Relatório do HW1 - CP

Diogo Domingues nº42950 e João Domingues nº41978

October 14, 2017

Abstract

Este relatório é destinado a uma análise profunda ao primeiro trabalho prático de CP onde são abordaos vários temas do mesmo. Para resolvermos o problema proposto que se destina à paralelização do código em C do jogo Othello fizemos uso do algoritmo `min_max` e paralelizámos o mesmo através da linguagem Cilk+.

1 Introduction

Durante este trabalho foi-nos pedido para paralelizar uma implementação de um programa em C que descreve a atuação do jogo Othello. Para além de alterar a implementação sequencial do programa em C para uma implementação em paralelo através do uso da linguagem Cilk+ o projeto foca-se na recolha e análise dos vários resultados obtidos. Achamos desafiante combinar a paralelização do programa com um algoritmo recurssivo, o que acaba por nos dificultar significativamente o trabalho. Ao longo do trabalho escolhemos implementar o algoritmo já paralelizado devido às dificuldades que acarretava primeiro desenhar o algoritmo e só depois pensar na sua paralelização. Durante o desenvolvimento de cada método, fomos testando o programa para termos a certeza que este estava a devolver o resultado esperado.

2 Approach

Este problema foi desenvolvido através da paralelização do código já implementado em C e da paralelização do código implementado por nós que resulta no algoritmo min-max e corte alpha-beta com negamax especificamente desenhado para o jogo Othello. Com a implementação do algoritmo min-max e corte alpha-beta tivemos de definir o numero de niveis da árvore min-max a pesquisar, que corresponde às jogadas futuras calculadas, antes de escolher a próxima "melhor" jogada de um determinado jogador. A utilização da variação do algoritmo min-max e corte alpha-beta garante uma melhoria na determinação da melhor jogada. O algoritmo permite o corte de vários nós durante a procura em árvore, aliviando assim o número das chamadas recurssivas. A eficiência em determinar quais os ramos que não devem ser explorados aumenta se os nós da árvore estiverem ordenados corretamente. Para o desenvolvimento deste algoritmo criámos uma struct "state" correspondente ao estado deste problema de min-max. Este estado contem um gameboard, um caracter para o turno e dois reducers `CILK_C_REDUCER_MAX` que servem para fazer de alpha e beta de cada estado (nó). Utilizando o código fornecido no `make_move` transformámos o for de fora num `cilf_for` para ser possivel ao cilk partir a execução do algoritmo no tabuleiro em linhas, paralelizando assim o código e baixando o tempo total de execução. Para podermos comparar o valor das varias possiveis jogadas com este `cilk_for`, e portanto, paralelamente, e para não termos de lidar com locks, que pioram bastante a eficiencia do programa, tivemos de utilizar o reducer `CILK_C_REDUCER_MAX_INDEX` e criámos um vetor de moves de forma a conseguir a melhor jogada. Ainda no `make_move` criámos o estado inicial, e dentro dos for's já referidos, quando é encontrada uma jogada possivel, a função `min_max` é chamada com o estado inicial, a jogada e a profundidade pretendida como input. No método `min_max` é criado o estado corrente e se a profundidade recebida pelo input for 0 retorna o resultado do jogo relativo ao jogador do estado corrente. Se não for este o caso é criado um estado com o alpha e beta inicializados a -beta e -alpha do estado pai respetivamente, implementando assim a tecnica negamax.Depois são executados dois for's para procurar as possiveis jogadas e quando encontradas, a cada uma, chama-se o `cilk_spawn` com a própria função para calcular o score desse filho. Comparamos os resultados dos vários filhos

com a ajuda de um reducer `CILK_C_REDUCER_MAX` e comparamos e atualizamos os alpha e beta com o auxilio dos dois reducers (iguais ao anterior) de cada estado.

Para paralelizar o algoritmo min-max e corte alpha-beta, utilizámos os seguintes padrões de controlo de paralelismo:

- Fork-join
- Reduction

3 Validation

A validação da solução e das várias fases da mesma foram feitas através da verificação dos resultados na consola. Para alcançar o código paralelizado de uma forma correta verificámos a escolha das jogadas por cada jogador e o desenrolar do jogo através de prints que devolviam o estado do jogo, i. e., o tabuleiro preenchido depois de cada jogada.

