12, com CPU E5_2650 @ 2.0GHz, com 8Gb de RAM. Foram realizados dois tipos de testes, mais extensivos, ambos a uma matriz 1000x1000, um denso e outro esparso.



Analisando os gráficos verifica-se que de facto as versões paralelas são muito mais eficientes que a sequêncial, executando o mesmo *input* em cerca de metade do tempo. Embora pelo gráfico não seja perceptível, num mundo mais esparso, o *schedueling dynamic* foi mais eficiente que o *static* embora apenas na ordem das unidades de segundo.

Em termos de speedup, os testes com a matriz de 1000x1000, revelaram que em média se verificou um speedup de aproximadamente 2.5 processadores, sendo o *speedup* do *dynamic*, ligeiramente superior ao do *static*.

Foram ainda realizados testes a outros tipos de matrizes, de média e baixas dimensões e tal como seria de esperar verifica-se que, para matrizes de pequenas dimensões a versão paralela não apresenta grandes acréscimos de performance relativamente à sequêncial, isto acontece, porque a paralelização dá-se ao nível da matriz e não da geração.

4. Notas

De forma a garantir determinismo em ambas as versões do projecto, alguns dos requisitos do projecto tiveram que ser contornados e certas assumpções tomadas. Nomeadamente:

1. Um lobo apenas procria na geração a seguir a “engravidar”.
2. Num conflito entre lobos, em que hava um esquilo na matriz, o conflito não é resolvido tendo em conta o *starvation period*, mas o *breeding period*, uma vez que todos assumem que comeram o esquilo.