TÉCNICO LISBOA

Computação Paralela Distribuida

MEE

Game of life - Open MP

Authors:

Afonso Oliveira 108271 afonso.p.oliveira@tecnico.ulisboa.pt
Duarte Diamantino 99139 duarte.antonio.diamantino@tecnico.ulisboa.pt
Gonçalo Cecílio 99144 goncalocecilio@tecnico.ulisboa.pt

2023/2024 – 2º Semester, P3

Conteúdo

1 Introdução

Neste relatório abordamos a utilização do *openMP*, tal como requerido no enunciado do projeto, implementado num "*Conways Game of life*" a partir de uma *seed* aleatória produzida através de *inputs* por argumento na execução do código. A lógica e implementação serie deste jogo foi já testada e aprovada anteriormente através de submissão. Para esta parte do projeto, iremos explorar a nossa implementação paralela do mesmo, assim como quais foram as preocupações em mente durante desenvolvimento.

2 Implementação

Como supracitado, realizamos a implementação paralela através do *openMP*, sendo o primeiro passo a escolha do segmento de código que devemos paralelizar. Como era de se esperar o paralelismo foi implementado neste projeto utilizando *openMP*. Para tal, temos primeiro de averiguar que parte do programa pode e deve ser paralelizado. Como mencionado na aula prática, usamos a ferramenta *Vtune* para nos ajudar nesta analise.

Esta ferramenta permite-nos fazer uma analise da utilização do processador em diferentes partes do programa, mais especificamente em que blocos e funções é que o processamento é mais demorado.

Function	Module	CPU Time	% of CPU Time
death_rule	world_gen_serial.exe	583.286s	58.6%
life_rule	world_gen_serial.exe	333.994s	33.6%
rules	world_gen_serial.exe	60.350s	6.1%
r4_uni	world_gen_serial.exe	11.447s	1.2%
gen_initial_grid	world_gen_serial.exe	4.899s	0.5%
[Others]	N/A*	1.273s	0.1%

Figura 1: Tempos de processamento por funções da implementação série

Tal como esperado por inspeção do código, pudemos averiguar através desta ferramenta que o processamento é mais demorado quer nas verificações das diferentes regras do jogo, quer na contagem do numero de células em cada espécie.

Abaixo mostramos a utilização do CPU na implementação serial, que na ausência de paralelismo mostra apenas a execução numa única thread.

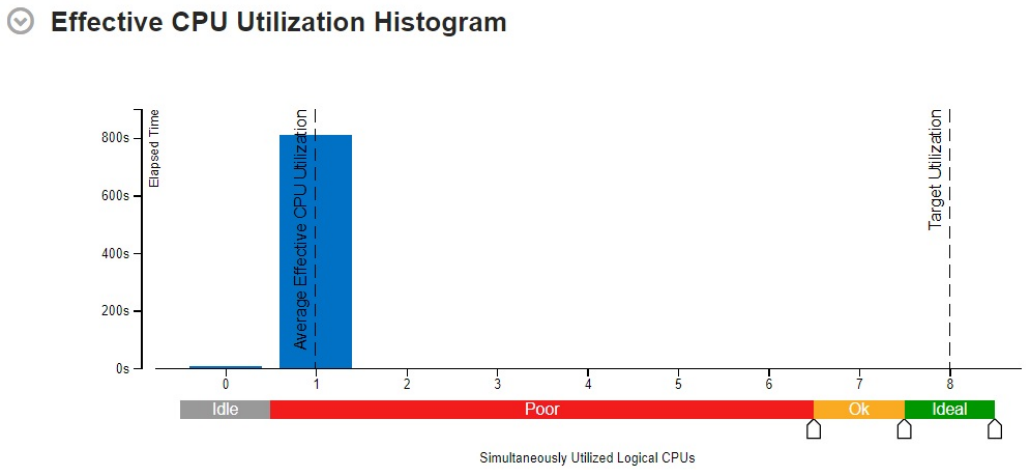

Figura 2: Utilização*threads* no serial

Desta forma, estamos prontos para proceder à implementação do *openMP*.

Pretendemos dividir o processamento da nossa grid entre várias*threads*, para acelerar significativamente o percorrimento da mesma. Temos já alguns possiveis problemas a apontar e escolhas a fazer. Em primeiro lugar, sendo que percorremos sempre todos os 3 eixos da grid, qual devemos paralelizar? A resposta a esta pergunta é que devemos paralelizar o eixo que percorremos menos vezes, ou seja, o eixo que é incrementado no primeiro *loop*"for". No nosso caso, este é o eixo do x.

Como estamos a dividir a *grid* entre diferentes camadas para paralelismo, cada processo paralelo têm também de ter acesso à propria variavel de contagem para percorrer as colunas e as linhas. Para tal, basta recorrer à primitiva private que, no nosso programa, é necessário para ambos eixo y e z `(private (aux\textunderscore y, aux\textunderscore z))`.

