FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Análise e Implementação do algoritmo "The Sieve of Erastosthenes"

(" O Crivo de Eratóstenes")

Mestrado Integrado em Engenharia Informática e Computação
Unidade Curricular de Computação Paralela
Ano Letivo 2015/2016

João António Custódio Soares 201206052
João Alberto Trigo de Bordalo Morais 201208217
25/05/2016

Índice

Descrição do Problema

Na unidade curricular de Computação Paralela foi proposto estudar o impacto de diferentes versões de paralelismo na performance do algoritmo "The Sieve of Erastosthenes" em grandes volumes de dados (dimensão entre 2^{25} e 2^{32}). Este algoritmo permite determinar os números primos no intervalo $[2, ?]$, sendo $? \in ? \backslash \{1\}$, conjunto de todos os números naturais excluindo valor '1'. Um determinado número é primo se e só se for divisível por '1' e por ele mesmo exclusivamente.
Neste sentido, relativamente ao impacto dos diferentes tipos de paralelismo, será avaliada a eficiência deste algoritmo para paralelismo com memória partilhada; paralelismo com memória distribuída; e ainda uma versão híbrida, isto é, paralelismo com memória distribuída e partilhada.
Para realizar o algoritmo de forma paralela com memória partilhada será utilizado a API do OpenMP e para a forma paralela com memória distribuída será utilizada a biblioteca do MPI.

Solução sequencial

O algoritmo é um método simples e prático de obter os números primos até um alcance definido **n** e tem complexidade temporal $O(n \log \log n)$. A simplicidade do código faz com que este seja usado como *benchmark* para comparar performance de compiladores e *chips*. Para a análise pretendida será implementada uma versão sequencial do algoritmo, uma versão paralela, num sistema de memória partilhada utilizando OpenMp, e duas versões paralelas utilizando MPI, num sistema de memória distribuída e num sistema de memória partilhada.

O algoritmo consiste em:

1- criar uma lista de números naturais não marcados de 2 até **n**;
2- determinar o valor de k como o menor número da lista não marcado;
3- marcar todos os números múltiplos de k entre k^2 e **n** da lista;
4- voltar a passo 2 até que $k^2 > n$.
Todos os números não marcados na lista são primos

Pseudocódigo:

```
   for (i = 0;(i+2)*(i+2) <= n; i++)
  {
      if (primes[i] != false)
      {
          for (j =(i+2)*(i+2) - 2; j < n; j += i + 2)
          {
                    primes[j] = false;
          }
      }
   }
```

Legenda: Exemplo do resultado do algoritmo quando n = 120.

Algoritmos paralelos e sua caracterização

Em seguida serão apresentadas as versões dos algoritmos paralelos e a sua respetiva caracterização. Todas as versões do algoritmo utilizam o algoritmo de "The Sieve of Erastosthenes" com algumas modificações. As versões dos algoritmos são os seguintes: paralelismo com memória partilhada (recorrendo à API do OpenMP); paralelismo com memória distribuída (recorrendo à biblioteca do OpenMPI); e paralelismo com memória distribuída e partilhada (recorrendo à API e biblioteca do OpenMP e OpenMPI, respetivamente). Para compreender cada uma destas versões do algoritmo é importante distinguir a parte sequencial da parte possível de executar em paralelo.

Paralelismo com memória partilhada

Parte da versão do algoritmo de paralelismo com memória partilhada pode ser executada em paralelo. Essa parte é após a determinação do próximo número primo: aquando do ciclo de marcação dos números múltiplos do atual primo. É neste momento que é atribuída a cada *thread* uma parte da marcação dos números múltiplos do atual primo. A título de ilustração, a alteração que é feita é neste bloco de código:

```
#pragma omp parallel for

for (ull j = (ull)pow(i + 2, 2) - 2; j < primesSize; j = j + i + 2)

{

    primes[j] = false;

}
```

A diretiva “#pragma omp parallel for” do OpenMP adicionada permite este bloco de código ser executado em paralelo pelas várias *threads* disponíveis/atribuídas previamente.
Esta versão é considerada como memória partilhada pois é partilhado entre os processos/*threads* a carga de dados, neste caso, a lista de números a marcar.