4 Evaluation

Os resultados foram gerados através da consola de comandos. Os resultados obtidos foram gerados a partir da máquina do departamento de informática sobre a qual nos foi facultado o acesso. Apartir da linha de comandos executá-mos testes que faziam variar o tamanho do tabuleiro entre 10x10, 20x20, 50x50 e 75x75, também faziam variar o numero de threads concurrentes entre 1, 2, 4, 8, 16 e 32 e finalmente o nivel de profundidade de procura no algoritmo min-max. Após a execução destets testes os resultados foram automáticamente colocados dentro de um ficheiro de texto e posteriormente recolhidos para uma ferramenta que nos facultou as imagens disponibilizadas.
Primeiramente faremos uma ánalise à imagem que descreve os resultados de uma execução com o parametro de níveis a pesquisar do algoritmo minimax com valor definido a 1.
Olhando para a Figura 1 podemos verificar que para o tabuleiros com tamanho de 10x10 o tempo de execução do programa diminui quando se acrescenta uma thread passando para 2 threads, no entanto à medida que as threads duplicam a partir daí até às 32 threads, o tempo de execução aumenta ligeiramente.
Relativamente à linha que define os resultados do tabuleiro 20x20 também podemos verificar que houve uma diminuição do tempo obtido ao aumentar inicialmente o número de threads de 1 para 2 e aumentando de 4 para 8 e de 8 para 16,contrariamente aquilo que acontece quando as threads são aumentadas de 2 para 4 e de 16 para 32.
No que toca aos grupos de pontos no gráfico que representam os tempos obtidos para os tabuleiros de 50x50 e 75x75 é notável que aumentando o numero de threads até 8, o tempo de execução do programa diminui. Para estes dois grupos, quando o numero de threads alcança as 16 e 32 threads, este valor temporal aumenta.
Com esta análise retiramos que para os testes executados para tabuleiros de menores dimensões e tempos relativamente baixos podem haver desvios significativos derivados da partilha da máquina com outros estudantes. Ao invés quando a carga executada na máquina é de maiores dimensões, verificá-mos que apartir das 16 threads, a nossa implementação induzia um resultado obtido com valores mais elevados.
Relativamente às Figuras 2 e 3 verificámos que para os grupos de resultados com os tabuleiros de dimensões 10x10 e 20x20, existe uma diminuição ligeira nos tempos obtidos à medida que se aumenta o número de threads concurrentes. Olhando agora para as linhas verde e azul, que correspondem respectivamente aos resultados dos tabuleiros de dimen sões 50x50 e 75x75, concluímos que segundo a nossa implementação, o tempo de execução do programa diminui até à utilização de 16 threads. Quando se ultrapassa esse valor e se alcança a execução do programa com 32 threads obtemos um valor temporal muito maior.

5 Conclusions

Concluímos que conseguimos implementar um jogador de othello (hard) e que tivemos resultados positivos relativos á paralelização, contudo, sentimos que podiamos ter ido mais além. Achamos que a implementação do padrão stencil no `get_heuristic` seria uma boa melhoria ao nosso programa.

Concluímos também que alguns dos valores retirados da execução de testes a partir da máquina do departamento podem estar significativamente alterados devido à partilha da máquina com outros utilizadores. Durante o desenvolvimento do programa e da execução dos testes de performance tivemos a oportunidade de verificar que aquando o aumento significativo do tabuleiro, os valores temporais obtidos são muito maiores, devido principalmente a um maior número de possibilidades de jogada.