De seguida, encontramos ainda outro possível problema intrínseco à nossa implementação. Sempre que percorremos a *grid* e calculamos o estado de uma célula (morta ou a espécie à qual pertence), contamos o numero de cada espécie na nova geração que estamos a criar. Como esta contagem é agora feita entre diferentes processos, é necessário utilizarmos a primitiva *"reduction"*, primitiva esta que nos permite partilhar uma variável entre vários processos para cálculos recursivos. Por fim, utilizamos também a primitiva *schedule* (*dynamic*), o que aloca dinamicamente memória entre os diferentes processos à medida que esta é necessária, apesar de esta primitiva não ter alterado a duração do processamento de modo significativo.

```
#pragma omp parallel private (aux_y, aux_z)
{
    #pragma omp for reduction(+:count_species) schedule(dynamic)
    (...rest of for loops for searching...)
}
```

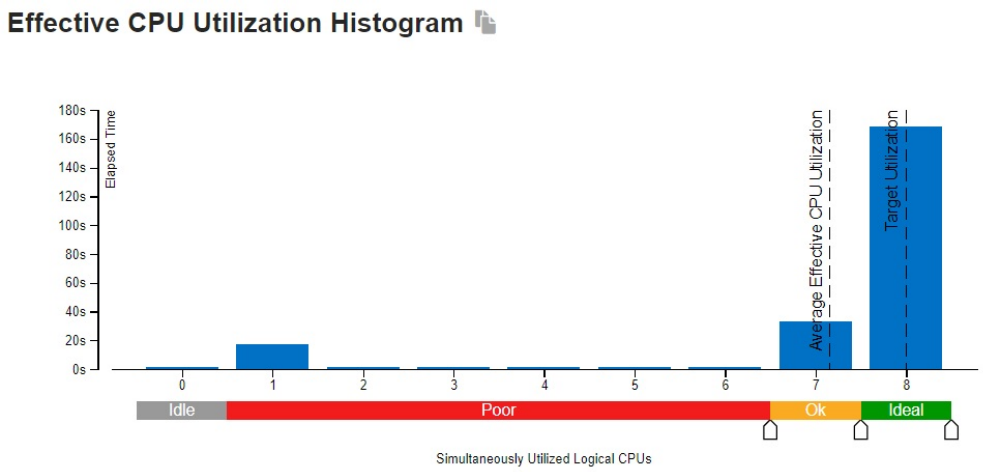

Figura 3: Utilização de*threads* em paralelismo no maior teste

O histograma da utilização do CPU acima mostra a utilização de cada *thread* para o teste corrido. Nesta implementação paralela podemos verificar que ,em antítese com o teste retirado da implementação em série, sãp utilizados mais unidades lógicas do CPU em simultâneo melhorando a *performance* da execução do código.

3 Conclusão / Resultados

Concluímos que existe uma efetiva melhoria numa implementação em paralelo quando comparada com a implementação em série. Todavia, para quantizar a melhoria referida, podemos proceder ao calculo do *Speedup*:

$$Speedup = \frac{T_s}{T_p} \tag{1}$$

Onde:

- T_s é o tempo de Execução Serie;
- T_p é o tempo de Execução Paralela;

Desta forma, utilizando o teste mais demorado como exemplo, podemos concluir que existe, de facto, um *Speedup* significativo. A relação entre o tempo de processamento e o número de *threads* utilizado é aproximadamente de 1:1, isto é, se utilizarmos 2 *threads*, o tempo de processamento desce para metade, e o *Speedup* é de, aproximadamente, 2. No entanto, esta relação é apenas verdadeira para números de *threads* iguais ou inferiores ao numero de cores físicos do processador do computador utilizado. Se declarar-mos o uso de 8 *threads*, por exemplo, isto implica a utilização de 4 *threads* lógicas, cujo processamento não equivale a cores físicos e, por esta razão, apesar de vermos ainda um decréscimo do tempo de processamento, este deixa de ser linear com o numero de *threads*.

Constatámos que os testes locais seriam inferiores em performance quando em comparação com o uso do *cluster*. Por conseguinte, iremos calcular os valores de *Speedup* quer analisando a execução do *cluster* tal como, analisando a execução local que produziu os resultados perambulares da figura 3

Serial Execution Time = 1083.9s		
Num_*threads*	Tempo Execução	*Speedup*
2	633.8s	1.7
3	444.2s	2.44
4	365.8s	2.96
8	260.4s	4.16

Tabela 1: Tempos de execução por número de *threads*-maquina local

Serial Execution Time = 996.2s		
Num_*threads*	Tempo Execução	*Speedup*
2	548.1s	1.82
4	304.3s	3.27
6	203.7s	4.89
8	191.1s	5.21

Tabela 2: Tempos de execução por número de *threads*-cluster

Concluindo, a implementação de paralelismo levou a tempos de execução diminuídos, tal como seria previsível.