

UTILIZAÇÃO DA INTERFACE OPENMP PARA O ALGORITMO DE ORDENAÇÃO SELECTION SORT

Rodrigo Odorizzi¹, Rodrigo Curvêllo²

^{1,2}Curso de Bacharelado em Ciência da Computação – Instituto Federal Catarinense
(IFC) 89160-240 – Rio do Sul – SC, Brasil

rodrigoodorizzi@hotmail.com, rodrigo.curvello@gmail.com

Abstract. *This work, to explain the concept and to use the parallel programming and the use of the interface for multiprogramming OpenMP. Information about policies and clauses is used for this interface. Next, the use of the sorting algorithm will be presented by selecting the elapsing time of the serial and parallel programming.*

Key-words: Paralelo, Programação, C++

Resumo. *Este trabalho, explicará o conceito e uso sobre programação paralela e o uso da interface para multiprogramação OpenMP. Abordará informações sobre diretivas e cláusulas usadas para esta interface. Em seguida será apresentado o uso do algoritmo de ordenação Selection Sort, elucidando o tempo de processamento, da programação serial e paralela.*

1. Introdução

Computação Paralela segundo Freire (2010) é um paradigma de computação que consiste, basicamente, na divisão de um problema em partes que podem ser resolvidas ao mesmo tempo, paralelamente, onde essa forma de computação já vem sendo usada há certo tempo para a resolução de problemas de complexidade elevada e que precisam de uma quantidade considerável de poder de processamento [1].

A interface de multiprogramação OpenMP para Sena(2008), se trata de uma API e um conjunto de diretivas que permite a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de threads. Suas funcionalidade podem atualmente ser utilizadas nas linguagens Fortran 77, Fortran 90, C e C++. Seu nome deriva de Open significa que é padrão e está definido por uma especificação de domínio público e MP são as siglas de Multi Processing [2]. O OpenMP teve a sua primeira versão em 1997.

2. Fundamentação Teórica

Para Sena(2008), Ao longo dos anos, o desenvolvimento científico tem exigido das simulações numéricas resultados cada vez mais confiáveis e próximos da realidade.

Dessa forma, para acompanhar esses avanços, a computação científica se depara com o desafio de superar as diversas barreiras que surgem em virtude da limitação física dos computadores, tais como a demanda por processamento numérico, armazenamento de dados, visualização, entre outros. Nesse contexto, a computação de alto desempenho (High Performance Computing - HPC), termo amplamente utilizado na computação científica, tem se apresentado como uma importante frente de pesquisa nos últimos anos. O termo pode ser definido como qualquer conjunto de técnicas que visem otimizar ou até mesmo viabilizar o processamento de simulações numéricas. Podemos citar como uma dessas técnicas o uso de processamento paralelo em clusters e supercomputadores. [3].

2.1 O Uso de aplicações de alto desempenho no Brasil

Comumente, as aplicações de alto desempenho são classificadas em quatro estágios, estágio rudimentar, onde um país não utiliza de soluções de alto desempenho, estágio inicial, onde o país utiliza de aplicações desenvolvidas mundialmente, estágio três, onde a nação começa a acrescentar funcionalidades nas aplicações que recebe, e estágio quatro, onde o país exporta essas aplicações e não apenas acrescenta.

Segundo (PANETTA), alguns fatores devem ser considerados para a classificação por uso de aplicações de alto desempenho, o seu nível de complexidade é um desses fatores, outro fator que deve se levar em consideração, é de um país ser consumidor de determinado nicho, e produtor de outro, isso na área de aplicação de alto desempenho, fatores esses que pesam na hora da avaliação.

Segue um resumo sobre os quatro estágios:

Estágio 1 – não usa de aplicações de alto desempenho

Nesta lista se encontram em sua maioria, nações de economia fechada, na qual tem se a dificuldade inclusive de se obter máquinas de alto desempenho para uso dessas aplicações, e ainda não desenvolvem soluções para o mercado mundial.

