Universidade do Minho

Mestrado em Engenharia Informática

Algoritmos Paralelos - 2014/2015

Odd-Even Sort Paralelizado

17 de Março de 2015

Fábio Gomes pg27752

Índice

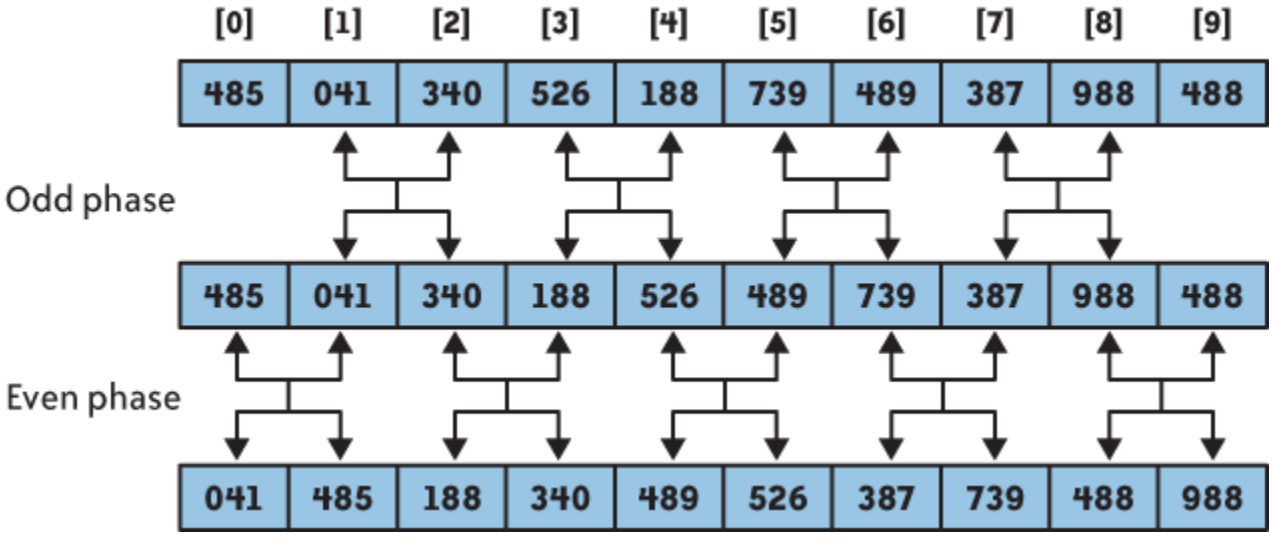
Introdução

O problema que nos foi apresentado está relacionado com o algoritmo de ordenação `Odd-Even Sort` que se baseia nas posições Ímpares e Pares. É nosso trabalho tentar paraleliza-lo.

Explicação do Problema

O *Odd-Even Sort* foi desenvolvido para tirar partido do paralelismo e das interconexões. O seu funcionamento é reduzido a 2 iterações que passam por comparar os índices Ímpares e os Pares com o elemento seguinte até que não sejam precisas mais trocas ficando ordenado.

O esquema seguinte explica o seu funcionamento.



Exemplo para um array de 10 elementos - 1

Análise do Código Fornecido

Este é o método `OESort`, o principal do algoritmo. Enquanto hajam mudanças o `exch` é 1 e a cada passagem no while é feita a troca do índice de paridade inicial (`start`).

```
void OESort(int NN, int *A)
{
    int exch = 1, start = 0, i;
    int temp;

    while (exch || start) {
        exch = 0;
        for (i = start; i < NN-1; i+=2) {
            if (A[i] > A[i+1]) {
                temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                exch = 1;
            }
        }
        if (start == 0) start = 1;
        else start = 0;
    }
}
```

