

UNIVERSITÄT
BAYREUTH

LEHRSTUHL FÜR
ANGEWANDTE INFORMATIK II

Parallele und Verteilte Systeme

Prof. Dr. Thomas Rauber

Gebäude: AI
Telefon: 0921/55-7700
Sekretariat: 0921/55-7701
Telefax: 0921/55-7702

Winter Semester 22/23

Written Exam on „Parallel and Distributed Systems I“

on Monday, the 8th of February in 2023, 8:00–9:30 o'clock, H 17 NWII and H 33 AI

Name: _____

Matriculation number: _____ ITS identifier: _____

Course of studies: _____

Task	1	2	3	4	5	6 (B)	Total
Achievable Points	10	25	20	20	15	15	90
Achieved Points							
Exercise (%):						Mark:	

*Please write your **given name**, your **family name** and your **matriculation number** on every sheet of paper! Pay attention that your solutions are **readable** and that it is **comprehensible** how you come to your **solutions**! Please hand in all the sheets of paper containing your **notes**, **auxiliary calculations** and **discarded solutions** at the end of the exam!*

***Allowed tools:** Copies of the lecture slides, calculator, notes from the lecture and exercise (including solutions), book covering the lecture.*

***Electronic tools** of any kind (smart phone, smart watch, ...) are **not allowed**!*

Task 1 (Deadlock in a Ring)

(5 + 5 = 10 Points)

Given is the following excerpt of a program in C with MPI, which implements a communication pattern in a ring:

```

1 int send_to_right(int* send, int send_cnt, int* recv, int N)
2 {
3     int rank, p, succ, pred, recv_cnt;
4     MPI_Status status;
5     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6     MPI_Comm_size(MPI_COMM_WORLD, &p);
7
8     pred = (rank - 1 + p) % p;
9     succ = (rank + 1) % p;
10
11     MPI_Recv(recv, N, MPI_INT, pred, 0, MPI_COMM_WORLD, &status);
12     MPI_Send(send, send_cnt, MPI_INT, succ, 0, MPI_COMM_WORLD);
13
14     MPI_Get_count(&status, MPI_INT, &recv_cnt);
15     return recv_cnt;
16 }

```

- Under the precondition of the function being executed by 3 MPI processes draft a diagram of the program flow in which a deadlock occurs.
- State two approaches, how the deadlock can be avoided. Specify the changed excerpt of the program for both approaches.

Task 2 (Parallel Reduction on a 2D Grid)

(5 + 20 = 25 Points)

Implement a reduction (single-accumulation operation) by point-to-point communication operations in MPI on a 2D process grid. For this, you are to assume that the processes have already been arranged in a virtual 2D process grid having the edge lengths (`proc_count_x`, `proc_count_y`) via the MPI function `MPI_Cart_create`, and that each process may only communicate in this 2D process grid with its adjacent processes by point-to-point communication operations. For this reduction on the 2D process grid, analogous to the reduction on a tree in the exercise, each process should first receive the data to be reduced from its adjacent processes, then perform a partial reduction on the received data and its own data to be reduced, and then send the result of this partial reduction further to another adjacent process in the direction of the root process. However, as an obvious exception in this procedure, the root process only needs to calculate the final result of the reduction operation from the data received from its adjacent processes and from its own data to be reduced.

For simplicity, assume that the root process is in the middle of the process grid, i.e. at the coordinates (`proc_count_x / 2`, `proc_count_y / 2`), and that `proc_count_x > 3 && proc_count_x % 2 == 0` and `proc_count_y > 3 && proc_count_y % 2 == 0` holds true. Moreover, assume that the addition is used as the reduction operator, and that the reduction is performed on `double` values.

For your implementation, proceed as follows:

- First draft the data flow of the reduction in the 2D process grid, i.e. for each process in the 2D process grid, from which adjacent processes it has to receive the data to be reduced and to which adjacent process it has to send the result of its partial reduction!
- Implement the reduction on a virtual 2D process grid via MPI by a function with the following signature:

```
void reduce_grid(const void *sendbuf, void *recvbuf, int count, MPI_Comm comm).
```

For your implementation, you are to assume that the communicator `comm` passed to the function is already a 2D process grid created by `MPI_Cart_create`. Within your implementation, you can query the edge lengths (`proc_count_x`, `proc_count_y`) of the 2D process grid with the function `MPI_Cart_get`!

