Adriano
Adriyel
Amon

Executado em maquina t3.mini na aws usando linux ubuntu

Problema :

Grupo de três pessoas. Entrega de relatório, seminário e programa.

O programa abaixo resolve a multiplicação, adição e subtração dos elementos de um vetor de inteiros:

```
#include <stdio.h>
int main ()
{
int x, soma=0, subtracao=0, mult=1, TAM = 1000;
int vet[TAM];
for (x=0;x<10;x++) {
vet[x] = x + 1;
}
for (x=0;x<10;x++) {
printf("vet[%d] = %d\n", x, vet[x]);
}
for (x=0;x<10;x++) {
soma = soma + vet[x];
subtracao = subtracao - vet[x];
mult = mult * vet[x];
}
printf("Soma = %d\n", soma);
printf("Subtracao = %d\n", subtracao);
printf("Multiplicacao = %d\n", mult);
}
```

A variável “vet” representa vetor sobre o qual as operações serão realizadas. Então, modifique o programa para executar em paralelo utilizando MPI.

Apresente
as primitivas utilizadas (comentando o código sempre que julgar necessário).

Faça
as seguintes versões MPI:

a) uma versão com apenas 4 processos executando. Nesta versão, cada processo faz uma função: 1 soma, 1 subtrai e 1 multiplica. O outro processo é responsável por avisar cada um dos outros 3 a sua função, e ao término imprimir os resultados.

b) uma versão com MPI com qualquer número de processos baseado em mestre/escravo, utilizando operações de comunicação coletiva. A divisão das tarefas deve ser a mais balanceada possível, e a
forma de implementação é livre.

c) uma versão com MPI baseada no modelo Pipeline.