Estágio 2 – Consumidor

Neste estágio encontram-se países que investem no conhecimento das aplicações de alto desempenho, pois geralmente um produto é geralmente usado, pois se conhece. Segundo o autor, Seguramente, o Brasil é um bom usuário de aplicações desenvolvidas no exterior. Basta observar as filiais de "software houses" internacionais aqui instaladas. Fregueses já estabelecidos são os parques das indústrias automotiva, aeronáutica e de petróleo.

Estágio 3 – Acrescenta funcionalidades

No estágio três, além de ser um consumidor, lugares onde estão nesse estágio, acrescentam funcionalidades conforme a sua necessidade, e mais do que isso, fazer o uso contínuo dessas ferramentas, afim de cada vez mais, acrescentar funcionalidades a este sistema.

Estágio 4 – Produtor

Neste último estágio, estão aqueles que além de consumir, e acrescentar funcionalidades, são os progenitores de aplicações paralelas, onde os utilizadores dos estágios anteriores usam as aplicações do produtor.

Utilização de OpenMP para programação paralelas

Segundo (MAILLARD), o OpenMP possibilita a distribuição das instruções de um programa entre vários fluxos de execução(ou threads),as quais compartilham parte da sua memória. O desenvolvimento desses programas OpenMP acontecem com o uso de pragmas, onde são declarados no cabeçalho do programa. O programa só funcionará caso a macro `_OPENMP` estiver definida.

Distribuindo cálculos e dados com OpenMP

Para distribuir cálculos e dados com OpenMP, se faz necessário o uso da diretiva `parallel`, onde através dela é disparada as threads concorrente, e o bloco que deve ser executado em paralelo na sequência. Há algumas cláusulas que garantem esse acesso compartilhado a dados na memória, dentre eles há o *shared* (*compartilha acesso global*), o *private* (*acesso privado*), *sections* (*executa determinada sessão*).

Sincronização de Threads

Em alguns casos, faz-se necessário o uso de apenas uma thread para executar uma determinada tarefa no programa, e isto é feito através da diretiva `single`, onde assim somente determinada thread executará determinada tarefa. Também é possível

especificar que todas as threads sejam executadas, mas de maneira não concorrente, isso tudo através da diretiva *omp critica*.

OpenMP 3.0

Segundo (MAILLARD), a norma OPENMP 3.0 foi lançada em 2008 e começa a estar disponível nos compiladores recentes (na data da escrita deste texto, o compilador icc da Intel já incluía o OPENMP 3.0). Uma das principais novidades é o suporte a um outro modelo de programação paralela além do paralelismo de laços: o paralelismo de tarefas. Ele é bem apropriado a algoritmos recursivos, onde uma(ou várias) tarefa(s), ao executar, dispara recursivamente outras tarefas.

Observa-se que esse paradigma de programação paralela é ao paralelismo de laços o que programação recursiva (em particular usada na programação funcional) é à programação iterativa. Muitos algoritmos eficientes usam recursividade, e é importante que o OpenMP possa dar suporte aos mesmos. Claramente, isso não significa que um programa que execute um laço naturalmente paralelo deva ser reprogramado de forma recursiva. Mas nos casos em que a recursividade fornece um algoritmo simples, é bom que a versão paralela seja semelhante.

Uso de compiladores OpenMP

Segundo (BONAT), a especificação OpenMP define um conjunto de diretivas de compilação, chamadas de função e variáveis de ambiente, de forma que a tarefa de paralelizar o código para uma arquitetura diferente seja uma responsabilidade delegada ao compilador dessa arquitetura em que se deseja rodar o programa, dessa forma não há alteração significativa no algoritmo para que o mesmo obtenha os benefícios de arquiteturas que ofereçam paralelismo, mas que também permita que o mesmo programa siga funcionando de forma sequencial.

