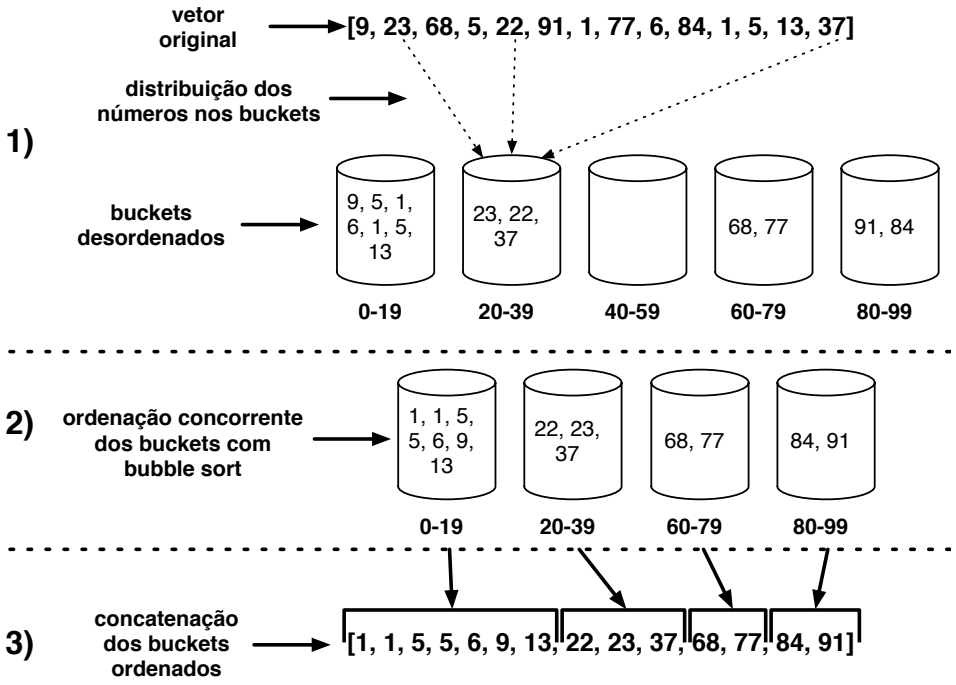
Trabalho 2 – Bucket Sort Distribuído

INE5410 - Programação Concorrente
Prof. Márcio Castro

2015/1

1 Definição

O *bucket sort* é um algoritmo de ordenação no qual o vetor a ser ordenado é dividido em um número finito de recipientes, denominados *buckets*. Cada *bucket* é então ordenado individualmente através de um algoritmo de ordenação (nesse trabalho utilizaremos o algoritmo *bubble sort*). A Figura 1 mostra o funcionamento do *bucket sort*, o qual é composto por 3 fases: 1) distribuição dos números do vetor desordenado nos *buckets*; 2) ordenação individual dos *buckets* utilizando o algoritmo *bubble sort*; e 3) redistribuição dos *buckets* ordenados no vetor original.



No exemplo mostrado acima considerou-se somente números inteiros positivos na faixa $[0; 99]$ os quais foram divididos em 5 *buckets*, cada um armazenando números em uma faixa contendo 20 valores possíveis ($[0, 19], [20, 39], ..., [80, 99]$). Note que um dos *buckets* ficou vazio, pois não havia nenhum número pertencente ao intervalo $[40, 59]$. *Buckets* vazios ou contendo somente 1 elemento não deverão ser considerados durante os passos 2) e 3).

2 Trabalho

O segundo trabalho da disciplina de Programação Concorrente consiste em implementar uma versão **distribuída** do *bucket sort* utilizando a linguagem C e a biblioteca MPI. Os seguintes parâmetros de entrada do programa **deverão ser informados na linha de comando**:

- O tamanho do vetor (*tamvet*) a ser ordenado;
- O número de *buckets* (*nbuckets*);
- O número de *processos MPI* (*nprocs*) .

A faixa de valores possíveis no vetor de entrada será sempre $[0; tamvet - 1]$. Por exemplo, um vetor de entrada de tamanho 10 ($tamvet = 10$) poderá conter somente números dentro da faixa $[0, 9]$ enquanto um vetor de entrada de tamanho 30 ($tamvet = 30$) poderá conter somente números dentro da faixa $[0, 29]$.

As faixas de números permitidos em cada *bucket* deverão ser determinadas durante a execução do programa em função do tamanho do vetor de entrada ($tamvet$) e do número de *buckets* ($nbuckets$):

- **Sempre que possível**, as faixas de números permitidos em cada *bucket* deverão ter o **mesmo tamanho**. Por exemplo, para $tamvet = 50$ e $nbuckets = 5$ cada *bucket* deverá permitir números dentro de uma faixa de 10 números ($50/5 = 10$). Logo, os intervalos seriam $[0, 9]$, $[10, 19]$, $[20, 29]$, $[30, 39]$ e $[40, 49]$.
- **Nos casos onde a divisão não é inteira**, as faixas de números permitidos em cada *bucket* deverão ter **uma diferença de tamanho de no máximo 1 número**. Por exemplo, para $tamvet = 20$ e $nbuckets = 8$ teríamos $20/8 = 2$ com resto igual à 4. Nesse caso, 4 *buckets* teriam intervalos de 3 elementos e 4 *buckets* teriam intervalos de 2 elementos. Um exemplo de intervalos para esse caso seria: $[0; 2]$, $[3; 5]$, $[6; 8]$, $[9; 11]$, $[12; 13]$, $[14; 15]$, $[16; 17]$ e $[18; 19]$.

