Trabalho 1: Bubble Sort (OpenMP)

O objetivo do trabalho é paralelizar, usando a ferramenta OpenMP, uma versão sequencial de um programa que ordena vários vetores usando o algoritmo Bubble Sort (vide código no final desta definição).

Ao mesmo tempo que serão usadas as diretivas de OpenMP vistas em aula sobre um problema real, será possível também fazer uma reflexão sobre as questões que devem ser levadas em consideração quando um código sequencial é paralelizado.

A versão sequencial ordena **NUM_ARRAYS** vetores de inteiros, com o tamanho de cada array (variável array_size) variando de 2500 (constante INI_ARRAY_SIZE) até 25000 (constante MAX_ARRAY_SIZE), de 2500 em 2500 elementos (constante INC_ARRAY_SIZE).

Para paralelizar a versão sequencial, as diretivas de OpenMP devem ser colocadas nos pontos corretos, usando cláusulas adequadas que busquem o melhor desempenho possível.

Para isso, é importante identificar corretamente o melhor laço a ser paralelizado, quais são as suas variáveis compartilhadas e privadas, se há seções críticas, a possibilidade de usar reduções e qual a melhor cláusula para escalonamento das iterações do laço paralelizado.

Após a paralelização, a versão paralela deve ter o seu desempenho avaliado, considerando a execução em um nodo do cluster GRAD, do Laboratório de Alto Desempenho da PUCRS (LAD), com **2**, **4**, **8** e **16 threads**, com ARRAY_SIZE (constante NÃO definida no código-fonte) igual a **25**, **150** e **500 vetores**.

Cada execução deve ser feita no mínimo 3 vezes, tomando-se os menores tempos para cada tamanho de array, de forma a minimizar interferências causadas por outros processos em execução no cluster.

Cada aluno ou grupo (de no máximo 2 integrantes) deve entregar um relatório em .PDF com: descrição dos testes realizados, descrição do processador, resultados obtidos (gráficos mostrando os resultados de várias execuções), análise dos resultados e conclusões.

Os itens para avaliação são:

- **Formação de um grupo de no máximo 2 integrantes** em até uma semana após a divulgação do enunciado do trabalho;
- **Execução da versão sequencial** abaixo (para viabilizar o cálculo de speed-up e eficiência) no cluster GRAD;
- **Implementação da versão paralela** do algoritmo em C e OpenMP;
- **Execução da versão paralela com número de vetores igual a 25, a 150 e a 500, para 2, 4, 8 e 16 threads**, cada uma com no **mínimo 3 repetições** considerando-se, ao final, o **menor tempo para cada tamanho de vetor**.
- **Cálculo de speed up e de eficiência** para os testes paralelos executados;
- **Análise do balanceamento de carga** na execução do programa paralelo;

- **Relatório escrito usando o template de artigo disponibilizado no moodle** (em formato .PDF), com no mínimo 2 páginas de texto e com gráficos (devem ser apresentados **gráficos de tempo, speed-up e eficiência**, separados para **25**, **150** e **500 vetores**, com os tamanhos dos vetores variando de 2500 até 25000);

Vencimento: quinta, 29 Set 2022, 19:15

Trabalho 1: Bubble Sort (OpenMP)

- Arquivo compactado no formato ZIP incluindo artigo em PDF, código-fonte e demais arquivos necessários à compilação e execução (é obrigatória a entrega do código-fonte da versão paralela de forma que se possa testá-la, caso seja necessário);
- Submissão através do moodle até o dia e o horário definidos na respectiva sala de entrega;
- Análise do número de horas máquina usadas pelo grupo no LAD durante o desenvolvimento do trabalho.

No dia da entrega do trabalho, todos os componentes do grupo devem comparecer à aula para apresentação de seu trabalho e devem estar em condições de provar de forma argumentativa que participaram do desenvolvimento do trabalho. Esta apresentação poderá ocorrer em um ou mais dos seguintes formatos:

- Apresentação oral para o professor, com os integrantes respondendo perguntas sobre o desenvolvimento do trabalho;
- Questionário escrito sobre o desenvolvimento do trabalho;
- Realização de uma execução no cluster GRAD para mostrar que os dados obtidos correspondem aos resultados citados no relatório.