Paralelização com OpenMP – Versão 1

A Paralelização passou por criar uma primeira versão (v1) em que o exch agora passa a ter o número de threads e em cada iteração é decrementado 1 por cada thread. Se alguma alteração se realizar o `exch` passa para o número de threads para que na próxima iteração do while a condição de paragem falhe e o programa continue a tentar ordenar. A barreira depois de decrementar o `exch` tem que estar presente pois pode acontecer o caso em que o programa nunca mais termina pois o exch é alterado quando é feita uma alteração e ainda há uma thread atrasada a fazer o `exch--`. O `pragma omp single` no fim faz barreira e troca a paridade, assim no início do ciclo seguinte todas as threads sabem que índice pegar.

```
void OESort(int NN, int *A){
    int exch = 1, start = 0, i;
    #pragma omp parallel
    {int temp;
     exch = omp_get_num_threads();
    while (1) {
        if(exch <= 0 && start == 0) break;
        #pragma omp critical
        exch--;
        #pragma omp barrier
        #pragma omp for
        for (i = start; i < NN-1; i+=2) {
            if (A[i] > A[i+1]) {
                temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                #pragma omp critical
                exch = omp_get_num_threads();}
        }
        #pragma omp single
        if (start == 0) start = 1;
        else start = 0;
     }}}
```

Paralelização com OpenMP – Versão 2

Esta Paralelização tem como objetivo reduzir o *overhead* causado pelo contínuo *sleep/wake* das threads. Ao fazer unroll aos 2 ciclos for que são indiretamente executados devido à paridade do algoritmo conseguimos fazer mais trabalho aproveitando o tempo em que a thread está ativa e eventualmente melhor performance.

São necessárias variáveis extra, `exch0` referente aos exchanges/trocas do primeiro ciclo for, a `exch1` para o segundo e a `first` que apenas serve para indicar o segundo ciclo que é a primeira iteração de trocas e se não existirem mudanças para não estranhar e tentar. Se fizer alguma alteração no primeiro ciclo `exch0` é posto a 1, baseado nisso o segundo ciclo apenas é executado se de facto houve alterações porque o `exch0` é 1. Se não trocarmos em nenhum dos ciclos o while acaba e damos por terminada a troca.

Esta versão é caracterizada por estar sempre a abrir e a fechar a zona paralela a cada iteração do ciclo while.

```
void OESort_v2(int NN, int *A)
{
    int exch0, exch1=1, i,first=0;
    while (exch1) {
        exch0=0;exch1=0;
        #pragma omp parallel
           {int temp;
            #pragma omp for private(i,temp)
            for (i = 0; i < NN-1; i+=2) {
                if (A[i] > A[i+1]) {
                    temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                    exch0=1;
                }
            }
            if(exch0 || !first){
            #pragma omp for private(i,temp)
            for (i = 1; i < NN-1; i+=2) {
                if (A[i] > A[i+1]) {
                    temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                    exch1=1;
                }
            }
               }}
          first=1;
    }}
```

Paralelização com OpenMP – Versão 3

Como a versão anterior pode trazer problemas devido à tal abertura e fecho da zona paralela, fiz uma alteração em que só se abre uma zona, fora do ciclo while, mas leva com 2 barreiras para sincronização. Esta mudança poderá trazer vantagens se os tempos de espera devido à barreira não forem tão altos em comparação à versão anterior.

```
void OESort_v3(int NN, int *A)
{
    int exch0, exch1=1, i,first=0;
    #pragma omp parallel
    {int temp;
    while (exch1) {
        #pragma omp barrier
        exch0=0;exch1=0;
        #pragma omp barrier

        #pragma omp for private(i,temp)
        for (i = 0; i < NN-1; i+=2) {
            if (A[i] > A[i+1]) {
                temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                exch0=1;
            }
        }
        if(exch0 || !first){
        #pragma omp for private(i,temp)
        for (i = 1; i < NN-1; i+=2) {
            if (A[i] > A[i+1]) {
                temp = A[i]; A[i] = A[i+1]; A[i+1] = temp;
                exch1=1;
            }
        }
           }
        first=1;
    }}
}
```

