Instituto Superior Técnico

Computação Paralela e Distribuída

Prof. Luís Guerra e Silva

Relatório Parte I

Sudoku Paralelo

Name:	*Number:*
Manel Serra	81063
José Vieira	90900
Pedro Carmo	90989

8 de Abril de 2018

Abordagem

Para a realização deste projeto foi utilizada a estratégia de busca em profundidade utilizando um algoritmo que se assemelha com uma *Deep First Search* (DFS). Sendo assim, o algoritmo em série é composto por três fases principais, a de verificação se o valor a introduzir é válido, a de progredir até à próxima posição livre e a de retroceder para a última posição alterada, seja detetado um erro.

Relativamente à implementação do paralelismo, pretende-se que cada processo implemente o mesmo algoritmo que em série, mas iniciando a sua pesquisa em pontos diferentes da árvore de procura. Quando o espaço de procura do processo se esgota, este vai buscar uma zona ainda não explorada à árvore de procura dos outros processos. Se algum dos processos encontrar uma solução, param todos de procurar. Para o caso de não existir solução, todos os processos acabam eventualmente de correr.

Em ambos os contextos são utilizadas máscaras de bits para identificar os valores que são ou não válidos em cada linha, coluna e casa. Por exemplo, num sudoku de 4x4, para testar se o número 3 é valido numa dada linha com máscara 1010, é feita a conversão de 3 para a sua representação em máscara (0100) e feito o bitwise OR de ambos os valores, resultando 1110. Uma vez que o resultado é diferente da máscara inicial seria seguro inserir o 3 nesta linha.

1.1 Decomposição

No seguimento da ideia apresentada, foi possível identificar dois estados distintos na execução de cada processo. Um primeiro, onde o processo efetua a procura em todo o espaço de procura associado a esta combinação posição-valor, e um segundo, onde o processo vai procurar trabalho por realizar, publicado por outros processos, e efetua a procura do mesmo modo que no primeiro estado.

De notar que, em todo o processo de procura, os processos vão publicando numa lista partilhada todos os valores válidos para cada posição que visitam, exceto a possibilidade que eles próprios vão efetuar.

1.2 Sincronização

Uma vez que os processos utilizam variáveis partilhadas , foi necessário introduzir a leitura e escrita das mesmas em regiões críticas, de forma a prevenir acessos simultâneos. A lista de trabalhos publicados, a matriz do sudoku e as matrizes das

máscaras são as variáveis partilhadas sujeitas a alterações por parte de diferentes processos. Relativamente ao ciclo *for* do primeiro estado, é utilizada a flag *nowait*, que retira a necessidade de sincronizar os processos aquando a finalização do do ciclo, permitindo-os prosseguir de forma independente para o segundo estado.

1.3 *Load Balancing*

A primeira decisão efetuada para equilibrar a carga de cada processo foi na questão da separação da árvore da DFS. Dado que a lista implementada para publicar/buscar trabalhos segue um comportamento *First In First Out*, a separação da árvore ocorre dos níveis superiores para os inferiores, uma vez que o trabalho dos níveis superiores fui publicado primeiro na lista. Isto permite que cada processo tenha um espaço de procura igual, bem como uma redução na entrada de zonas críticas para ir buscar trabalho.

Outros aspetos relevantes é o escalonamento dinâmico (*dynamic schedulling*) em *chunks* unitárias, o que permite distribuir cada iteração do ciclo *for* do primeiro estado pelos vários processos, à medida que estes se encontram disponíveis.

Resultados

Quando medidos os tempos de execução para *sudokus* de 4x4, 9x9, 9x9 sem solução e 16x16, obtém-se o gráfico da figura 2. Para o caso de 16x16, é importante salientar que os *puzzles* não foram resolvidos em tempo útil nas versões *serial*, paralela com um e dois processos e para o caso da matriz 4x4, os valores dos tempos são relativamente baixos e bastante próximos.

Pode-se observar então que a versão *serial* apresenta resultados parecidos à versão paralela com um processo, bem como a melhoria no tempo de execução destas para as versões com mais processos, como seria expectável.

Note-se também que, da versão paralela com 4 processos para a versão com 8 processos, existe um aumento de tempo de execução significativo. Isto pode ser causado pelos *overheads* introduzidos pelas secções críticas, sendo que quanto mais processos existirem, maior será a necessidade de utilização da lista, bem como a utilização da matriz do *sudoku*.

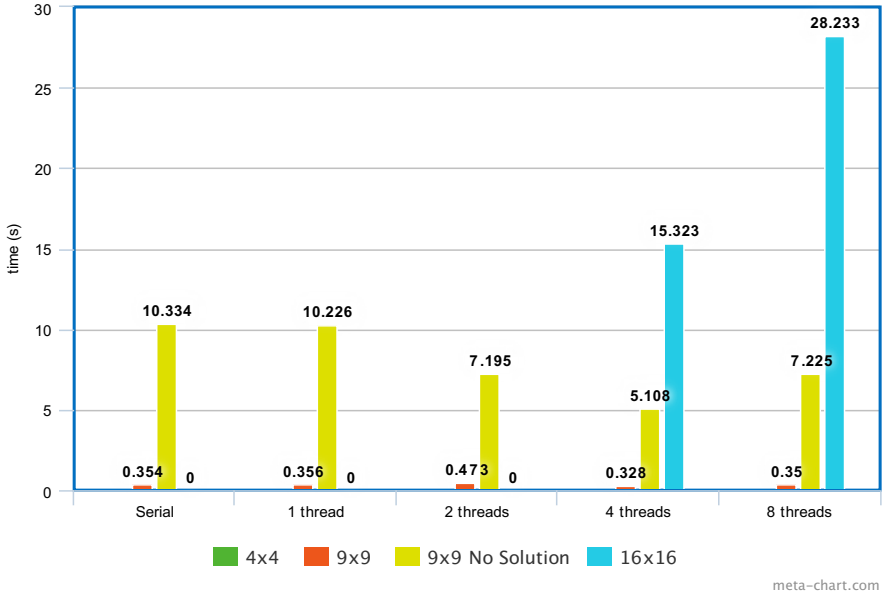

Figura 2.1: Medições do tempo de execução em função do número, para diferentes *puzzles*

Limitações

Apesar do programa fornecer resultados válidos, pontualmente poderá não funcionar de forma adequada, podendo ficar preso na sua execução. Isto deve-se ao facto de haver situações em que o algoritmo não consegue testar todos os casos. Isto acontece normalmente a partir das matrizes de 16x16, inclusive. Este erro deve ser então corrigido, também para poder testar a eficiência em matrizes de tamanhos maiores.