O *bucket sort* **distribuído** deverá funcionar de acordo com a estratégia de paralelização ***mestre/escravo***, havendo 1 *processo mestre* e $nprocs - 1$ *processos escravos*. O funcionamento dos processos é descrito abaixo:

- O *processo mestre* inicializa o vetor com valores aleatórios de acordo com o número de elementos especificado no parâmetro de entrada ($tamvet$). As funções `srand()` e `rand()` deverão ser utilizadas para gerar números aleatórios dentro do intervalo $[0; tamvet - 1]$. Em seguida o *processo mestre* distribui os elementos do vetor desordenado nos seus respectivos *buckets*, de acordo com o número de *buckets* especificado ($nbuckets$). Os *buckets* deverão ser identificados com valores no intervalo $[0, nbuckets - 1]$. Então, o *processo mestre* realiza a distribuição inicial dos *buckets* desordenados para os *processos escravos* (um *bucket* para cada *processo escravo*) e aguarda as respostas (**atenção:** *buckets* vazios ou que contenham apenas 1 elemento **não deverão ser enviados para os escravos**). Ao receber uma resposta de um *processo escravo*, o *processo mestre* reenvia um novo *bucket* desordenado ao *processo escravo* que respondeu. Esse processo é repetido até que não existam mais *buckets* a serem ordenados. Ao final, o *processo mestre* envia uma mensagem de finalização para os *processos escravos*. Então, o *processo mestre* redistribui os elementos ordenados dos *buckets* no vetor original, sobreescrevendo os valores antigos armazenados no vetor.
- Os *processos escravos* aguardam que o *processo mestre* envie uma tarefa contendo um *bucket* desordenado. Ao receber um *bucket* desordenado, o *processo escravo* realiza a ordenação dos elementos dentro do *bucket* com auxílio do *bubble sort*. Após ordenar o *bucket*, o *processo escravo* envia uma resposta para o mestre contendo o *bucket* ordenado, e então, volta a aguardar uma nova tarefa. Ao receber uma mensagem de finalização, o *processo escravo* termina sua execução.

O programa deverá funcionar em todos os casos, independentemente do tamanho do vetor, número de *buckets* ou processos MPI, exceto nos seguintes casos: quando o número de processos MPI for menor que 2 (ou seja, é necessário ao menos 1 *processo mestre* e 1 *processo escravo*) ou quando o número de *buckets* for maior que o tamanho do vetor. Nesses casos, o programa deverá informar um erro ao usuário.

2.1 Saída

O *processo mestre* deverá **obrigatoriamente** imprimir na tela (XX e YY são o identificador do *bucket* e *rank* do *processo escravo*, respectivamente):

- O vetor desordenado;
- Mensagens *"Mestre ENVIOU bucket XX para Escravo YY"*, sempre que o *processo mestre* **enviar** um *bucket* XX desordenado para um *processo escravo* YY;
- Mensagens *"Mestre RECEBEU bucket XX para Escravo YY"*, sempre que o *processo mestre* **receber** um *bucket* XX ordenado de um *processo escravo* YY;
- O vetor ordenado.

Um exemplo de saída válido é mostrado no final do documento.

3 Grupos, Avaliação e Entrega

O trabalho deverá ser realizado **necessariamente** em grupos de **2 alunos**. Os alunos deverão apresentar o trabalho ao professor assim como mostrar sua solução em funcionamento. As apresentações serão feitas durante as aulas nos seguintes dias:

- **09/06/2015:** Grupos A ao G (demais grupos estão liberados desta aula e terão presença confirmada).
- **11/06/2015:** Grupos H ao O (demais grupos estão liberados desta aula e terão presença confirmada).

Pelo menos um dos integrantes de cada grupo deverá enviar através do Moodle um arquivo contendo o código fonte em C da solução para o trabalho. A data/hora limite para o envio dos trabalhos é **07/06/2015 às 23h55min**. **Não será permitida a entrega de trabalhos fora desse prazo.**

O professor irá avaliar não somente a corretude mas também o desempenho e a clareza da solução. Além disso, os alunos serão avaliados pela apresentação e entendimento do trabalho. **A implementação e apresentação valerão 40% e 60% da nota do trabalho, respectivamente**.