Testes e Análise de Resultados

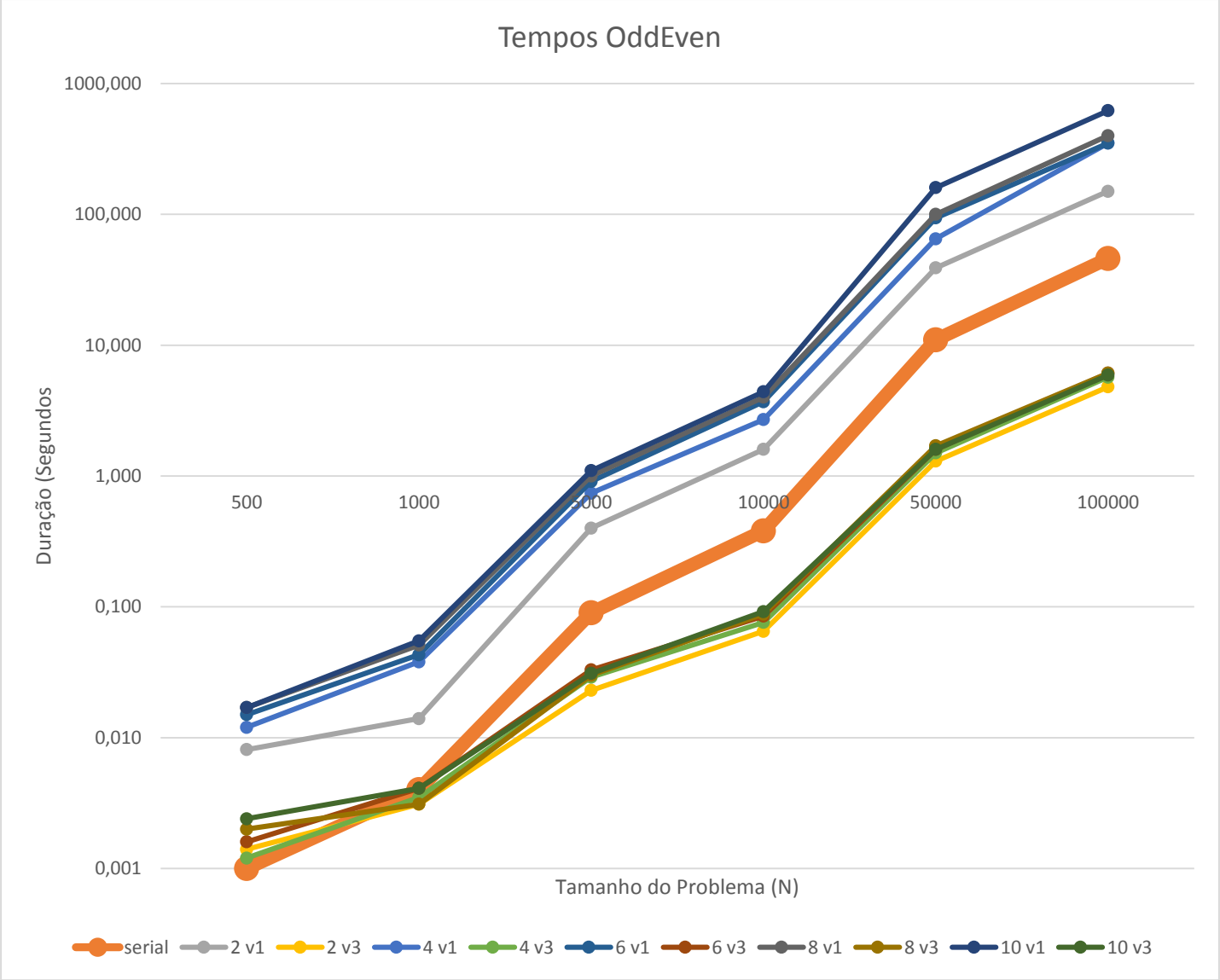
Com tudo concluído, é tempo de fazer medições de tempos de execução e analisar os resultados.

