Programação Paralela e Distribuída
Programação Paralela com MPI – Modelo Divisão e Conquista
Exercício – *Merge Sort*

O objetivo do trabalho é implementar usando a biblioteca MPI uma versão paralela usando o modelo divisão e conquista do algoritmo de ordenação *Merge Sort*. Após implementado, o programa deve ser executado no cluster para realização das medições de desempenho e, por fim, deve-se gerar o gráfico de *speed up*.

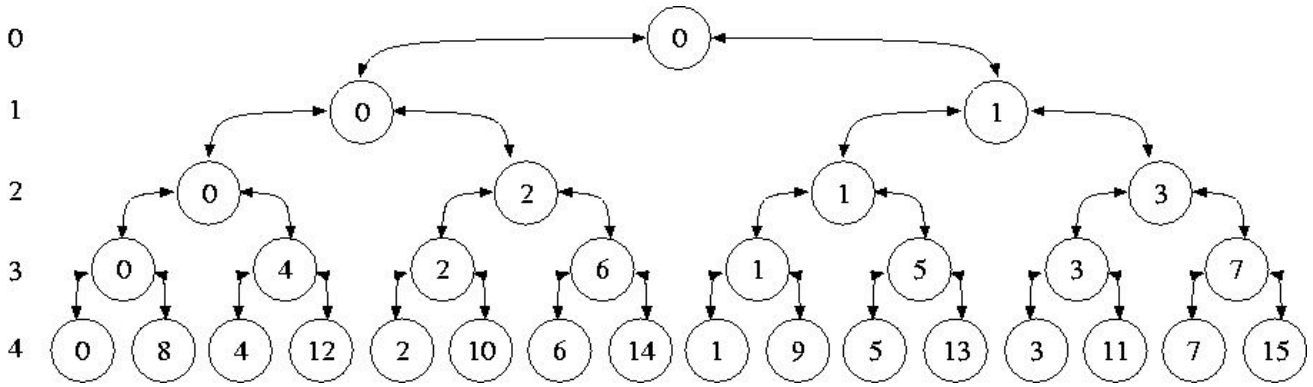
O algoritmo *Merge Sort* funciona da seguinte forma: "A lista desordenada é primeiramente dividida pela metade. Cada metade é então dividida pela metade novamente, e este processo continua até que números individuais sejam obtidos. Então pares de números são combinados (*merged*) em listas ordenadas de dois números. Pares destas listas de 4 números são combinadas em listas ordenadas de 8 números. Este processo continua até que seja obtida a lista ordenada completa."

A versão sequencial do algoritmo está apresentada abaixo:

```
void Sort(int array[], int begin, int end)
{
   int mid;
   if (begin == end)
     return;
   if (begin == end - 1)
     return;
   mid = (begin + end) / 2;
   Sort(array, begin, mid);
   Sort(array, mid, end);
   Merge(array, begin, mid, end);
}
```

A versão paralela usando divisão e conquista consiste em realizar divisões do vetor de entrada até que cada processo possua um vetor de tamanho (N/P), onde N é o tamanho do vetor inicial e P o número de processos. Quando a divisão estiver completa, cada processo realiza a ordenação do pedaço do vetor local (usando o *Merge Sort* sequencial) e envia o vetor ordenado para o processo pai. Quando o processo pai recebe os vetores ordenados dos processos filhos, ele realiza a operação de *Merge Sort* e reenvia o vetor resultado ordenado para o seu processo pai. Este processo é repetido até chegar no processo que iniciou a execução, o qual realiza o *Merge Sort* e obtém o vetor resultado final.

Com o objetivo de minimizar o número de processos que ficam ociosos durante a execução, ao invés de um processo dividir um vetor e enviar metade para dois processos filhos diferentes, ele guarda metade para ser processada por ele mesmo, e a outra metade é enviada para um novo processo. A figura abaixo apresenta a árvore gerada para 16 processos.



O cálculo do identificador dos "processos-filhos", no qual um processo deverá enviar as metades do vetor é obtida da seguinte forma: o processo *i* guarda uma metade do vetor para ser processada localmente e envia a outra metade para o processo $i+2^k$, onde k é a profundidade da árvore. Por exemplo, o processo 6, na iteração de profundidade 3, guarda metade do vetor e envia a outra metade para o processo $(6+2^3)$, isto é, para o processo 14.

O programa deverá receber como parâmetros de entrada um número N, representando o número de valores a ser testado e o nome de um arquivo, que contém a lista de valores inteiros a serem ordenados. Serão fornecidos 3 casos de teste para realização das medições no cluster. A saída que deve ser gerada é a lista ordenada (crescente) dos valores de entrada (1 valor por linha) e também o tempo de execução da aplicação.

Os itens para avaliação são:

- implementação da versão paralela do algoritmo (*Merge Sort*);
- medição dos tempos de execução para as versões sequencial e paralelas (usando 2, 4, 8 e 16 processos) para os 3 casos de teste;
- cálculo do *speed up* de cada versão paralela para cada caso de teste;
- geração do gráfico de *speed up* para cada versão paralela com os 3 casos de teste;
- otimização do código;
- clareza do código (utilização de comentários e nomes de variáveis adequadas).

PS: A nota do Trabalho 2 será uma composição das notas dos exercícios que serão realizados em aula. **O exercício pode ser feito em duplas.**