Dentro os compiladores existentes podemos elencar o Intel Compiler, GOMP - GCC OpenMP e o Sun Studio.

Intel Compiler

Compilador que pode se obter uma licença por 30 dias, O código executável gerado por este compilador possui referencia para ligar em tempo de execução com as

bibliotecas `libguide` e `libpthread`, onde deve se setar as variáveis de ambiente para o compilador.

GOMP – GCC

A partir da versão 4.2 foi adicionado suporte a implementação `OpenMp`, para C, C++ e `fortran`. Sua execução se dá através do comando: `$ gcc -o saida programa.c -fopenmp`.

Sun Studio

Compilador gratuito, com o binário gerado por este compilador liga-se em tempo de execução com duas bibliotecas, `libmtsk` e `libpthread`.

2.3 Classificações de Arquiteturas Paralelas

Para Rose (2003) pode se verificar que as máquinas paralelas vem sendo cada vez mais usadas pelo ganho computacional que elas desempenham, porém elas ainda possuem um custo bem elevado e é de difícil programação. E para tentar amenizar este cenário, é importante conhecer os modelos que existem de arquitetura paralela e saber como funciona cada componente específico.

As arquiteturas paralelas podem ser classificadas e combinadas em quatro tipo:

- SI (Single Instruction) e SD (Single Data): Neste primeiro modelo, há um único conjunto de instrução atuando sob um único fluxo de dados, onde este fluxo de instrução alimenta uma única unidade de controle onde está por sua vez, ativa a unidade de processamento, e esta unidade de processamento atua sob o único fluxo de dados existente, onde é lido, processado e reescrito na memória.

- SI (Single Instruction) e MD (Multiple Data): Neste modelo uma única instrução é executada ao mesmo tempo sobre múltiplos dados. O processamento é controlado por uma única unidade de controle, alimentada por um único fluxo de instruções. A mesma instrução é enviada para os diversos processadores envolvidos na execução. Todos os processadores executam suas instruções em paralelo de forma síncrona sobre diferentes fluxos de dados.

- MI (Multiple Instruction) e SD (Single Data): Neste modelo múltiplos fluxos de instruções atuam sob um único fluxo de dados e múltiplas unidades de processamento possuem sua própria unidade de controle recebendo um fluxo diferente de instruções. Essas unidades de processamento executam suas diferentes instruções sobre o mesmo fluxo de dados. Diferentes instruções operam a mesma posição de memória ao mesmo tempo, executando instruções diferentes.

-MI (Multiple Instruction) MD (Multiple Data): Neste modelo, cada unidade de controle recebe um fluxo de instruções próprio e repassa-o para sua unidade processadora para que seja executado sobre um fluxo de instruções próprio. Dessa forma, cada processador executa o seu próprio programa sobre seus próprios dados de forma assíncrona. Sendo assim, o princípio MIMD é bastante genérico, pois qualquer grupo de máquinas, se analisado como uma unidade (executando, por exemplo, um sistema distribuído), pode ser considerado uma máquina MIMD.

3.0 Algoritmos de ordenação

Segundo LUCCHESI (2004), a ordenação é um dos problemas fundamentais em Computação e existem vários algoritmos para resolvê-lo. Pode-se demonstrar que um algoritmo baseado apenas em comparações de chaves exige da ordem de $O(n \log n)$ operações de comparação, no pior caso.

Na computação existe uma série de algoritmos que utilizam diferentes técnicas de ordenação para organizar um conjunto de dados, eles são conhecidos como *Métodos de Ordenação* ou *Algoritmos de Ordenação*. Estes métodos podem ser divididos em métodos de ordenação interna ou métodos de ordenação externa.

Dentro dos métodos de ordenação interna, encontra-se os *métodos simples* e os *métodos eficientes*. Entende-se por método simples, métodos que são adequados para pequenos vetores, ou seja, são programas pequenos e fáceis de entender. Possuem complexidade $C(n) = O(n^2)$, ou seja, requerem $O(n^2)$ comparações. O método de ordenação Selection Sort é um método simples.