Os testes foram feitos para Dimensões (Size) e número de threads realizados para os 3 tipos de versões. Foram compilados com flag -O2. Os resultados são os seguintes apresentados na tabela.

	OddEven															
	SER	OpenMP														
		2 Threads			4 Threads			6 Threads			8 Threads			10 Threads		
Size	serial	v1	v2	v3	v1	v2	v3	v1	v2	v3	v1	v2	v3	v1	v2	v3
500	0,001	0,008	0,001	0,001	0,012	0,001	0,001	0,015	0,002	0,002	0,017	0,002	0,002	0,017	0,002	0,002
1000	0,004	0,014	0,003	0,003	0,038	0,003	0,004	0,043	0,004	0,004	0,051	0,004	0,003	0,055	0,004	0,004
5000	0,090	0,400	0,023	0,023	0,740	0,028	0,029	0,910	0,028	0,033	1,000	0,033	0,030	1,100	0,032	0,031
10000	0,380	1,600	0,061	0,065	2,700	0,075	0,076	3,700	0,085	0,085	4,000	0,087	0,088	4,400	0,087	0,092
50000	11,0	39,0	1,3	1,3	65,0	1,5	1,5	94,0	1,6	1,6	100,0	1,7	1,7	160,0	1,7	1,6
100000	46,0	150,0	4,3	4,8	350,0	5,7	5,7	350,0	6,1	6,1	400,0	6,2	6,1	620,0	6,1	5,9

Continuando o raciocínio da alínea anterior, em que se queria comparar as versões 2 (abertura e fecho da zona paralela) e 3 (uma zona paralela apenas) que possuem unroll do ciclo for, podemos agora fazer a sua comparação. Como era previsto as diferenças não seriam muitas, conseguindo a versão 3 ficar mais rápida com o aumento de threads e a v2 a diminuir um pouco em comparação com a v3 mas ambas ficam com tempos muito bons. Assim, para problemas de pequena dimensão e recursos recomendo a v2.

Fiquei muito impressionado com os valores obtidos para a v1, sempre conseguiu fazer pior tempo que a serial, a razão direta é devido a todas as barreiras (implícitas ou não) que ele tem conseguindo estar muito tempo nessas zonas. Tal problema foi resolvido com as 2 versões lançadas depois.



Neste gráfico temos uma comparação mais visual dos tempos (foram excluídos os tamanhos 50000 e 100000 pois o tempo da v1 era muito alto e impossibilitava a leitura do mesmo) da versão serial com as diferentes variâncias de threads com a v3 (como os tempos eram muito semelhantes não era necessário ter as duas no gráfico pois iam sobrepor-se frequentemente) e o eixo da Duração em escala logarítmica.

Analisando-o é fácil notar as 4 linhas acima da versão Serial (mais grossa) pois é a versão pior. Depois a ascensão da performance da versão com as duas fases de ciclo a ser melhor que a original com a diferença de threads a não ser muito determinante para estes tamanhos curtos no gráfico.

Conclusão

Para além de ser um caso prático e cativar por si só foi também um trabalho muito bom na medida em que permitiu usar novos conhecimentos adquiridos nesta Unidade Curricular que não foram abordados anteriormente. Um trabalho que apesar de parecer simples engloba em si vários aspectos que tiveram de ser devidamente considerados para que o algoritmo funcionasse corretamente.

É sempre bom conseguir uma versão melhor que a Serial e tal foi obtida usando o unroll do ciclo for em 2 ciclos para tirar partido das threads e diminuir barreiras que são talvez dos fatores que mais tempo fazem perder em versões paralelas. Com esta implementação o número de saídas e entradas da região paralela foi reduzida para metade (v2) e o início dos ciclos for foi definido previamente facilitando o compilador e reduzindo o `pragma omp single` que apenas trocava a variável start. Esta redução de sincronização de threads provou valer a pena ter feito mais código do que inicialmente facultado ficando até mais simples de entender o que faz em relação à versão 1, ou mesmo à serial.