HOCHSCHULE
HAMM-LIPPSTADT

Parallel Computing

Präsentiert von Jean Sokolov

Inhaltsverzeichnis

- Problemstellung
- Implementation in OpenMP
- Implementation in MPI

PROBLEMSTELLUNG

Matrix-Multiplikation

- Parallelisieren der Kalkulation
- In C++ mit OpenMP und MPI

IMPLEMENTATION IN C++

Single-thread

m2m_serial.cpp

```cpp
/*
 ============================================================================
 Name        : m2m_serial.cpp
 Author      :
 Version     :
 Copyright   : Your copyright notice
 Description : Matrix matrix multiplication
 ============================================================================
 */
#include <stdio.h>
#include <stdlib.h>

double A[1000][1000];
double B[1000][1000];
double C[1000][1000];

void init_m() {
// initialise
    for (int i = 0; i < 1000; ++i) {
        for (int y = 0; y < 1000; ++y) {
            A[i][y] = rand();
            B[i][y] = rand();
        }
    }
}

void m2m() {
    for (int i = 0; i < 1000; i++) {
        for (int j = 0; j < 1000; j++) {
            for (int k = 0; k < 1000; k++) {
                C[i][j] += A[i][k] * B[k][j];

            }
            printf("%.1f \n", C[i][j]);
        }
    }
}

int main(int argc, char *argv[]) {
    init_m();
    m2m();

    return 0;
}
```

IMPLEMENTATION IN OPENMP

Multi-thread

m2m_serial.cpp | main.cpp

```cpp
/*
 ============================================================================
 Name        : main.cpp
 Author      :
 Version     :
 Copyright   : Your copyright notice
 Description : Matrix matrix multiplication in OpenMP
 ============================================================================
 */
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

double A[1000][1000];
double B[1000][1000];
double C[1000][1000];

void init_m() {
// initialise matrices
    for (int i = 0; i < 1000; ++i) {
        for (int y = 0; y < 1000; ++y) {
            A[i][y] = rand();
            B[i][y] = rand();
        }
    }
}

int main(int argc, char *argv[]) {
    int i, j, k;
    init_m();

#pragma omp parallel for private(i, j, k) shared(A,B,C)
    for (i = 0; i < 1000; i++) {
        for (j = 0; j < 1000; j++) {
            for (k = 0; k < 1000; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
            printf("%.1f \n", C[i][j]);
        }
    }
    return 0;
}
```

Laufzeit und Validierung

```
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ time ./m2m_openmp.o > outOpenMP

real    0m1,424s
user    0m7,452s
sys     0m1,700s
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ time ./m2m_serial.o > outSerial

real    0m3,655s
user    0m3,600s
sys     0m0,056s
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ ls -l
insgesamt 50820
-rwxr-xr-x 1 isd isd    13336 Jun 11 15:50 m2m_openmp.o
-rw-r--r-- 1 isd isd      829 Jun 11 15:50 m2m_serial.cpp
-rwxr-xr-x 1 isd isd     8840 Jun 11 15:50 m2m_serial.o
-rw-r--r-- 1 isd isd      892 Jun 11 15:50 main.cpp
-rw-r--r-- 1 isd isd 26000000 Jul 12 22:19 outOpenMP
-rw-r--r-- 1 isd isd 26000000 Jul 12 22:19 outSerial
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ cat outOpenMP | grep 1184512763025494114304.0
1184512763025494114304.0
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ cat outSerial | grep 1184512763025494114304.0
1184512763025494114304.0
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$ cmp outSerial outOpenMP
outSerial outOpenMP differieren: Byte 3, Zeile 1
isd@debian-eclipse:~/eclipse_luna_workspace/OpenMP/src$
```