6 Bibliography

[mit] http://supertech.lcs.mit.edu/cilk/lecture-3.pdf
[charts] https://www.onlinecharttool.com/
[cilkplus-reducers] https://www.cilkplus.org/docs/doxygen/include-dir/group___reducers_min_max.html
[stackexchange] https://tex.stackexchange.com/questions/

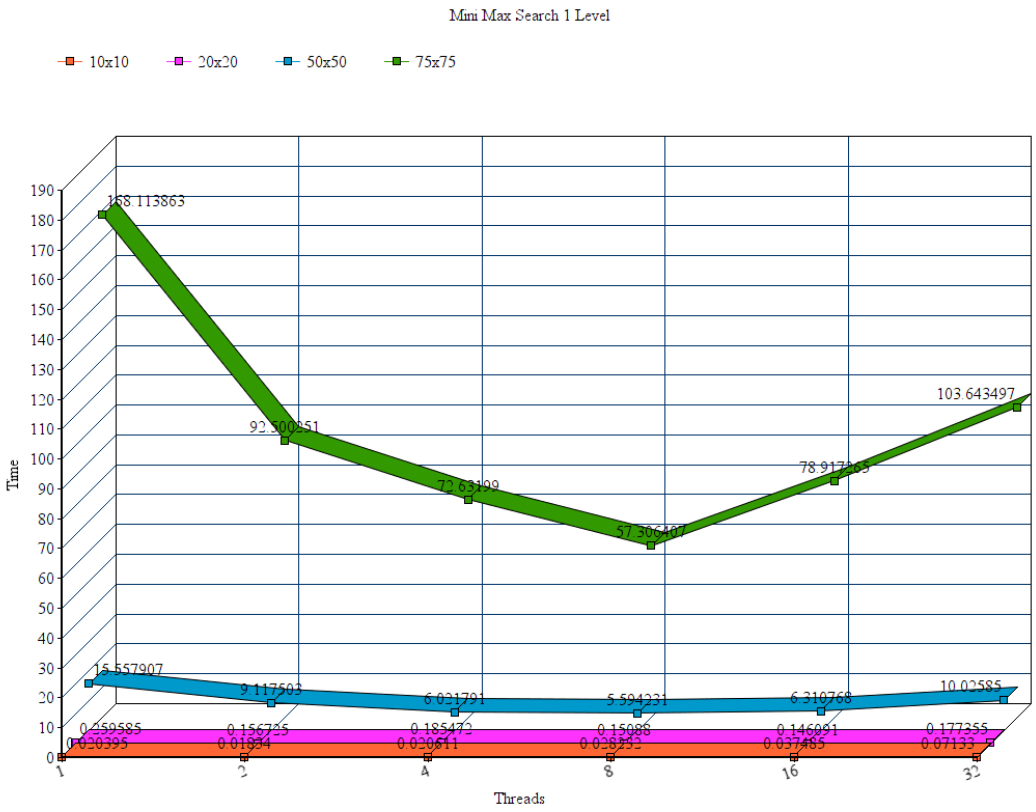

Figure 1: Esta imagem representa o gráfico dos resultados obtidos para uma procura de 1 nível.