Para todos os casos, apresente o resultado final, o tempo de execução, o speed up e a eficiência.

```
// mpi_version_a.c  (4 processos: 0=coordenador; 1=soma; 2=subtracao; 3=produto)
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
   MPI_Init(&argc,&argv);
   int p, rank;
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   const int N = 10;
   int vet[N];
   int soma=0, subtracao=0, mult=1;

   // Para speedup/eficiência: mede tempo serial na rank 0
   double tser_start=0.0, tser_end=0.0, Tserial=0.0;
   if (rank==0) {
      for (int i=0;i<N;i++) vet[i] = i+1;

      tser_start = MPI_Wtime();
      int s=0, m=1;
      for (int i=0;i<N;i++){ s += vet[i]; m *= vet[i]; }
      int sub = -s; // por definição do seu programa
      tser_end = MPI_Wtime();
      Tserial = tser_end - tser_start;

      // Apenas para conferir, mas o resultado oficial virá da execução paralela
      // printf("[SERIAL] soma=%d sub=%d mult=%d\n", s, sub, m);
   }

   // --- Paralelo (exactamente 4 processos)
   if (p != 4) {
      if (rank==0) {
         printf("Esta versao requer exatamente 4 processos (mpirun -np 4)!\n");
      }
      MPI_Finalize();
      return 0;
   }

   // Tags
   const int TAG_FUNC = 10;
   const int TAG_VET  = 11;
   const int TAG_RES  = 12;

   double tpar_start=0.0, tpar_end=0.0;
```

```
  if (rank==0) {
     // rank0 envia qual funcao cada processo vai executar
     int func1=1, func2=2, func3=3;
     MPI_Send(&func1,1,MPI_INT,1,TAG_FUNC,MPI_COMM_WORLD);
     MPI_Send(&func2,1,MPI_INT,2,TAG_FUNC,MPI_COMM_WORLD);
     MPI_Send(&func3,1,MPI_INT,3,TAG_FUNC,MPI_COMM_WORLD);

     // envia o vetor para cada worker
     tpar_start = MPI_Wtime();
     MPI_Send(vet,N,MPI_INT,1,TAG_VET,MPI_COMM_WORLD);
     MPI_Send(vet,N,MPI_INT,2,TAG_VET,MPI_COMM_WORLD);
     MPI_Send(vet,N,MPI_INT,3,TAG_VET,MPI_COMM_WORLD);

     // recebe os resultados
     int res_s, res_sub, res_mul;
     MPI_Recv(&res_s, 1, MPI_INT, 1, TAG_RES, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
     MPI_Recv(&res_sub,1, MPI_INT, 2, TAG_RES, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
     MPI_Recv(&res_mul,1, MPI_INT, 3, TAG_RES, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
     tpar_end = MPI_Wtime();

     soma = res_s;
     subtracao = res_sub;
     mult = res_mul;

     double Tpar = tpar_end - tpar_start;
     double speedup = (Tserial>0)? (Tserial / Tpar) : 0.0;
     double eficiencia = speedup / p;

     printf("=== Versao (a) 4 processos ===\n");
     printf("Soma = %d\n", soma);
     printf("Subtracao = %d\n", subtracao);
     printf("Multiplicacao = %d\n", mult);
     printf("Tempo serial   = %.6f s\n", Tserial);
     printf("Tempo paralelo = %.6f s\n", Tpar);
     printf("Speedup        = %.4f\n", speedup);
     printf("Eficiencia     = %.4f\n", eficiencia);
  } else {
     // workers
     int func;

MPI_Recv(&func,1,MPI_INT,0,TAG_FUNC,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
     MPI_Recv(vet,N,MPI_INT,0,TAG_VET,MPI_COMM_WORLD,MPI_STATUS_IGNORE);

     int result= (func==3? 1:0); // mult inicia em 1; soma/sub em 0
     if (func==1) { // soma
        for (int i=0;i<N;i++) result += vet[i];
     } else if (func==2) { // subtracao: 0 - sum(vet)
        for (int i=0;i<N;i++) result -= vet[i];
```

```
    } else if (func==3) { // multiplicacao
        for (int i=0;i<N;i++) result *= vet[i];
    }
    MPI_Send(&result,1,MPI_INT,0,TAG_RES,MPI_COMM_WORLD);
  }

  MPI_Finalize();
  return 0;
}
```

```
// mpi_version_b.c  (mestre/escravo; coletivas; qualquer p>=1)
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
  MPI_Init(&argc,&argv);
  int p, rank;
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  const int N = 10;
  int vet[N];
  if (rank==0) {
    for (int i=0;i<N;i++) vet[i] = i+1;
  }

  // Tempo serial na rank 0
  double Tserial=0.0;
  if (rank==0){
    double t0 = MPI_Wtime();
    int s=0, m=1;
    for (int i=0;i<N;i++){ s+=vet[i]; m*=vet[i]; }
    int sub = -s;
    double t1 = MPI_Wtime();
    Tserial = t1 - t0;
    // printf("[SERIAL] soma=%d sub=%d mult=%d\n", s, sub, m);
  }

  // Broadcast de N (aqui constante, mas ilustrativo)
  int n_global = N;
  MPI_Bcast(&n_global,1,MPI_INT,0,MPI_COMM_WORLD);

  // Prepara Scatterv (divisão balanceada)
  int counts[p], displs[p];
  int base = n_global / p, rem = n_global % p;
  for (int i=0, off=0; i<p; ++i) {
    counts[i] = base + (i < rem ? 1 : 0);
    displs[i] = off;
    off += counts[i];
  }
  int nlocal = counts[rank];
  int buf[nlocal];
```

```
   // Tempo paralelo: só a parte de distribuição + computação + redução
   MPI_Barrier(MPI_COMM_WORLD);
   double tpar0 = MPI_Wtime();

   MPI_Scatterv(vet, counts, displs, MPI_INT, buf, nlocal, MPI_INT, 0, MPI_COMM_WORLD);

   // Parciais locais
   long long sum_local = 0;
   long long prod_local = 1;
   for (int i=0;i<nlocal;i++){
      sum_local += buf[i];
      prod_local *= buf[i];