Paralelismo com memória distribuída

A versão do algoritmo de paralelismo com memória distribuída é caracterizada por definir, para cada processo, o bloco de memória associado ao mesmo. Em seguida, após a determinação do próximo número primo é enviado, feito um *Broadcast*, pelo processo principal (rank = 0) do próximo número primo para cada processo marcar no seu bloco os números múltiplos desse primo. Em seguida será apresentado o código referente às alterações feitas de modo a tornar numa versão paralela com memória distribuída:

- Calculo do tamanho de cada bloco, primeiro índice desse bloco, e último índice do mesmo bloco:

```
#define BLOCK_LOW(i,p,n) (i)*(n)/(p)

#define BLOCK_HIGH(i,p,n) (BLOCK_LOW((i)+1,p,n)-1)

#define BLOCK_SIZE(i,p,n) (BLOCK_LOW((i)+1,p,n)-BLOCK_LOW(i,p,n))

ull blockSize = BLOCK_SIZE(rank, size, primesSize-1);

ull lowValue = BLOCK_LOW(rank,size,primesSize-1)+2;

ull highValue = BLOCK_HIGH(rank,size,primesSize-1)+2;
```

- Determinação do primeiro valor do bloco a ser marcado, não implicando ser o primeiro valor do bloco, mas sim o primeiro número múltiplo do primo.

```
if (pow(i,2) < lowValue){

        if((lowValue) % (i) == 0)

            startBlockValue = lowValue;

        else

            startBlockValue = lowValue + ((i) - (lowValue % (i)));

    }

    else

        startBlockValue = pow(i,2);
```

- Envio a todos os processos do próximo valor primo:

```
MPI_Bcast(&i, 1,MPI_UNSIGNED_LONG,0,MPI_COMM_WORLD);
```

A função “MPI_Bcast()” do OpenMPI tem o propósito de enviar aos processos o próximo valor primo.
Esta versão do algoritmo é considerada como memória distribuída pois é distribuído uma parte do bloco dos números por cada processo.

Paralelismo híbrido (memória distribuída e partilhada)

A versão paralela com memória distribuída e partilhada do algoritmo de "The Sieve of Erastosthenes", como o nome indica, recorre aos conceitos aplicados nos dois algoritmos anteriores, isto é, a cada processo é atribuído uma parte do bloco dos números; é indicado a todos os processos o próximo número primo a utilizar para marcar os seus múltiplos; e para cada processo é atribuída a cada *thread* uma parte da marcação dos números múltiplos do atual primo.

É esperado melhores tempos de execução para a versão em paralelo com memória distribuída e partilhada (híbrida) em comparação às outras versões, quer paralelas quer sequencial, uma vez que a versão híbrida utiliza a possibilidade de distribuir pelos vários processos partes do bloco de números, aliando o poder computacional de *multithreading* a serem operados a cada parte desses bloco. Ou seja, dar a cada processo pequenos blocos onde a cada bloco destes há várias threads a operar sobre eles. Tal como mencionado anteriormente, há limitações provenientes da(s) própria(s) máquina(s) e ainda o excesso de processos e/ou *threads* pode influenciar a performance, sendo que a partir de certo número, de processos e/ou *threads*, o tempo de execução aumenta de forma exponencial. Esta situação deve-se principalmente ao facto de haver concorrência entre os processos e entre as próprias threads.
Nos próximos tópicos serão explicados e justificados, empiricamente, qual a melhor versão dos algoritmos, apresentando comparações entre as versões e alguns valores de referência.