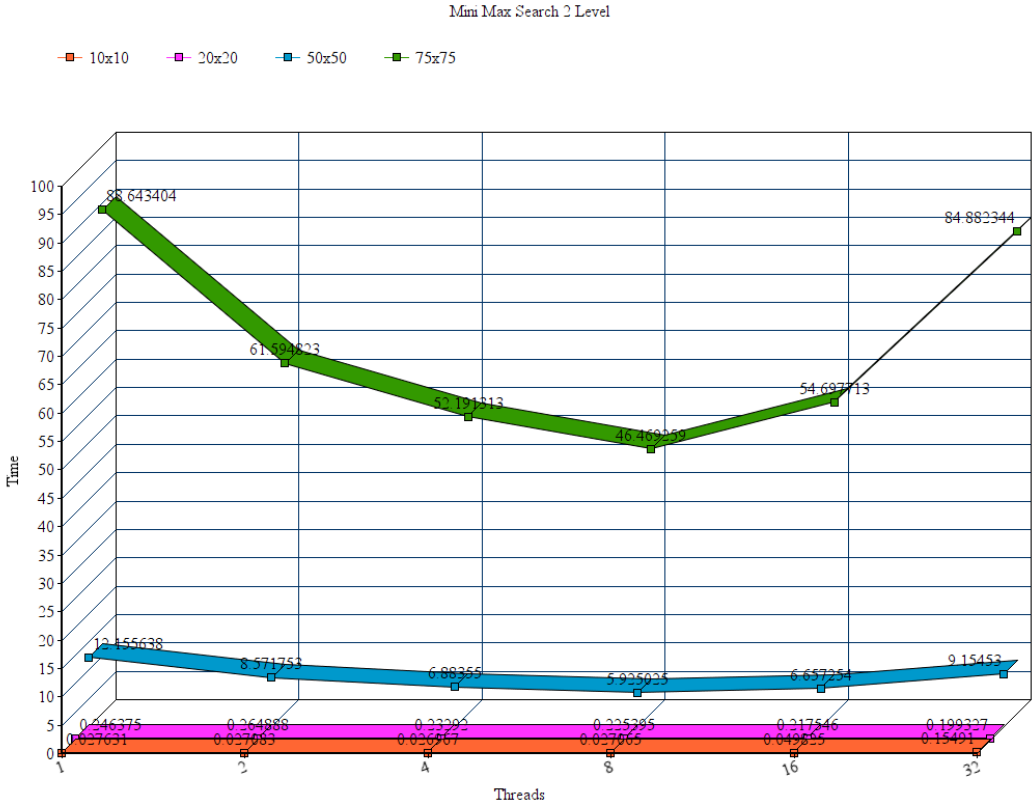

Figure 2: Esta imagem representa o gráfico dos resultados obtidos para uma procura de 2 níveis.

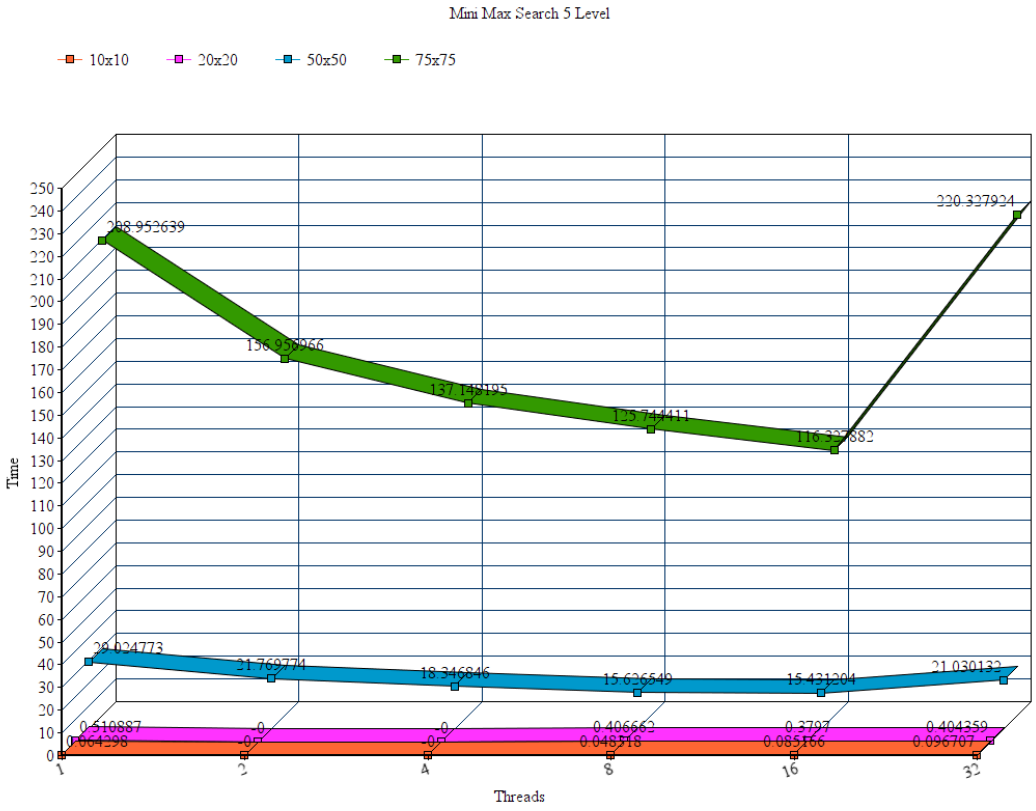

Figure 3: Esta imagem representa o gráfico dos resultados obtidos para uma procura de 5 níveis.