NÃO serão avaliados trabalhos que:

- Tiverem sido entregues por um grupo que NÃO foi definido dentro do prazo estipulado;
- NÃO forem entregues na respectiva opção de entrega no moodle da disciplina, até o dia e horário definidos;
- NÃO forem devidamente apresentados;
- NÃO incluírem o relatório em formato de artigo, conforme descrito acima;
- NÃO contiverem resultados de execução no cluster ou que NÃO tiverem sido executados no cluster do LAD;
- NÃO contiverem gráficos mostrando os resultados da execução;
- NÃO incluírem o código-fonte (compilando sem erros) no arquivo compactado entregue no moodle.

Vencimento: quinta, 29 Set 2022, 19:15

Trabalho 1: Bubble Sort (OpenMP)

```
#include <stdio.h>
#include <omp.h>

#define MAX_ARRAY_SIZE 25000
#define INI_ARRAY_SIZE 2500
#define INC_ARRAY_SIZE 2500

// ESTRUTURA DE DADOS COMPARTILHADA
int arrays[NUM_ARRAYS][MAX_ARRAY_SIZE];

// BUBBLE SORT
void BubbleSort(int n, int * vetor) {
  int c =0, d, troca, trocou =1;

  while (c < (n-1) && trocou ) {
       trocou = 0;
       for (d = 0 ; d < n - c - 1; d++)
          if (vetor[d] > vetor[d+1]) {
             troca     = vetor[d];
             vetor[d]   = vetor[d+1];
             vetor[d+1] = troca;
             trocou = 1;
          }
       c++;
  }
}

int main() {
  int i, j, array_size;
  double tempo;

  for (array_size = INI_ARRAY_SIZE; array_size <= MAX_ARRAY_SIZE; array_size += INC_ARRAY_SIZE) {

     // INICIALIZA OS ARRAYS A SEREM ORDENADOS
     #pragma omp parallel for private(j) num_threads(omp_get_num_procs())
     for (i=0 ; i<NUM_ARRAYS; i++) {
        for (j=0 ; j<array_size; j++) {
           if (i%5 == 0)
             arrays[i][j] = array_size-j;
           else
             arrays[i][j] = j+1;
        }
     }

     // REALIZA A ORDENACAO
     tempo = -omp_get_wtime();
     for (i=0 ; i<NUM_ARRAYS; i++) {
        BubbleSort(array_size, &arrays[i][0]);
     }
     tempo += omp_get_wtime();

     // VERIFICA SE OS ARRAYS ESTAO ORDENADOS
     for (i=0 ; i<NUM_ARRAYS; i++)
        for (j=0 ; j<array_size-1; j++)
           if (arrays[i][j] > arrays[i][j+1])
             return 1;

     // MOSTRA O TEMPO DE EXECUCAO
     printf("%d %lf\n",array_size, tempo);
  }
  return 0;
}
```

Vencimento: quinta, 29 Set 2022, 19:15

Trabalho 1: Bubble Sort (OpenMP)

Roteiro para a realização do trabalho:

Acessar a GRAD:
Abrir o CMD como administrador e digitar: ssh portoalegre\<matrícula>@sparta.pucrs.br
Digitar YES e autenticar com a senha de rede
Acessar a GRAD: ssh -o PasswordAuthentication=yes <usuárioGRAD>@grad.lad.pucrs.br
Digitar YES e autenticar com a senha cadastrada anteriormente na GRAD.

Preparar o ambiente GRAD:
Criar pasta do trabalho: mkdir trab_i
Acessar a pasta trab_i: cd trab_i
Criar pasta para alocar os resultados de output: mkdir out
Criar pasta para alocar os resultados de tempo: mkdir tmp

Preparar arquivo .c sequencial:
Criar arquivo .c da aplicação no editor nano: nano bss.c
Copiar o código do moodle e colar no arquivo bss.c
Salvar arquivo e sair do editor nano: ctrl + O > Enter > ctrl + X

Preparar arquivo .c paralelo:
Criar arquivo .c da aplicação no editor nano: nano bsp.c
Copiar o código do moodle e colar no arquivo bsp.c
Inserir diretiva #pragma omp parallel for num_threads(NUM_THREADS) na linha 47
Salvar arquivo e sair do editor nano: ctrl + O > Enter > ctrl + X