3.1 O método Select Sort

A ordenação por seleção ou *selection sort* consiste em selecionar o menor item e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição, segue estes passos até que reste um único elemento. Para todos os casos (melhor, médio e pior caso) possui complexidade $C(n) = O(n^2)$ e não é um algoritmo estável. Segue abaixo o pseudo código elucidando a explicação acima.

01	var i,j,aux,menor : INTEIRO
02	PARA i=0 ATE (vet.tamanho-1) PASSO 1
03	menor = i
04	PARA j=i+1 ATE vet.tamanho PASSO 1
05	SE (vet[menor] > vet[j]) ENTAO
06	menor = j
07	FIMSE
08	FIMPARA
09	SE (menor != i) ENTAO
10	aux = vet[menor]
11	vet[menor] = vet[i]
12	vet[i] = aux
13	FIMSE
14	FIMPARA

Tabela 1 – Pseudo Código do algoritmo Selection Sort. Acervo: fonte do autor, 2019.

No Pseudo código, temos a declaração das variáveis (linha 01), abertura de dois laços de repetição (linhas 02 e 04) e nas linhas 03 e 05 atribuição dos menores índices percorrendo os vetores. Na linha 05 é atribuído o menor valor a j. e da linha 09 até a

linha 12, é feita a troca de posição do menor valor j pelo valor comparado através do laço de repetição. Linhas 13 e 14 finalizam o programa.

A seguir será mostrada uma imagem do resultado de uma implementação desse pseudo código.

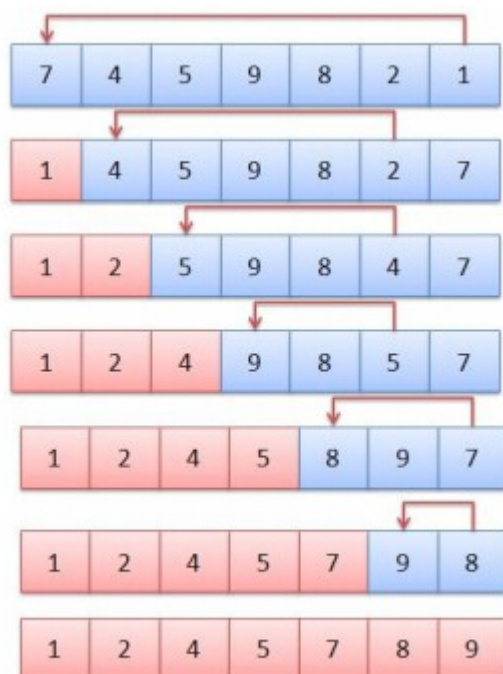


Figura 1 – Execução do algoritmo Selection Sort.

3.2 Vantagens do Selection Sort

Dentre as vantagens do algoritmo Selection Sort pode ser mencionado:

Estável: não altera a ordem dos dados iguais.

Fácil implementação e algoritmo simples.

3.3 Desvantagens do Selection Sort

Uma desvantagem é que o número de comparações é igual para o melhor caso, caso médio e o pior caso. Assim, mesmo que o vetor esteja ordenado o custo continua quadrático (n^2). Sua eficiência diminui drasticamente a medida que o número de elementos no array aumenta.

3.4 Origem do Algoritmo

Segundo artigo encontrado no site <http://joeldasilva.com.br>, tem como data de criação do algoritmo Select Sort em 1962, porém não informa quem foi seu criador, consultando demais fontes pela web, não foi possível encontrar maiores a respeito de seu fundador.

3.5 Implementação do algoritmo Selection Sort

Para elucidação será apresentado uma listagem com os resultados obtidos com através da implementação do algoritmo Selection Sort, utilizando a linguagem de programação C++ através do programa Microsoft Visual Studio 2017.

