Coursework 1 for M522 Numerical Analysis (Parallel Computing)

Pavel Buklemishev

February 21, 2025

1 Use OpenMP to speed up the code execution and demonstrate the difference from a serial execution

2 Considering one approach that uses a temporal array to store the results of multiplications before summation, can one use OpenMP reduction to reduce memory allocation?

Code:

```
#include "utils.h"
#include <omp.h>
#include <stdio.h>
#include <cmath>
using namespace std;
double compute_inner_nonparallel(long N, double * x, double *y){
  double z = 0;

  for(int i =0; i<N; i++){
      z += sqrt(x[i]*y[i]) ;
  }

  return z;
}
double compute_inner_reduction(long N, double* x, double* y) {

  double z;

  #pragma omp parallel for reduction(+:z)
  for (long i = 0; i < N; i++){
          z +=sqrt(x[i] * y[i]);}
  return z;
}
double compute_inner_parallel(long N, double * x, double *y){
```

```
  int num_threads = omp_get_max_threads();
  double* sum_per_thread =  (double*) malloc(num_threads* sizeof(double));

  #pragma omp parallel
  {
  int i_thread = omp_get_thread_num();
  double i_sum = 0;
  #pragma omp for
  for(long i =0; i<N; i++){
          i_sum += sqrt(x[i]*y[i]);
  }

  sum_per_thread[i_thread] = i_sum;
  }
  double z =0;
  for(int i = 0; i< num_threads; i++)z += sum_per_thread[i];

  free(sum_per_thread);
  return z;
}

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

  long N = 100000000;

    double* x = (double*) malloc(N * sizeof(double));
        double* y = (double*) malloc(N * sizeof(double));
for (long i = 0; i < N; i++) {
        x[i] = i+1;
        y[i] = 2.0 / (i+1);
}
  double sum;
  Timer t;
  t.tic();

  sum = compute_inner_nonparallel(N, x,y);


  printf("res nonparallel - malloc: %f time elapsed = %f\n",sum, t.toc());

  t.tic();

  sum = compute_inner_parallel(N, x,y);

  printf("res parallel - malloc: %f time elapsed = %f\n",sum, t.toc());
  t.tic();
  sum = compute_inner_reduction(N, x, y);

```

```
  printf("res reduction: %f time elapsed = %f\n",sum, t.toc());

  free(x);
  free(y);
  return 0;
}
```

Result:

```
    (base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# mpiicpx -qopenmp inner_product.cpp
    (base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# ./a.out
    res nonparallel - malloc: 141421356.137423 time elapsed = 0.084770
    res parallel - malloc: 141421356.238374 time elapsed = 0.051157
    res reduction: 141421356.238374 time elapsed = 0.051621
```

Needed to add the sqrt for the demonstrational purposes.

3 Using optimization flags and/or different compilers, demonstrate the differences in the code execution times

g++

```
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# g++ -fopenmp inner_product.cpp
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# ./a.out
res nonparallel - malloc: 141421356.465761 time elapsed = 0.219311
res parallel - malloc: 141421356.238399 time elapsed = 0.048981
res reduction: 141421356.238399 time elapsed = 0.050593
```

g++ with o3 flag

```
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# g++ -fopenmp inner_product.cpp -O3
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# ./a.out
res nonparallel - malloc: 141421356.465761 time elapsed = 0.163962
res parallel - malloc: 141421356.238399 time elapsed = 0.045187
res reduction: 141421356.238399 time elapsed = 0.050634
```

icpx

```
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# icpx -qopenmp inner_product.cpp
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# ./a.out
res nonparallel - malloc: 141421356.137423 time elapsed = 0.083285
res parallel - malloc: 141421356.238374 time elapsed = 0.053423
res reduction: 141421356.238374 time elapsed = 0.049398
```

icpx -o3

```
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# icpx -qopenmp inner_product.cpp -o3
(base) root@pavel-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/pavel/parallel# ./a.out
res nonparallel - malloc: 141421356.137423 time elapsed = 0.083232
res parallel - malloc: 141421356.238374 time elapsed = 0.045548
res reduction: 141421356.238374 time elapsed = 0.043733
```

4 (Optional) Use a hybrid approach: MPI + OpenMP to further parallelise the code

Since my PC does not support MPI, I am attaching a draft of the code that implements the idea of MPI + OpenMP coupled parallelization as I can't debug it properly.

The idea:

1. Divide the vector into $N-1$ equal parts and assign the remainder to the Nth processor.

2. Apply the `compute_inner_reduction` function to each vector part to compute the partial inner sum.

3. Reduce the sums across processors by summing them to obtain the inner product of the entire vector.

```cpp
#include "mpi.h"
#include <stdio.h>
using namespace std;
double compute_inner_reduction(long N, double* x, double* y) {

  double z;

  #pragma omp parallel for reduction(+:z)
  for (long i = 0; i < N; i++){
```

```
            z +=(x[i] * y[i]);}
  return z;
}

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

long N = 100000000;
len_per_proc = N/size;
len_rest_proc= N%size;
double dotProduct;
if(rank==0){
double* x = (double*) malloc(N * sizeof(double));
double* y = (double*) malloc(N * sizeof(double));
for (long i = 0; i < N; i++) {
        x[i] = i+1;
        y[i] = 2.0 / (i+1);}
}
if(rank == size-1){
len_per_proc =len_rest_proc;

}

double sum=0;
double *x_i = (double*) malloc(len_per_proc * sizeof(double));
double *y_i = (double*) malloc(len_per_proc * sizeof(double));


MPI_Scatter(x, len_per_proc, MPI_DOUBLE, x_i, len_per_proc,
   MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Scatter(y, len_per_proc, MPI_DOUBLE, y_i, len_per_proc,
   MPI_DOUBLE, 0, MPI_COMM_WORLD);

//CODE THAT REALIZES VECTOR RESIDUE ALLOCATION TO THE LAST
   PROCESSOR.

sum  = compute_inner_reduction(N, x_i, y_i);
MPI_Reduce(&sum, &dotProduct, 1, MPI_DOUBLE, MPI_SUM, 0,
   MPI_COMM_WORLD);


cout<<dotProduct<<endl;

MPI_Finalize();
return 0;
}
```