IMPLEMENTATION IN MPI

Statische Prozesszahl

```
10 #include "mpi.h"
11 #include <iostream>
12 using namespace std;
13
14 #define num 1000
15 #define USE_MPI_IN_PLACE 0
16
17 double A[num][num];
18 double B[num][num];
19 double C[num][num];
20
21 void init_m() {
22 // initialise
23     for (int i = 0; i < num; ++i) {
24         for (int y = 0; y < num; ++y) {
25             A[i][y] = rand();
26             B[i][y] = rand();
27         }
28     }
29 }
30
31
32 int main(int argc, char *argv[]) {
33     int n, rank, size, i, j, k;
34     init_m();
35
36     MPI::Init(argc, argv);
37     size = MPI::COMM_WORLD.Get_size();
38     rank = MPI::COMM_WORLD.Get_rank();
39
40     if (num%size!=0) {
41         if (rank==0) printf("Matrix size not divisible by number of processors\n");
42         MPI_Finalize();
43         exit(-1);
44     }
45
46     n = num/size;
47     MPI::COMM_WORLD.Bcast(B, num*num, MPI::DOUBLE, 0);
48     if (rank == 0)
49         MPI::COMM_WORLD.Scatter(A, num*n, MPI::DOUBLE, MPI_IN_PLACE, num*n, MPI::DOUBLE, 0);
50     else
51         MPI::COMM_WORLD.Scatter(A, num*n, MPI::DOUBLE, A[rank*n], num*n, MPI::DOUBLE,  0);
52
53     for (i = n*rank; i < n*(rank+1); i++) {
54         for (j = 0; j < num; j++) {
55             for (k = 0; k < num; k++) {
56                 C[i][j] += A[i][k] * B[k][j];
57             }
58             printf("%.1f \n", C[i][j]);
59         }
60     }
61     if (rank == 0)
62         MPI::COMM_WORLD.Gather(MPI_IN_PLACE, num*n, MPI::DOUBLE, C[n*rank], num*n, MPI::DOUBLE, 0);
63     else
64         MPI::COMM_WORLD.Gather(C[n*rank], num*n, MPI::DOUBLE, C, num*n, MPI::DOUBLE, 0);
65
66     MPI::Finalize();
67     return 0;
68 }
```

Dynamische Prozesszahl

MPI_test.cpp | B_row_by_row.cpp | Dynamic_proc_count.cpp

```cpp
#include "mpi.h"
#include <iostream>
using namespace std;

#define num 1000
#define USE_MPI_IN_PLACE 0

double A[num][num];
double B[num][num];
double C[num][num];

void init_m() {
// initialise
    for (int i = 0; i < num; ++i) {
        for (int y = 0; y < num; ++y) {
            A[i][y] = rand();
            B[i][y] = rand();
        }
    }
}


int main(int argc, char *argv[]) {
    int n, rank, size, i, j, k, proc;
    init_m();
    MPI::Init(argc, argv);
    size = MPI::COMM_WORLD.Get_size();
    rank = MPI::COMM_WORLD.Get_rank();

    proc = size;
    while (num%proc!=0) {
        //if (rank==0) printf("Matrix size not divisible by number of processors\n");
        proc--;
    }
    int old_rank = rank;
        MPI_Comm new_comm;
        MPI_Comm_split(MPI_COMM_WORLD, old_rank<proc, old_rank, &new_comm);
        int new_rank, new_size;
        MPI_Comm_rank(new_comm, &new_rank);
        MPI_Comm_size(new_comm, &new_size);
        MPI_Barrier(new_comm);
        //if (rank==0&&proc!=size)
        //printf("Matrix cannot be divided by specified process count. Continuing with %i processes instead.\n", new_size);

        //printf("#%i here\n", old_rank);
        n = num/proc;
        MPI_Bcast(B, num*num, MPI::DOUBLE, 0, new_comm);
        if (old_rank<proc){
            if (new_rank == 0)
                MPI_Scatter(A, num*n, MPI::DOUBLE, MPI_IN_PLACE, num*n, MPI::DOUBLE, 0, new_comm);
            else{
                MPI_Scatter(A, num*n, MPI::DOUBLE, A[new_rank*n], num*n, MPI::DOUBLE,  0, new_comm);
            }
            for (i = n*new_rank; i < n*(new_rank+1); i++) {
                for (j = 0; j < num; j++) {
                    for (k = 0; k < num; k++) {
                        C[i][j] += A[i][k] * B[k][j];
                    }
                    //printf("%.1f \n", C[i][j]);
                }
            }
        }
        if (new_rank<proc){
            if (new_rank == 0)
                MPI_Gather(MPI_IN_PLACE, num*n, MPI::DOUBLE, C[n*new_rank], num*n, MPI::DOUBLE, 0, new_comm);
            else
                MPI_Gather(C[n*new_rank], num*n, MPI::DOUBLE, C, num*n, MPI::DOUBLE, 0, new_comm);
        }
        //printf("#%i done \n", rank);
    MPI::Finalize();
    return 0;
}
```