Métricas de performance

As métricas de performance usadas foram as seguintes:

- ***Speedup*** em função do **número máximo** da dimensão dos dados (**N**);
- **Número de instruções por segundo**, ***MOP/s***, em função do **número máximo** da dimensão dos dados (**N**);
- **Eficiência** em função do **número de processadores**;

Através destas métricas será possível comparar e avaliar qual a melhor versão de paralelismo implementado e ainda quantificar a sua superioridade/inferioridade e ainda a eficiência na sua escalabilidade. O *Speedup* permitirá avaliar o quanto a versão paralela será melhor/pior ao nível do tempo de execução relativamente à versão sequencial do algoritmo. O *MOP/s* (milhões de instruções por segundo) permitirá avaliar a velocidade média de processamento. A eficiência medirá a taxa de utilização dos processadores quando o programa é executado em paralelo, desta maneira será possível ver a qualidade do paralelismo.
Para determinar o valor destas métricas de performance será feito o seguinte cálculo:

- *Speedup*:

$$Speedup = \frac{Tsequencial}{Tparalelo}$$

- MOP/s:

$$MOP/s = \frac{2^N \; log \; log \; 2^N}{Texecução \; * \; 10^6}$$

- Eficiência:

$$Eficiência = \frac{Speedup}{Nº \; Processadores}$$

Teoricamente, é esperado que a versão sequencial do algoritmo tenha uma performance pior relativamente a todas as versões de paralelismo. Isto deve-se ao facto de distribuir e dividir as tarefas, possíveis fazer em paralelo, pelos processos e respetivas *threads*, havendo, deste modo, rapidez na execução do algoritmo até certo limite imposto pela(s) máquina(s) utilizadas. Este limite é definido pela potência dos processadores, o número de processadores e/ou máquinas e a capacidade de *threading* associada aos processadores.

Características do(s) computador(es) de testes:
Modelo Processador: Intel(R) Core (™) i7-4790 CPU @ 3.60GHz
Nº Processadores: 8
Tamanho Cache: 8192 KB
Memória Ram total: 16342672 KB

Experiencias realizadas:

Todas as experiencias utilizadas tiveram o objetivo de determinar o tempo de execução de cada versão do algoritmo tendo em conta as seguintes variantes, em função do algoritmo:

- **Versão Sequencial:**

Realizado num único computador. Variando o valor máximo,2^N, sendo N ∈ $[25; 32]$;

- **Versão paralela com memória partilhada:**

Realizado num único computador. Variando o valor máximo, 2^N, sendo N ∈ $[25; 32]$; variando o número de *threads* entre $[1; 8]$;

- **Versão paralela com memória distribuída:**

Realizado em quatro computadores. Variando o valor máximo,2^N, sendo N ∈ $[25; 32]$; variando o número de processos entre $[1; 32]$, adaptando o número de processos/pc entre 1, 2,4 e 8;

- **Versão paralela híbrida (memória distribuída e partilhada):**

Realizado em quatro computadores. Variando o valor máximo, 2^N, sendo N ∈ $[25; 32]$;;variando o número de *threads* entre $[1; 8]$; variando o número de processos entre $[1; 32]$, adaptando o número de processos/pc entre 1, 2,4 e 8;

Medidas de tempo dos programas em paralelo

De modo a obter as métricas de performance, foi preciso determinar o tempo de execução para cada uma das versões do algoritmo. No próximo gráfico serão apresentados os melhores valores para cada uma das versões do algoritmo:

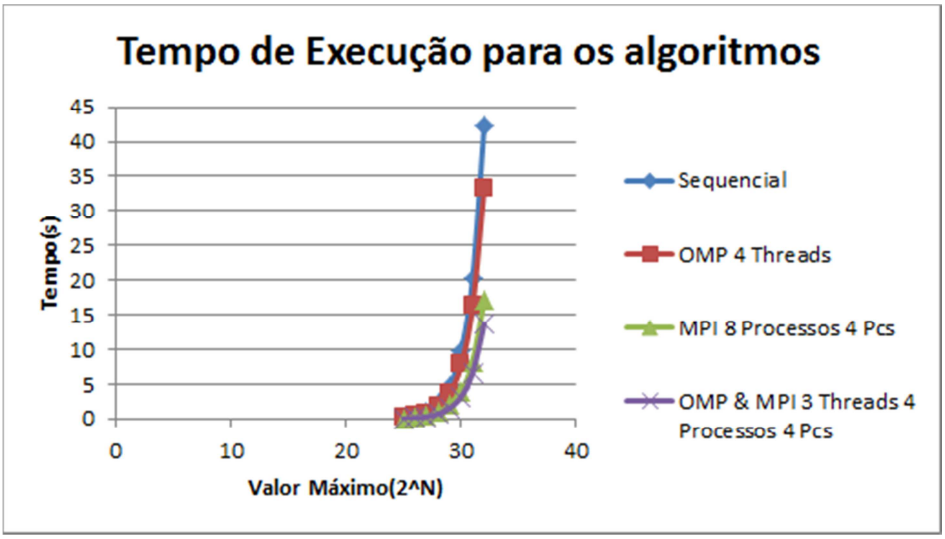

Gráfico 1: gráfico de dispersão representativo dos melhores tempos de execução de cada versão do algoritmo.

Destaca-se, como esperado, com o aumento de volume de dados, o tempo de execução das versões dos algoritmos aumenta exponencialmente. Realça-se que o algoritmo OMP & MPI com 3 *Threads*, 4 Processos e 4 Pcs (versão híbrida) apresenta os melhores resultados, isto é, menor tempo de execução, comparando com o MPI com 8 Processos e 4 Pcs (versão memória distribuída); comparando com OMP com 4 *Threads* (versão memória partilhada), que, por sua vez, este, versão memória partilhada, tem pior tempo de execução que o anterior, versão memória distribuída; e comparando com o Sequencia, que por sua vez, este é pior que a versão memória partilhada.

Escrita e Análise dos resultados

Como referido anteriormente, para avaliar a performance dos algoritmos e analisar a Escalabilidade do mesmo, recorreu-se às métricas de *Speedup* e MOP/s para avaliar a performance e a métrica da Eficiência para analisar escalabilidade. Em seguida será apresentado, graficamente, os valores mais pertinentes para estes dois estudos.

Avaliação da Performance

Através do cálculo do *Speedup*, será possível achar uma referência comum entre os algoritmos e estabelecer uma comparação entre eles, indicando o melhor entre eles.
Uma vez que o *Speedup* compara, relativamente, o quanto o tempo de execução em paralelo é superior ao tempo de execução sequencial, não aparece o valor do *Speedup* para a versão do algoritmo sequencial no gráfico seguinte.

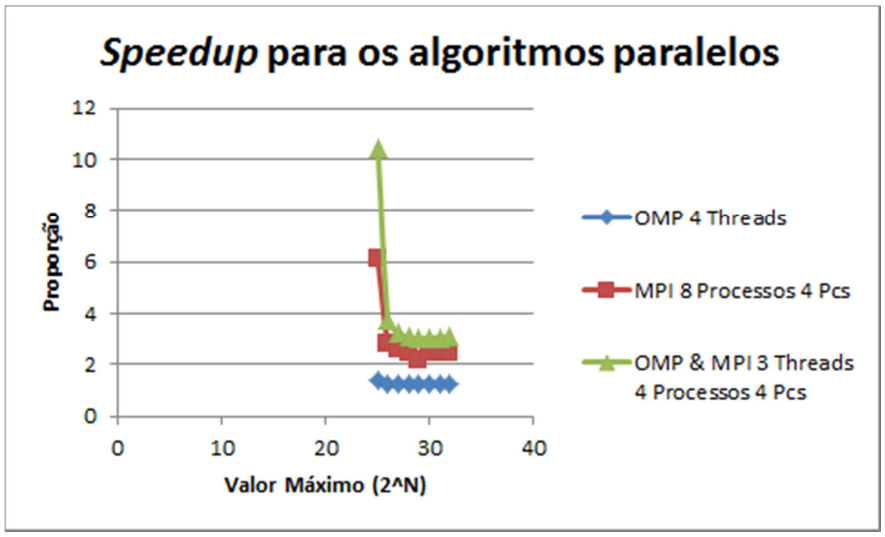

Gráfico 2: gráfico de dispersão representativo dos valores de *Speedup* de cada versão do algoritmo em paralelo para os melhores tempos.

