Gerando série a partir de uma lista de números

Dada uma lista de números de entrada, construa um programa MPI que gere uma série de saída cujos valores referem-se aos elementos de entrada multiplicados pela sua posição N na lista, considerando N > 0.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mpi.h>

#define       MASTER        0
#define       TAG           0
#define       SAIFORA       -1

int main() {
      int rank, nprocs;
      int tamanho;

      MPI_Init(NULL, NULL);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

      int *v, indice;

      if (rank == MASTER) {
            scanf("%d",&tamanho);
            v = (int *) malloc(tamanho*sizeof(int));
            for (int i=0; i<tamanho; i++)
                  scanf("%d", &v[i]);
            for (int i=1; i<nprocs; i++) {
                  MPI_Send(&tamanho, 1, MPI_INT, i, TAG, MPI_COMM_WORLD);
                  MPI_Send(v, tamanho, MPI_INT, i, TAG, MPI_COMM_WORLD);
            } /* fim-for */
      } else {
            MPI_Recv(&tamanho, 1, MPI_INT, MASTER, TAG, MPI_COMM_WORLI
```

```
        v = (int *) malloc(tamanho*sizeof(int));
        MPI_Recv(v, tamanho, MPI_INT, MASTER, TAG, MPI_COMM_WORLD
    }
    if (rank == MASTER) {
        indice=0;
        int prox;
        prox = rand()%nprocs; // Ajustar o seek p garantir variação
        if (prox == 0) prox = 1;
        if (prox >=nprocs) prox = nprocs-1;
        while (1) {
            MPI_Send(&indice, 1, MPI_INT, prox, TAG, MPI_COMM_WORLD);
            MPI_Recv(&indice, 1, MPI_INT, prox, TAG, MPI_COMM_WORLD,
            if (indice >= tamanho) {
                indice = SAIFORA;
                for (int i=1; i<nprocs; i++)
                    MPI_Send(&indice, 1, MPI_INT, i, TAG, MPI_COMM_WC
                break;
            } /* fim-if */
            prox = rand()%nprocs;
            if (prox == 0) prox = 1;
            if (prox >=nprocs) prox = nprocs-1;
        } /* fim-while */
    } else {
        int req=0;
        while (1) {
            MPI_Recv(&indice, 1, MPI_INT, MASTER, TAG, MPI_COMM_WOR
            if (indice == SAIFORA)
                break;
            else {
                req++;
                int intervalo=req*rank;
                for (int i=indice; i<(indice+intervalo); i++) {
                    if (i<tamanho) printf("%d ", (i+1)*v[i]); // v[i]*posição_
                    sleep(0.5);
                } /* fim-for */
                indice+=intervalo;
                MPI_Send(&indice, 1, MPI_INT, MASTER, TAG, MPI_COMM_
            } /* fim-else */
```

```
            } /* fim-while */
        } /* fim-slave */
        if (rank == MASTER) printf("\n");
        MPI_Finalize();
        return 0;
} /* fim-main */
```

Carrossel de *threads* OpenMP

Dado um valor n de entrada, construa um programa OpenMP que produza como saída a soma dos naturais no intervalo de [0 ... n-1] multiplicado pelo thid (identificador da *thread*) responsável por cada número natural da série. Considerar ainda que a primeira *thread* a iniciar o processo de soma deve ser escolhida pela *thread* MASTER (via função randômica).

```
#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define        MASTER  0

int main(int argc, char *argv[]) {

        int thid, nthreads;

        int x=0,token = -1;

        int limite = atoi(argv[1]);

        int soma=0;
```

```
#pragma omp parallel private(thid) shared(nthreads, x, token)
{
 thid = omp_get_thread_num();
 #pragma omp single
 nthreads = omp_get_num_threads();
 if (thid == MASTER)
      token = rand()% nthreads; // Falta ajustar a seed dessa função p não
                        // garantir que x fique entre 1 e (nthreads - 1)
 #pragma omp barrier //Necessário p garantir que x foi atualizado pela M
 if (thid != MASTER)
      while (x<limite){
          if (thid == token) {
            if (x<limite) {
              printf("Thread %d/%d → x=%d\n", thid, nthreads, x);
              soma+=(x*thid);
            } /*fim-if*/
            token = token+1;
            if (token == nthreads)
                  token = 1;
```

```
                x++;
            } /* fim-if */
        } /* fim-while */
    } /* fim-área paralela */
    printf("A soma da série = %d\n", soma);
    return 0;
}
```