Compilações:
Sequencial com 25 Arrays: gcc -o bss-25 -DNUM_ARRAYS=25 -fopenmp bss.c
Sequencial com 150 Arrays: gcc -o bss-150 -DNUM_ARRAYS=150 -fopenmp bss.c
Sequencial com 500 Arrays: gcc -o bss-500 -DNUM_ARRAYS=500 -fopenmp bss.c
Paralela com 25 arrays e 2 Threads: gcc -o bsp-25.2 -DNUM_ARRAYS=25 -DNUM_THREADS=2 -fopenmp bsp.c
Paralela com 25 arrays / 4 Threads: gcc -o bsp-25.4 -DNUM_ARRAYS=25 -DNUM_THREADS=4 -fopenmp bsp.c
Paralela com 25 arrays / 8 Threads: gcc -o bsp-25.8 -DNUM_ARRAYS=25 -DNUM_THREADS=8 -fopenmp bsp.c
Paralela com 25 arrays / 16 Threads: gcc -o bsp-25.16 -DNUM_ARRAYS=25 -DNUM_THREADS=16 -fopenmp bsp.c
Paralela com 150 arrays / 2 Threads: gcc -o bsp-150.2 -DNUM_ARRAYS=150 -DNUM_THREADS=2 -fopenmp bsp.c
Paralela com 150 arrays / 4 Threads: gcc -o bsp-150.4 -DNUM_ARRAYS=150 -DNUM_THREADS=4 -fopenmp bsp.c
Paralela com 150 arrays / 8 Threads: gcc -o bsp-150.8 -DNUM_ARRAYS=150 -DNUM_THREADS=8 -fopenmp bsp.c
Paralela com 150 arrays / 16 Threads: gcc -o bsp-150.16 -DNUM_ARRAYS=150 -DNUM_THREADS=16 -fopenmp bsp.c
Paralela com 500 arrays / 2 Threads: gcc -o bsp-500.2 -DNUM_ARRAYS=500 -DNUM_THREADS=2 -fopenmp bsp.c
Paralela com 500 arrays / 4 Threads: gcc -o bsp-500.4 -DNUM_ARRAYS=500 -DNUM_THREADS=4 -fopenmp bsp.c
Paralela com 500 arrays / 8 Threads: gcc -o bsp-500.8 -DNUM_ARRAYS=500 -DNUM_THREADS=8 -fopenmp bsp.c
Paralela com 500 arrays / 16 Threads: gcc -o bsp-500.16 -DNUM_ARRAYS=500 -DNUM_THREADS=16 -fopenmp bsp.c

Batchjobs			
Tipo	Nome	Saida	Tempo
Sequencial	bjs	bjs.out	bss-25.tmp, bss-150.tmp e bss-500.tmp
Paralelo (2 threads)	bjp2	bjp2.out	bsp-25.2.tmp, bsp-150.2.tmp e bsp-500.2.tmp
Paralelo (4 threads)	bjp4	bjp4.out	bsp-25.4.tmp, bsp-150.4.tmp e bsp-500.4.tmp
Paralelo (8 threads)	bjp8	bjp8.out	bsp-25.8.tmp, bsp-150.8.tmp e bsp-500.8.tmp
Paralelo (16 threads)	bjp16	bjp16.out	bsp-25.16.tmp, bsp-150.16.tmp e bsp-500.16.tmp

Vencimento: quinta, 29 Set 2022, 19:15

Trabalho 1: Bubble Sort (OpenMP)

Batchjob 1 - Sequencial: nano bjs

```
#!/bin/bash
#SBATCH --export=ALL
#SBATCH -N 1
#SBATCH -t 300
#SBATCH --exclusive
#SBATCH --no-requeue
#SBATCH -o bjs.out
#SBATCH -D /home/pp03004/trab_i
echo bss-25
echo Initial Time is `date`
./bss-25 > ./tmp/bss-25.tmp
echo Final Time is `date`
echo bss-150
echo Initial Time is `date`
./bss-150 > ./tmp/bss-150.tmp
echo Final Time is `date`
echo bss-500
echo Initial Time is `date`
./bss-500 > ./tmp/bss-500.tmp
echo Final Time is `date`
```

Batchjob 2 - Paralelo (2 threads): nano bjp2

```
#!/bin/bash
#SBATCH --export=ALL
#SBATCH -N 1
#SBATCH -t 300
#SBATCH --exclusive
#SBATCH --no-requeue
#SBATCH -o bjp2.out
#SBATCH -D /home/pp03004/trab_i
echo bsp-25.2
echo Initial Time is `date`
./bsp-25.2 > ./tmp/bsp-25.2.tmp
echo Final Time is `date`
echo bsp-150.2
echo Initial Time is `date`
./bsp-150.2 > ./tmp/bsp-150.2.tmp
echo Final Time is `date`
echo bsp-500.2
echo Initial Time is `date`
./bsp-500.2 > ./tmp/bsp-500.2.tmp
echo Final Time is `date`
```