Analisando o gráfico do *Speedup*, como conjeturado, a versão do algoritmo híbrido apresenta a melhor performance relativamente às outras versões paralelas do algoritmo. Salienta-se que o *Speedup* mantem-se constante em função do volume de dados, retirando o primeiro valor do gráfico que deve-se ao facto de o cálculo do tempo de execução para valores pequenos tenha pouca precisão, sendo, por esta razão, pouco fidedigno.

Por sua vez, os MOP/s (milhares de instruções por segundo) permitirá medir a velocidade média de processamento do algoritmo e, por consequência, afirmar qual dos algoritmos é o mais rápido.

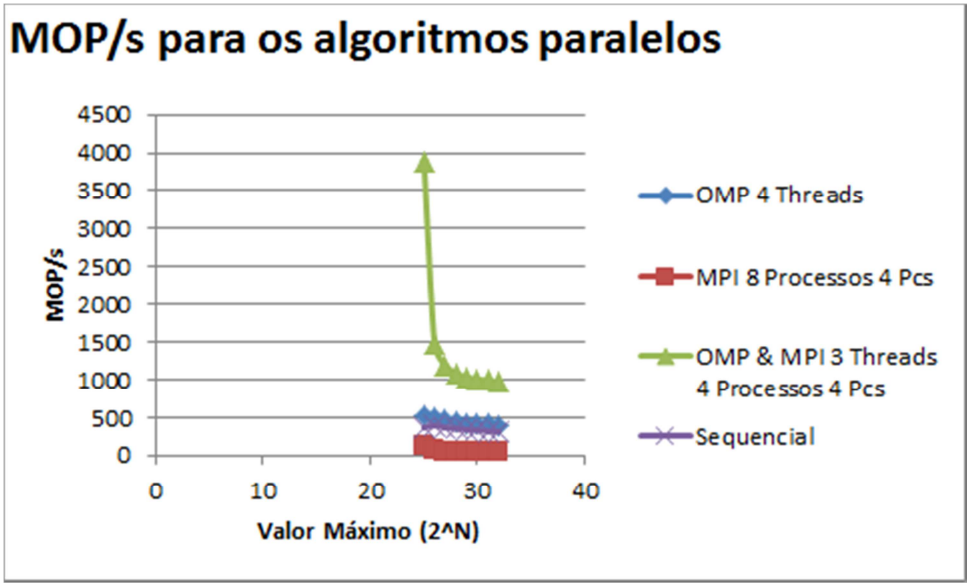

Gráfico 3: gráfico de dispersão representativo dos valores de MOP/s de cada versão do algoritmo em paralelo para os melhores tempos.

Mais uma vez, é evidente que a versão do algoritmo com maior velocidade média de processamento é a versão híbrida. Salienta-se, como no caso anterior, a anomalia para o primeiro valor ser díspar em relação aos outros. Este facto deve-se, como fora justificado anteriormente, a imprecisões na determinação do tempo de execução pois o valor é muito pequeno. Pode-se verificar que com o aumento do N, a velocidade média de processamento mantem-se constante.

Análise da Escalabilidade

A análise da escalabilidade permite, através da eficiência, verificar a qualidade da paralelização, isto é, permite perceber se o algoritmo mantém a mesma qualidade (mesmo rácio de qualidade) aumentando o número de processadores e o volume de dados.
O gráfico seguinte apresenta os valores do estudo efetuado para a versão paralela dos algoritmos de memória distribuída e híbrida. Apenas estes algoritmos é que foram sujeitos a vários processadores.