4 Bubble Sort

Os grupos poderão utilizar a implementação do algoritmo *bubble sort* mostrada a seguir para ordenar os elementos dentro dos *buckets*. Os parâmetros v e tam correspondem ao vetor a ser ordenado e o seu tamanho, respectivamente.

```
1  void bubble_sort(int *v, int tam){
2      int i, j, temp, trocou;
3      for(j = 0; j < tam - 1; j++){
4          trocou = 0;
5          for(i = 0; i < tam - 1; i++){
6              if(v[i + 1] < v[i]){
7                  temp = v[i];
8                  v[i] = v[i + 1];
9                  v[i + 1] = temp;
10                 trocou = 1;
11             }
12         }
13         if(!trocou) break;
14     }
15 }
```

5 Exemplo de Saída

Exemplo de uma execução com $tamvet = 64$, $nbuckets = 32$ e $nprocs = 8$:

```
6 15 28 14 43 7 23 49 27 40 26 49 35 33 4 62 53 60 61 20 10 48 36 18 38 28 32 29 36 16 3 42 31 31 56 10 38 16 59 2 56
22 51 27 55 56 25 44 52 23 1 63 7 37 17 45 1 49 10 37 2 13 16 33
Mestre ENVIOU bucket 0 para Escravo 1
Mestre ENVIOU bucket 1 para Escravo 2
Mestre ENVIOU bucket 2 para Escravo 3
Mestre ENVIOU bucket 3 para Escravo 4
Mestre ENVIOU bucket 5 para Escravo 5
Mestre ENVIOU bucket 6 para Escravo 6
Mestre ENVIOU bucket 7 para Escravo 7
Mestre RECEBEU bucket 0 do Escravo 1
Mestre ENVIOU bucket 8 para Escravo 1
Mestre RECEBEU bucket 1 do Escravo 2
Mestre ENVIOU bucket 9 para Escravo 2
Mestre RECEBEU bucket 2 do Escravo 3
Mestre ENVIOU bucket 10 para Escravo 3
Mestre RECEBEU bucket 3 do Escravo 4
Mestre ENVIOU bucket 11 para Escravo 4
Mestre RECEBEU bucket 5 do Escravo 5
```

Mestre ENVIOU bucket 12 para Escravo 5
Mestre RECEBEU bucket 9 do Escravo 2
Mestre ENVIOU bucket 13 para Escravo 2
Mestre RECEBEU bucket 6 do Escravo 6
Mestre ENVIOU bucket 14 para Escravo 6
Mestre RECEBEU bucket 8 do Escravo 1
Mestre ENVIOU bucket 15 para Escravo 1
Mestre RECEBEU bucket 10 do Escravo 3
Mestre ENVIOU bucket 16 para Escravo 3
Mestre RECEBEU bucket 11 do Escravo 4
Mestre ENVIOU bucket 17 para Escravo 4
Mestre RECEBEU bucket 12 do Escravo 5
Mestre ENVIOU bucket 18 para Escravo 5
Mestre RECEBEU bucket 13 do Escravo 2
Mestre ENVIOU bucket 19 para Escravo 2
Mestre RECEBEU bucket 15 do Escravo 1
Mestre ENVIOU bucket 20 para Escravo 1
Mestre RECEBEU bucket 16 do Escravo 3
Mestre ENVIOU bucket 21 para Escravo 3
Mestre RECEBEU bucket 14 do Escravo 6
Mestre ENVIOU bucket 22 para Escravo 6
Mestre RECEBEU bucket 17 do Escravo 4
Mestre ENVIOU bucket 24 para Escravo 4
Mestre RECEBEU bucket 18 do Escravo 5
Mestre ENVIOU bucket 25 para Escravo 5
Mestre RECEBEU bucket 19 do Escravo 2
Mestre ENVIOU bucket 26 para Escravo 2
Mestre RECEBEU bucket 20 do Escravo 1
Mestre ENVIOU bucket 27 para Escravo 1
Mestre RECEBEU bucket 21 do Escravo 3
Mestre ENVIOU bucket 28 para Escravo 3
Mestre RECEBEU bucket 22 do Escravo 6
Mestre ENVIOU bucket 29 para Escravo 6
Mestre RECEBEU bucket 7 do Escravo 7
Mestre ENVIOU bucket 30 para Escravo 7
Mestre RECEBEU bucket 24 do Escravo 4
Mestre ENVIOU bucket 31 para Escravo 4
Mestre RECEBEU bucket 25 do Escravo 5
Mestre RECEBEU bucket 26 do Escravo 2
Mestre RECEBEU bucket 27 do Escravo 1
Mestre RECEBEU bucket 28 do Escravo 3
Mestre RECEBEU bucket 29 do Escravo 6
Mestre RECEBEU bucket 30 do Escravo 7
Mestre RECEBEU bucket 31 do Escravo 4
1 1 2 2 3 4 6 7 7 10 10 10 13 14 15 16 16 16 17 18 20 22 23 23 25 26 27 27 28 28 29 31 31 32 33 33 35 36 36 37 37 38 38 40 42 43 44 45 48 49 49 49 51 52 53 55 56 56 56 59 60 61 62 63