Vencimento: quinta, 29 Set 2022, 19:15

Batchjob 3 - Paralelo (4 threads): nano bjp4

```
#!/bin/bash
#SBATCH --export=ALL
#SBATCH -N 1
#SBATCH -t 300
#SBATCH --exclusive
#SBATCH --no-requeue
#SBATCH -o bjp4.out
#SBATCH -D /home/pp03004/trab_i
echo bsp-25.4
echo Initial Time is `date`
./bsp-25.4 > ./tmp/bsp-25.4.tmp
echo Final Time is `date`
echo bsp-150.4
echo Initial Time is `date`
./bsp-150.4 > ./tmp/bsp-150.4.tmp
echo Final Time is `date`
echo bsp-500.4
echo Initial Time is `date`
./bsp-500.4 > ./tmp/bsp-500.4.tmp
echo Final Time is `date`
```

Batchjob 4 - Paralelo (8 threads): nano bjp8

```
#!/bin/bash
#SBATCH --export=ALL
#SBATCH -N 1
#SBATCH -t 300
#SBATCH --exclusive
#SBATCH --no-requeue
#SBATCH -o bjp8.out
#SBATCH -D /home/pp03004/trab_i
echo bsp-25.8
echo Initial Time is `date`
./bsp-25.8 > ./tmp/bsp-25.8.tmp
echo Final Time is `date`
echo bsp-150.8
echo Initial Time is `date`
./bsp-150.8 > ./tmp/bsp-150.8.tmp
echo Final Time is `date`
echo bsp-500.8
echo Initial Time is `date`
./bsp-500.8 > ./tmp/bsp-500.8.tmp
echo Final Time is `date`
```

Trabalho 1: Bubble Sort (OpenMP)

Batchjob 5 - Paralelo (16 threads): nano bjp16

```
#!/bin/bash
#SBATCH --export=ALL
#SBATCH -N 1
#SBATCH -t 300
#SBATCH --exclusive
#SBATCH --no-requeue
#SBATCH -o bjp16.out
#SBATCH -D /home/pp03004/trab_i
echo bsp-25.16
echo Initial Time is `date`
./bsp-25.16 > ./tmp/bsp-25.16.tmp
echo Final Time is `date`
echo bsp-150.16
echo Initial Time is `date`
./bsp-150.16 > ./tmp/bsp-150.16.tmp
echo Final Time is `date`
echo bsp-500.16
echo Initial Time is `date`
./bsp-500.16 > ./tmp/bsp-500.16.tmp
echo Final Time is `date`
```

Ao final, temos em nosso diretório os seguintes arquivos:

```
pp03004@grad: /home/pp03004/trab_i
pp03004@grad:~/trab_i$ ls
bjp16  bjp4  bjs         bsp-150.2  bsp-150.8  bsp-25.2  bsp-25.8    bsp-500.2  bsp-500.8  bss-150  bss-500  out
bjp2   bjp8  bsp-150.16  bsp-150.4  bsp-25.16  bsp-25.4  bsp-500.16  bsp-500.4  bsp.c      bss-25   bss.c    tmp
```

Submeter cada batchjob criado: sbatch “nomeDoBatchjob”

Verificar andamento da fila: squeue

```
pp03004@grad:~/trab_i$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4813 LAD_geral testepar  pp03007 PD       0:00     16 (PartitionNodeLimit)
              4814 LAD_geral testepar  pp03007 PD       0:00     16 (PartitionNodeLimit)
              4919 LAD_geral      bjs  pp03004  R       0:20      1 grad01
              4920 LAD_geral     bjp2  pp03004  R       0:13      1 grad02
              4921 LAD_geral     bjp4  pp03004  R       0:10      1 grad03
              4922 LAD_geral     bjp8  pp03004  R       0:07      1 grad04
              4923 LAD_geral    bjp16  pp03004  R       0:03      1 grad05
```

Caso queira cancelar algum processo que foi executado com defeito: scancel jobID

Vencimento: quinta, 29 Set 2022, 19:15