Primeiramente será mostrado os valores obtidos na ordenação de 10 vetores, com 10000 posições, e valores aleatórios de 1 a 10000 para cada posição.


```

9903
9915
9921
9931
9937
9938
9945
9956
9957
9962
9963
9966
9973
9976
9976
9990
9994
9995

Media de Tempo: 0.102219

C:\Users\rodrigo\source\repos\selection_oficial2\Debug\selection_oficial2.exe <p
rocesso 9988> foi encerrado com o código 0.
Pressione qualquer tecla para fechar esta janela...

```

Figura 2 – Resultados obtidos com o uso da programação paralela. Acervo do próprio autor.

Conforme pode ser visto na figura 2, o programa obteve um tempo em média de 0,10 segundos na ordenação de um vetor de 10000 posições com valores aleatórios. E para a obtenção desses resultados, utilizou-se as seguintes diretivas.

`#pragma omp parallel shared(n)`: diretiva utilizada na linha 22. onde com isso foi possível paralelizar todo o código responsável pelo método selectionsort, juntamente com a diretiva `shared`, onde efetuou o compartilhamento da variável `n`, utilizada para os laços de repetição do código.

`#pragma omp for`: diretiva de compartilhamento de trabalho utilizada na linha 26, aliada com a diretiva da linha 22, possibilita o compartilhamento das variáveis entre as threads.

`#pragma omp critical`: cláusula utilizada na linha 38 (segue imagem a seguir), apenas uma thread tem acesso a este escopo, sendo que as demais threads necessitam esperar pela conclusão deste processo, para prosseguir.

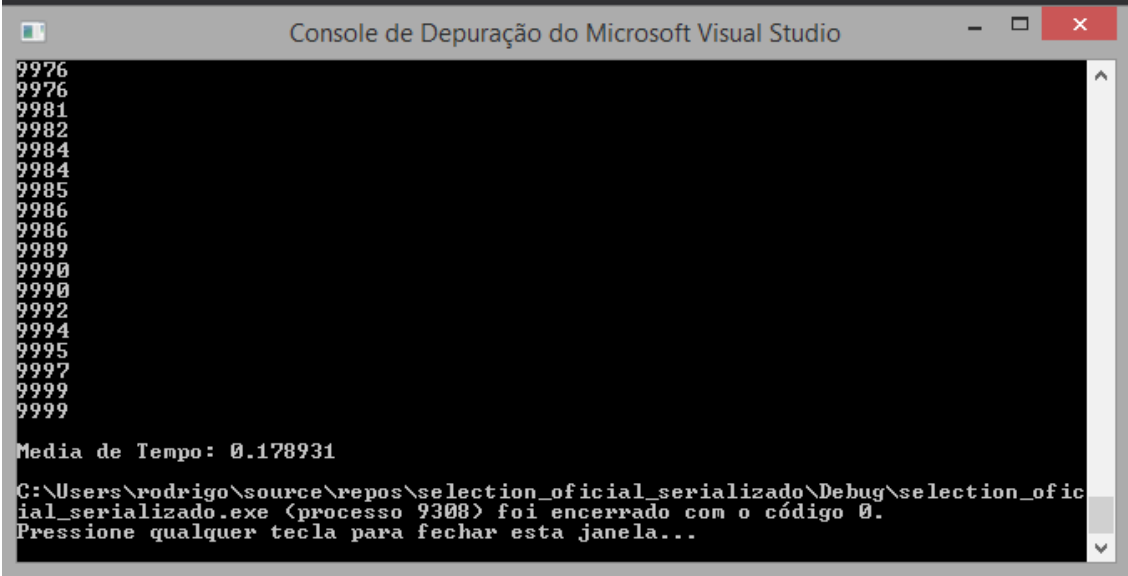
```

20 void selectionSort(int arr[], int n) {
21
22     #pragma omp parallel shared(n)
23     {
24
25         int i, j, minIndex, tmp;
26         #pragma omp for
27
28         for (i = 0; i < n - 1; i++) {
29
30             minIndex = i;
31
32             for (j = i + 1; j < n; j++)
33                 if (arr[j] < arr[minIndex])
34                     minIndex = j;
35             if (minIndex != i) {
36                 tmp = arr[i];
37                 arr[i] = arr[minIndex];
38                 #pragma omp critical
39                     arr[minIndex] = tmp;
40             }
41         }
42     }
43 }
44

```

Figura 3 – Método selectionSort, onde realiza a ordenação dos vetores. Acervo do próprio autor.