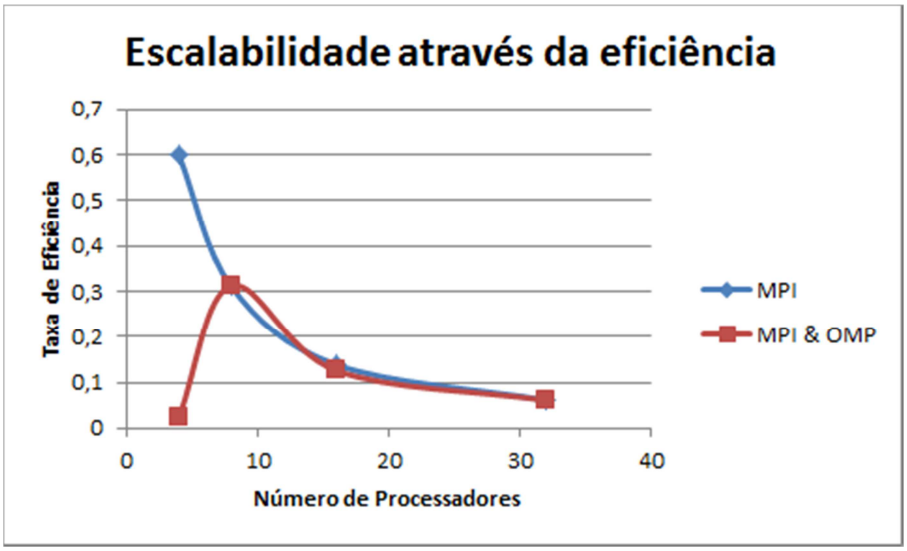

Gráfico 4: gráfico de dispersão representativo dos valores de Eficiência para as versões de memória distribuída e híbrida para os melhores tempos.

Através da interpretação deste gráfico, como era previsto, os valores do rácio da Eficiência mantêm-se constante com o aumento do número de processadores. O mesmo se sucede com o aumento do volume de dados. Isto significa que para além das versões paralelas do algoritmo (excluindo a versão de memória partilhada) serem de qualidade, têm a capacidade de serem escaláveis para um volume bastante superior de dados e de processadores.

Conclusão

Após uma análise dos dados recolhidos e devida comparação e reflexão, chega-se à conclusão que a versão do algoritmo mais eficiente é a versão paralela com memória distribuida e partilhada (versão híbrida). Antes de efetuar os testes já era especulável que

esta versão fosse a que iria demonstrar melhores resultados pois está-se a dividir por entre vários processos pequenos fragmentos de memória onde várias *threads* vão operar sobre eles.
Traduzindo para a seguinte analogia, esta situação seria o mesmo que numa fábrica de produção de um refrigerante de lata. Esta fábrica estará dividida em vários setores, chamados de processos, para produzir o conteúdo, produzir o recipiente, produzir o rótulo, etc. Dentro de cada setor haverá pessoas, chamadas de *threads*, com o propósito de realizar as tarefas associadas ao seu setor. Desta maneira e como os setores são independentes entre si, isto permite realizar tarefas em paralelo o que, consequentemente, permite uma otimização e ganho de tempo durante o processo. Infelizmente o aumento de *threads* e ou processos não permite um aumento da performance linear pois o excesso de *threads* e/ou processos pode fazer com que haja atrasos o que, por conseguinte, gera declínio de performance.
Não menosprezando individualmente os modelos de memória partilhada nem memória distribuída, estes também aumentão a performance, sendo que o modelo de memória distribuída tem resultados superiores ao modelo de memória partilhada, pois, o modelo de memória distribuída tira partido de várias máquinas para efetuar o processamento.
Assim sendo, a dualidade memória partilhada e distribuída é uma vantagem quando bem empregues nos blocos onde é possível escalonar atividades em paralelo pois acelera o processamento do algoritmo, aumentando assim a sua performance.