   }

   long long sum_global=0, prod_global=0;
   MPI_Reduce(&sum_local, &sum_global, 1, MPI_LONG_LONG, MPI_SUM, 0,
MPI_COMM_WORLD);
   MPI_Reduce(&prod_local,&prod_global,1, MPI_LONG_LONG, MPI_PROD,0,
MPI_COMM_WORLD);

   MPI_Barrier(MPI_COMM_WORLD);
   double tpar1 = MPI_Wtime();

   if (rank==0) {
      int soma = (int)sum_global;
      int subtracao = -soma;  // conforme semântica original
      long long mult = prod_global;

      double Tpar = tpar1 - tpar0;
      double speedup = (Tserial>0)? (Tserial / Tpar) : 0.0;
      double eficiencia = speedup / p;

      printf("=== Versao (b) mestre/escravo + coletivas (p=%d) ===\n", p);
      printf("Soma = %d\n", soma);
      printf("Subtracao = %d\n", subtracao);
      printf("Multiplicacao = %lld\n", mult);
      printf("Tempo serial   = %.6f s\n", Tserial);
      printf("Tempo paralelo = %.6f s\n", Tpar);
      printf("Speedup        = %.4f\n", speedup);
      printf("Eficiencia     = %.4f\n", eficiencia);
   }

   MPI_Finalize();
   return 0;
}
```

```
// mpi_version_c.c  (Pipeline com 4 processos: 0 feeder/coletor; 1 soma; 2 subtrai; 3 multiplica)
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
```

```
   MPI_Init(&argc,&argv);
   int p, rank;
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if (p != 4) {
      if (rank==0) printf("Esta versao pipeline requer exatamente 4 processos.\n");
      MPI_Finalize();
      return 0;
   }

   const int N = 10;
   int vet[N];
   if (rank==0) for (int i=0;i<N;i++) vet[i]=i+1;

   // Tempo serial (rank 0) para speedup
   double Tserial = 0.0;
   if (rank==0){
      double t0=MPI_Wtime();
      long long m=1;
      int s=0;
      for (int i=0;i<N;i++){ s+=vet[i]; m*=vet[i]; }
      (void)m; // só para não avisar unused; resultado oficial virá do paralelo
      Tserial = MPI_Wtime() - t0;
   }

   const int TAG_DATA=20, TAG_DONE=21, TAG_RES=22;
   // *** BARRIER COLETIVO ANTES DE COMEÇAR O PIPELINE ***
   MPI_Barrier(MPI_COMM_WORLD);

   double tpar0=0.0, tpar1=0.0;

   if (rank==0) {
      tpar0 = MPI_Wtime();

      // Feeder: envia elementos um-a-um para rank1
      for (int i=0;i<N;i++){
         MPI_Send(&vet[i],1,MPI_INT,1,TAG_DATA,MPI_COMM_WORLD);
      }
      // Envia sentinela
      int sent = -1;
      MPI_Send(&sent,1,MPI_INT,1,TAG_DONE,MPI_COMM_WORLD);

      // Recebe resultados finais do rank 3
      int results[3];

MPI_Recv(results,3,MPI_INT,3,TAG_RES,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      tpar1 = MPI_Wtime();

      int soma = results[0];
      int subtracao = results[1];
      int mult = results[2];
```

```
    double Tpar = tpar1 - tpar0;
    double speedup = (Tserial>0)? (Tserial / Tpar) : 0.0;
    double eficiencia = speedup / p;

    printf("=== Versao (c) Pipeline (4 processos) ===\n");
    printf("Soma = %d\n", soma);
    printf("Subtracao = %d\n", subtracao);
    printf("Multiplicacao = %d\n", mult);
    printf("Tempo serial   = %.6f s\n", Tserial);
    printf("Tempo paralelo = %.6f s\n", Tpar);
    printf("Speedup        = %.4f\n", speedup);
    printf("Eficiencia     = %.4f\n", eficiencia);

} else if (rank==1) {
    // Estagio 1: soma; passa adiante para rank2
    int soma=0;
    while (1) {
        MPI_Status st;
        int val;
        MPI_Recv(&val,1,MPI_INT,0,MPI_ANY_TAG,MPI_COMM_WORLD,&st);
        if (st.MPI_TAG==TAG_DONE) {
            // repassa sentinela para proximo estagio
            MPI_Send(&val,1,MPI_INT,2,TAG_DONE,MPI_COMM_WORLD);
            break;
        }
        soma += val;
        MPI_Send(&val,1,MPI_INT,2,TAG_DATA,MPI_COMM_WORLD);
    }
    // envia soma parcial ao rank 3
    MPI_Send(&soma,1,MPI_INT,3,100,MPI_COMM_WORLD);

} else if (rank==2) {
    // Estagio 2: subtrai; passa adiante para rank3
    int sub=0;
    while (1) {
        MPI_Status st;
        int val;
        MPI_Recv(&val,1,MPI_INT,1,MPI_ANY_TAG,MPI_COMM_WORLD,&st);
        if (st.MPI_TAG==TAG_DONE) {
            MPI_Send(&val,1,MPI_INT,3,TAG_DONE,MPI_COMM_WORLD);
            break;
        }
        sub -= val;
        MPI_Send(&val,1,MPI_INT,3,TAG_DATA,MPI_COMM_WORLD);
    }
    // envia sub parcial ao rank 3
    MPI_Send(&sub,1,MPI_INT,3,101,MPI_COMM_WORLD);

} else if (rank==3) {
    // Estagio 3: multiplica; recebe elementos de rank2
    int mult=1;
```

```
        while (1) {
            MPI_Status st;
            int val;
            MPI_Recv(&val,1,MPI_INT,2,MPI_ANY_TAG,MPI_COMM_WORLD,&st);
            if (st.MPI_TAG==TAG_DONE) break;
            mult *= val;
        }
        // Recebe soma e sub dos estágios 1 e 2:
        int soma, sub;
        MPI_Recv(&soma,1,MPI_INT,1,100,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        MPI_Recv(&sub, 1,MPI_INT,2,101,MPI_COMM_WORLD,MPI_STATUS_IGNORE);

        int results[3] = {soma, sub, mult};
        MPI_Send(results,3,MPI_INT,0,TAG_RES,MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

