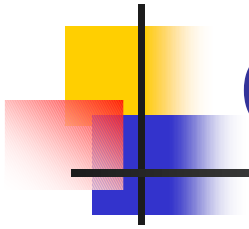
Algoritmos Paralelos BSP/CGM

Edson Norberto Cáceres e Siang Wun Song
DCT/UFMS e DCC/IME/USP



Ordenação – SplitSort

// **Passo 1.** Distribua os dados para as tarefas

// **Passo 2.** Calcule as p-1 separatrizes (quartil)

// **Passo 3.** Envie os p-1 quartis para a tarefa 0

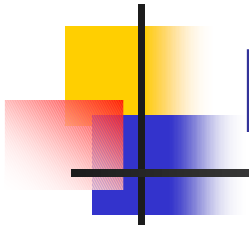
// **Passo 4.** Receba os p-1 quartis e calcule o conjunto S (p-1 quartis dos elementos recebidos tarefa 0)

// **Passo 5.** Envie S a todas as tarefas

// **Passo 6.** Calcule os buckets

// **Passo 7.** Envie os buckets

// **Passo 8.** Ordene dos buckets



Passo 1 - Inicialização

```
// Passo 1. Inicialização
  MPI_Init(&argc, &argv);
  // número de tarefas
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  // identificação da tarefa
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  // tamanho do subvetor
  tam = (int) TAMMAX/size;
```

Passo 2 – Distribuição dos Dados

```
// vetor de dados
int VetorDados[16] =
                {3,20,51,12,35,7,10,40,4,23,1,9,6,8,21,2};

// Passo 2. Distribua os dados entre as tarefas
  MPI_Scatter(VetorDados, tam, MPI_INT, SubVetor,
   tam, MPI_INT, root, MPI_COMM_WORLD);
// rank 0 e SubVetor[0-3] = [3, 20, 51, 12]
// rank 1 e SubVetor[0-3] = [35, 7, 10, 40]
// rank 2 e SubVetor[0-3] = [4, 23, 1, 9]
// rank 3 e SubVetor[0-3] = [6, 8, 21, 2]
```

Passo 3 - Quartis

```
// Passo 3. Calcule os p-1 quartis
  size_e = sizeof(int);
  qsort(SubVetor, tam, size_e, Compar);
for (i = 1; i < size; i++)
      Quartis[i-1] = (SubVetor[((i*tam)/size)-

 1]+SubVetor[((i*tam)/size)])/2;
```

Passo 3 - Quartis

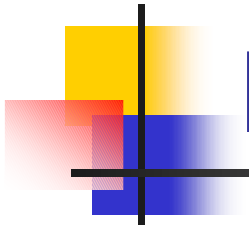
// rank 0 e SubVetor[0-3] = [3, 12, 20, 51]
// Quartis[0-2] = [7, 16, 35]
// rank 1 e SubVetor[0-3] = [7, 10, 35, 40]
// Quartis[0-2] = [8, 22, 37]
// rank 2 e SubVetor[0-3] = [1, 4, 9, 23]
// Quartis[0-2] = [2, 6, 16]
// rank 3 e SubVetor[0-3] = [2, 6, 8, 21]
// Quartis[0-2] = [4, 7, 14]

Passo 4 – Enviar as Separatrizes

- **Todas as tarefas enviam os dados para a tarefa root (com recebimento):**

```
MPI_Gather(void* sendbuf, int sendcount,
                MPI_Datatype sendtype, void* recvbuf,
                int recvcount, MPI_Datatype recvtype,
                int root, MPI_Comm comm);
```

```
// Passo 4. Envie os p-1 quartis para a tarefa 0
  rbuf = (int *)malloc(size*(size-1)*sizeof(int));
  MPI_Gather(Quartis, size-1, MPI_INT, rbuf, size-1,
                MPI_INT, root, MPI_COMM_WORLD);
```



Passo 4 – P_0 recebe os Quartis

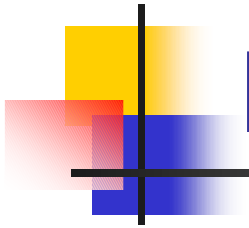
/ / rank 0

/ / rbuf[0-11] = [7, 16, 35, 8, 22, 37, 2, 6, 16, 4, 7, 14]

Passo 5 – Computar o conjunto S

```
// Passo 5. Calcule o conjunto S – P0
  S = (int *)malloc(size*sizeof(int));
  if (rank == root) {
    tamrbuf = size*(size-1);
    qsort(rbuf, tamrbuf, size_e, Compar);
    for (i = 1; i < size; i++)
      S[i-1] = (rbuf[((i*tamrbuf)/size)-1] +

   rbuf[((i*tamrbuf)/size)])/2;
      S[size-1] = 10000;}
  free(rbuf);
```



Passo 5 – P_0 computa S

// rank 0

// rbuf[0-11] = [2, 4, 6, 7, 7, 8, 14, 16, 16, 22, 35, 37]

// S[0-2] = [6, 11, 16]

Passo 6 – Enviar S para as tarefas

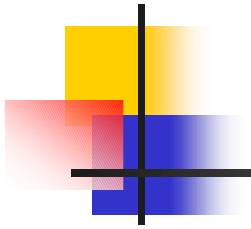
```
// Passo 6. Envie S a todas as tarefas
MPI_Bcast(S, size, MPI_INT, root,

MPI_COMM_WORLD);
```