Task 3 (Optimization of the Performance)

(10 + 10 = 20 Points)

You are to optimize the performance of the following MPI code fragment under the precondition of this code fragment being run on a cluster with one process per node:

```

1 #define SIZE (64*1024*1024)
2 double *A, *B, *C, *RES, *X, *Y; // all arrays are allocated with
3                                     // malloc(sizeof(double)*SIZE)
4 int my_rank;
5 MPI_Comm_rank(&my_rank);
6 if(my_rank == r0) // r0 == 0
7 {
8     //... Process r0 computes A via several complex steps
9
10    MPI_Barrier(MPI_COMM_WORLD) // Barrier is required and should not be removed
11
12    clock_t t_start = clock();
13
14    compute_b(/*output:*/ B, /*input:*/ A);
15    compute_c(/*output:*/ C, /*input:*/ A);
16    MPI_Send(B, SIZE, MPI_DOUBLE, r1, 11, MPI_COMM_WORLD);
17    MPI_Recv(Y, SIZE, MPI_DOUBLE, r1, 27, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
18    compute_res(/*output:*/ RES, /*input:*/ C, /*input:*/ Y);
19
20    clock_t t_end = clock();
21    double t_elapsed = (double)(end_t - start_t) / CLOCKS_PER_SEC;
22    printf("Time ellapsed from barrier to acquiring RES: %f \n", t_elapsed);
23 }
24 else if (my_rank == r1) // r1 == 1
25 {
26     //... Process r1 computes X via several complex steps
27
28    MPI_Barrier(MPI_COMM_WORLD) // Barrier is required and should not be removed
29
30    MPI_Recv(B, SIZE, MPI_DOUBLE, r0, 11, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
31    compute_y(/*output:*/ Y, /*input:*/ B, /*input:*/ X);
32    MPI_Send(Y, SIZE, MPI_DOUBLE, r0, 27, MPI_COMM_WORLD);
33 }

```

In order to optimize the performance of this code fragment, proceed as follows:

- Draw a diagram illustrating the execution of the code fragment over time starting from the barrier, i.e. of the calculations of the two processes and the communication between these two processes! Assume that no system buffer is available and the following runtimes:
 - Execution of `compute_b` by `r0`: 2 ms
 - Execution of `compute_c` by `r0`: 6 ms
 - Execution of `compute_res` by `r0`: 2 ms
 - Execution of `compute_y` by `r1`: 1 ms
 - Network transfer of an array of size `SIZE` between `r0` and `r1`: 2 ms.

According to your diagram, how long does it take for the result becoming `RES` available on the process `r0`? What problem does your diagram reveal?

- How do you have to improve this code fragment with another type of single transfer operations so that the result `RES` becomes available on the process `r0` as fast as possible? Implement your improvements to the code fragment! With your improvements, how long does it take for the result `RES` becoming available on the process `r0`?

Task 4 (Parallelization of a Time-Step Based Simulation)

(5 + 15 = 20 Points)