```
mpicc -O2 -o mpi_version_a mpi_version_a.c
mpirun -np 4 ./mpi_version_a
```

```
mpicc -O2 -o mpi_version_b mpi_version_b.c
mpirun -np 8 ./mpi_version_b   # use o p que quiser (>=1)
```

```
mpicc -O2 -o mpi_version_c mpi_version_c.c
mpirun -np 4 ./mpi_version_c
```

```
mpirun -n 4 -oversubscribe codigo_a
=== Versao (a) 4 processos ===
Soma = 55
Subtracao = -55
Multiplicacao = 3628800
Tempo serial   = 0.000000 s
Tempo paralelo = 0.000169 s
Speedup        = 0.0014
Eficiencia     = 0.0003
[ec2-user@ip-172-31-37-22 ~]$ mpirun -n 8 -oversubscribe codigo_b
=== Versao (b) mestre/escravo + coletivas (p=8) ===
Soma = 55
Subtracao = -55
Multiplicacao = 3628800
Tempo serial   = 0.000000 s
Tempo paralelo = 0.000534 s
Speedup        = 0.0004
Eficiencia     = 0.0001
[ec2-user@ip-172-31-37-22 ~]$ mpirun -n 4 -oversubscribe codigo_c
^C[ec2-user@ip-172-31-37-22 ~]$
[ec2-user@ip-172-31-37-22 ~]$ rm codigo_c.c
[ec2-user@ip-172-31-37-22 ~]$ vi codigo_c.c
[ec2-user@ip-172-31-37-22 ~]$ mpicc codigo_c.c -o codigo_c
[ec2-user@ip-172-31-37-22 ~]$ mpirun -n 4 -oversubscribe codigo_c
=== Versao (c) Pipeline (4 processos) ===
```

```
Soma = 55
Subtracao = -55
Multiplicacao = 3628800
Tempo serial   = 0.000000 s
Tempo paralelo = 0.000383 s
Speedup        = 0.0007
Eficiencia     = 0.0002
```