Passo 7 – Calcular os Buckets

```
// Passo 7. Calcula os buckets
  sbuf = (int *)malloc(size*sizeof(int));
  send_displ =  (int *)malloc(size*sizeof(int));
  j = 0;   k = 1;
  for (i = 0; i < tam; i++) {
     if (SubVetor[i] < S[j])  k++;
     else { sbuf[j] = k-1;  j++; k = 1; i--; } }
  sbuf[j] = k-1;
  send_displ[0] = 0;
  for (i = 1; i < size; i++)
    send_displ[i] = send_displ[i-1] + sbuf[i-1];
  free(S);
```

Passo 7 – Calcular os Buckets



rank 0 e SubVetor[0-3] = [3, 12, 20, 51]
rank 1 e SubVetor[0-3] = [7, 10, 35, 40]
rank 2 e SubVetor[0-3] = [1, 4, 9, 23]
rank 3 e SubVetor[0-3] = [2, 6, 8, 21]
S[0-2] = [6, 11, 16]
rank 0 sbuf[0-3] = [1, 0, 1, 2]
send_displ[0] = [0, 1, 1, 2]
rank 1 sbuf[0-3] = [0, 2, 0, 2]
send_displ[0-3] = [0, 0, 2, 2]
rank 2 sbuf[0-3] = [2, 1, 0, 1]
send_displ[0-3] = [0, 2, 3, 3]
rank 3 sbuf[0-3] = [1, 2, 0, 1]
send_displ[0-3] = [0, 1, 3, 3]

Passo 8 – Troque os Buckets

- **Enviar o tamanho dos buckets para todas as tarefas e receber o tamanho dos buckets:**

```
MPI_Alltoall(void* sendbuf, int sendcount,
             MPI_Datatype sendtype, void* recvbuf,
             int recvcount, MPI_Datatype recvtype,
             MPI_Comm comm);
```

```
// Passo 8.1. Envie (receba) o tamanho dos buckets
  rbuf =  (int *)malloc(size*sizeof(int));
  rec_displ =  (int *)malloc(size*sizeof(int));
  MPI_Alltoall(sbuf, 1, MPI_INT, rbuf, 1, MPI_INT,
                                  MPI_COMM_WORLD);
```

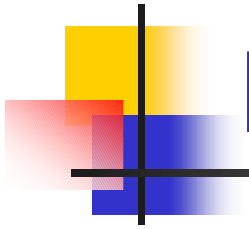
Passo 8 – No. Element. Recebidos

// rank 0
// rbuf[0-3] = [1, 0 ,2, 1] rec_displ[0-3] = [0, 1, 1, 3]
// rank 1
// rbuf[0-3] = [0, 2, 1, 2] rec_displ[0-3] = [0, 0, 2, 3]
// rank 2
// rbuf[0-3] = [1, 0, 0, 0] rec_displ[0-3] = [0, 1, 1, 1]
// rank 3
// rbuf[0-3] = [2, 2, 1, 1] rec_displ[0-3] = [0, 2, 4, 5]

Passo 8 – Troque os Buckets

- **Enviar o tamanho dos buckets para todas as tarefas e receber o tamanho dos buckets:**

```
MPI_Alltoallv(void* sendbuf, int *sendcount, int *sdispls,
        MPI_Datatype sendtype, void* recvbuf,
        int *recvcount, int *recvdispls,
        MPI_Datatype recvtype, MPI_Comm comm);
// Passo 8.2. Envie (receba) os buckets
rec_displ[0] = 0;
  for (i = 1; i < size; i++)
    rec_displ[i] = rec_displ[i-1] + rbuf[i-1];
  MPI_Alltoallv(SubVetor, sbuf, send_displ,
                MPI_INT, BuckVetor, rbuf, rec_displ,
                MPI_INT, MPI_COMM_WORLD);
  free(sbuf); free(send_displ);  free(rec_displ);
```



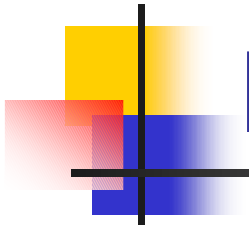
Passo 9 – Ordene os Buckets

```
// Passo 9. Ordene dos Buckets
  tamB = 0;
  for (i = 0; i < size; i++)
     tamB = tamB + rbuf[i];
  qsort(BuckVetor, tamB, size_e, Compar);
  free(rbuf);
```

Passo 10 – Imprima o vetor

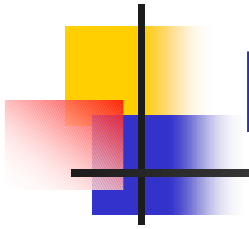
```
// Passo 10. Imprima o vetor Ordenado
  for (i = 0; i < tamB; i++)
      printf("rank %d e BuckVetor[%d]:
   %d\n",rank, i,

   BuckVetor[i]);
```



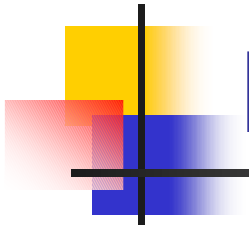
Passo 10 – Imprima o vetor

```
// rank 0
// BuckVetor[0-3] = [1, 2, 3, 4]
// rank 1
// BuckVetor[0-4] = [6, 7, 8, 9, 10]
// rank 2
// BuckVetor[0] = [12]
// rank 3
// BuckVetor[0-5] = [20, 21, 23, 35, 40, 51]
```



Passo 11 – Finalize o MPI

```
// Passo 11. Finalize o MPI
  MPI_Finalize();
  return 0;
```



Funções

```
// Implementação das funções

int Compar(const int *Lhs, const int *Rhs)
{
  int Leftkey = *Lhs;
  int Rightkey = *Rhs;

  return Leftkey < Rightkey ? -1 : Leftkey !=
   Rightkey;
} // fim função Compar
```

E ....