In the exercises to the lecture, we have already learned about the simple time-step based simulation of the heat equation. This simulation, in order to calculate the next state of one of its components, only had to consider the component itself and the two adjacent components. However, unlike the simulation of the heat equation, there are also other time-step based simulations, such as an N-body simulation, which, in order to calculate the next state of one of its components, must consider all other components. Since an N-body simulation is too complex for the exam, we consider a fictitious time-step-based simulation with a computational structure similar to an N-body simulation in this task. The sequential source code for this fictitious simulation is already given:

```

1 float compute_contribution    // Cheap to compute
2     (float s_1, float s_2); // but opaque implementation
3
4 int main(int argc, char** args)
5 {
6     printf("Enter simulation size:\n");
7     int sim_size;
8     scanf("%i", &sim_size);
9     float* S_k = malloc(sizeof(float) * sim_size);
10    float* S_kn = malloc(sizeof(float) * sim_size);
11
12    for(int i = 0; i < sim_size; i++)
13        S_k[i] = i;
14
15    for(int k = 0; k < num_time_steps; k++)
16    {
17        for(int i = 0; i < sim_size; i++)
18        {
19            double dSdt_i = 0;
20            for(int j = 0; j < sim_size; j++)
21                dSdt_i += compute_contribution(S_k[i], S_k[j]);
22
23            S_kn[i] = S_k[i] + dSdt_i;
24        }
25        swap(&S_k, &S_kn);
26    }
27
28    float S_max = FLT_min;
29    for(int i = 0; i < sim_size; i++)
30        S_max = fmaxf(S_max, S_k[i]);
31    printf("S_max: %f\n", S_max);
32
33    free(s_k);
34    free(s_kn);
35
36    return 0;
37 }

```

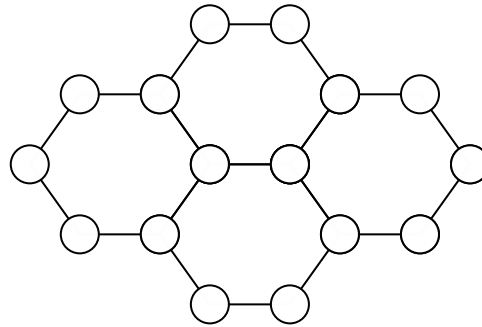
You are now to parallelize this sequential program with MPI:

- First, briefly present an approach for parallelizing this program with MPI. Specifically, address how to distribute the computations in the time loop as fairly as possible among the MPI processes. Also address which MPI-communication operations you need for your parallelizing this program.
- Now write a program implementing your approach. Take special care in your program that the value entered by the user for `sim_size` does not have to be a multiple of the number of MPI processes started.

Task 5 (Comprehension Questions on Networks)

(5 + 10 = 15 Points)

- a) State the properties (node count, edge count, degree, node connectivity, edge connectivity, bisection width and diameter) of the following graph:



- b) Omega networks and butterfly networks having the same dimension, i.e. having the same number of I/O wires, are isomorphic to each other. Because of this isomorphism, it is possible to transform these switching networks into each other by permuting the input wires, permuting the output wires and also permuting the switches within each stage. However, for this isomorphism, the distinction between the upper input and the lower input of the switches as well as the distinction between the upper output and the lower output of the switches must be ignored. Transform an 8×8 omega network into an 8×8 butterfly network by creating a permutation for the inputs (Π_{Inputs}), for the switches of each stage ($\Pi_{\text{1st Stage}}$, $\Pi_{\text{2nd Stage}}$, $\Pi_{\text{3rd Stage}}$) and the outputs (Π_{Outputs})!

Task 6 (Bonus Task about the Embedding of Graphs)

(4 + 2 + 9 = 15 Points)

- a) Embed a ring consisting of 16 nodes into a 2D grid consisting of 4×4 nodes by first drawing the grid, and then inserting the node numbers of the ring into the nodes of the grid!
- b) Given two graphs g_1 and g_2 with $\text{degree}(g_1) > \text{degree}(g_2)$. Under this condition, is it still potentially possible to embed g_1 in g_2 ? Give reasons for your answer!
- c) In general, is it possible to embed a given complete binary tree of arbitrary height into a 2D grid, which you are allowed to choose arbitrarily large for this embedding? If such an embedding is impossible for a 2D grid, then perhaps it is possible for 3D, 4D or 5D grids? Give reasons for your answer!

Hints:

- In a complete binary tree, how many nodes can you reach starting from the root node if you are allowed to go a maximum of n edges?
- In a d dimensional grid and depending on the dimension d of the grid, how many nodes can one reach starting from an inner node of the grid, if one is allowed to walk a maximum of n edges? For the argumentation in this task, an upper bound for this number of reachable nodes in a grid is also sufficient, for which e.g. a bounding cube can be used in a 3D grid.
- How does this affect whether embedding the tree in the grid is possible?