Laufzeit und Validierung

```
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ time sudo mpirun -np 8 ./m2m_MPI > out3

real    0m1,053s
user    0m6,612s
sys     0m1,140s
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ time sudo mpirun -np 7 ./m2m_MPI > out3

real    0m1,125s
user    0m6,404s
sys     0m2,240s
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ time sudo mpirun -np 6 ./m2m_MPI > out3

real    0m1,178s
user    0m6,728s
sys     0m2,284s
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ time sudo mpirun -np 3 ./m2m_MPI > out3

real    0m2,486s
user    0m7,284s
sys     0m4,672s
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ ls -l out3
-rw-r--r-- 1 isd isd 26000000 Jul 12 22:23 out3
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$ cat out3 | grep 1184512763025494114304.0
1184512763025494114304.0
isd@debian-eclipse:~/eclipse_ptp_workspace/m2m_MPI/Debug$
```

Speichereffiziente Variation

```
7  #include "mpi.h"
8  #include <iostream>
9  using namespace std;
10
11 #define num 1000
12 #define USE_MPI_IN_PLACE 0
13
14 double A[num][num];
15 double B[num][num];
16 double C[num][num];
17 double tmpB[num][num];
18
19 MPI_Status status;
20
21 void init_m() {
22 // initialise
23     for (int i = 0; i < num; ++i) {
24         for (int y = 0; y < num; ++y) {
25             A[i][y] = rand();
26             B[i][y] = rand();
27         }
28     }
29 }
30
31 void rotate_b_matrix(){
32     for (int i = 0; i < num; ++i) {
33         for (int y = 0; y < num; ++y) {
34             tmpB[num-1-y][i]=B[i][y];
35         }
36     }
37     for (int i = 0; i < num; ++i) {
38         for (int y = 0; y < num; ++y) {
39             B[i][y]=tmpB[i][y];
40         }
41     }
42 }
```

Speichereffiziente Variation

```
43
44  int main(int argc, char *argv[]) {
45      int n, rank, size, i, j, k, proc;
46      init_m();
47      rotate_b_matrix();
48
49      MPI::Init(argc, argv);
50      MPI_Barrier(MPI_COMM_WORLD);
51      size = MPI::COMM_WORLD.Get_size();
52      rank = MPI::COMM_WORLD.Get_rank();
53
54      proc = size-1;
55      while (num%proc!=0) {
56          //if (rank==0) printf("Matrix size not divisible by number of processors\n");
57          proc--;
58      }
59      int old_rank = rank;
60          MPI_Comm new_comm;
61          MPI_Comm_split(MPI_COMM_WORLD, old_rank<=proc, old_rank, &new_comm);
62          int new_rank, new_size;
63          MPI_Comm_rank(new_comm, &new_rank);
64          MPI_Comm_size(new_comm, &new_size);
65          MPI_Barrier(new_comm);
66
67          n = num/proc;
68          //printf("packet_size=%i here\n using %i slaves\n", n, proc);
69          if (new_rank==0){
70              for (int i = 1; i<=proc; i++){
71                  for (int j = 0; j<num; j++){
72                      for (int k = n*(i-1); k < n*i; k++) {
73                          MPI_Send(&B[num-1-j], num, MPI_DOUBLE, i, 0, new_comm);
74                      }
75                      MPI_Send(&A[j], num, MPI_DOUBLE, i, 0, new_comm);
76                  }
77              }
78          }
79          if (new_rank<=proc && new_rank != 0){
80              for (int i = n*(new_rank-1); i < n*new_rank; i++) {
81                  MPI_Recv(&A[i], num, MPI_DOUBLE, 0, 0, new_comm, &status);
82                  for (int j = 0; j < num; j++) {
83                      MPI_Recv(&B[num-1-j], num, MPI_DOUBLE, 0, 0, new_comm, &status);
84                      for (int k = 0; k < num; k++) {
85                          C[i][j] += A[i][k] * B[num-1-j][k];
86                      }
87                      printf("%.1f \n", C[i][j]);
88                  }
89              }
90          }
91          MPI_Barrier(new_comm);
92      MPI::Finalize();
93      return 0;
94  }
95
```

Quellcode

- https://github.com/JeanSokolov/ParallelComputing

Vielen Dank für Ihre Aufmerksamkeit