A seguir figura 4.0, será apresentado o algoritmo selection sort sem o uso das diretivas, ou seja, o programa Serializado. Onde o programa obteve um tempo em média de 0,17 segundos na ordenação de um vetor de 10000 posições com valores aleatórios.



```
Console de Depuração do Microsoft Visual Studio
9976
9976
9981
9982
9984
9984
9985
9986
9986
9989
9990
9990
9992
9994
9995
9997
9999
9999

Media de Tempo: 0.178931

C:\Users\rodrigo\source\repos\selection_oficial_serializado\Debug\selection_oficial_serializado.exe (processo 9308) foi encerrado com o código 0.
Pressione qualquer tecla para fechar esta janela...
```

Considerações finais

Com a constante necessidade de um processamento computacional maior e mais rápido, tanto por conta do desenvolvimento de pesquisas acadêmicas, quando para uso comercial, a programação paralela se mostra cada vez mais útil para as fins. E graças a essas necessidades, tem se desenvolvido várias ferramentas para a realização desses processos, como a criação da interface OpenMP. Também tem surgido vários compiladores como Intel Compiler, GOMP - GCC OpenMP e o Sun Studio. E com o atual crescimento desta necessidade, consequentemente cada vez mais, a programação paralela estará fazendo parte em nossa realidade.

Link do projeto: https://github.com/RodrigoOdorizzi/selection_sort.git

Referências

O avanço da computação paralela. Disponível em: <<http://www.dsc.ufcg.edu.br/~pet/jornal/fevereiro2010/materias/carreira.html>>. Acesso em 17 de Junho de 2019

[2] SENA, M.C.R; COSTA, J.A.DE.C. **Tutorial OpenMp C/C++** - 2008

[3] ROSE, C.A. DE. **Fundamentos de Processamento de Alto Desempenho** – 2003

[4] PANETTA, J. **Aplicações de alto desempenho no Brasil**. ERAD – São Leopoldo, 2002.

MAILLARD, N; Cera, M.C. **Desenvolvendo aplicações OpenMP** - ERAD – Passo Fundo, 2010.

BONAT, A; Giusti, F, **Análise de compiladores com suporte a OpenMP**. ERAD – Santa Cruz do Sul, 2008.

Conheça os principais algoritmos de ordenação. Disponível em:

<<https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao>>.

Acesso em 21 de Maio de 2019

Métodos de Ordenação. Disponível em:

<http://www.inf.ufg.br/~hebert/disc/aed1/AED1_04_ordenacao1.pdf>. Acesso em 21

de Maio de 2019.

Algoritmos de ordenação. Disponível em:

<<http://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>>. Acesso em 21 de

Maio de 2019.

Programador aprendendo. Disponível em:

<<http://programadoraprendendo.blogspot.com/2012/11/algoritmo-selection-sort.html>>.

Acesso em 21 de Maio de 2019.

Pesquisa e ordenação dos dados. Disponível em:

<http://joeldasilva.com.br/pod_2018_1/aula_02/Aula_02_POD_Ordenacao_de_Dados+_Insert_Sort.pdf>. Acesso em 21 de Maio de 2019.

LUCCHESI, C. L. Estrutura de dados e técnicas de programação, 2004.

<http://www.dsc.ufcg.edu.br/~pet/jornal/fevereiro2010/materias/carreira.html>