P. Koumoutsakos, M. Troyer
ETH Zentrum, CAB H 69.2
CH-8092 Zürich

Exam

Issued: December 18, 2012, 14:00
Hand in: December 18, 2012, 17:00

Exam directives. In order to pass the exam, the following requirements have to be met:

- Clear your desk (no cell phones, cameras, etc.): on your desk you should have your Legi, your pen, your notes and books. We provide you with the necessary paper, exam sheets and electronic material.

- Read carefully the first three pages of the exam. Write your name, Legi-ID and the computer IP (or the hostname) where requested. Before handing in the exam, **PUT YOUR SIGNATURE ON PAGE 2**.

- The teaching assistants will give you the necessary paper sheets. You are not allowed to use any other paper sheets to write the solutions you hand in.

- Your answers should be handwritten in blue or black (no pencils allowed), clearly readable and in English. Only one answer per question is accepted. Invalid answers should be clearly crossed out. Whenever you write a C++ code, include also the associated comments.

- To answer new questions (e.g. Question 1, not sub-questions!), always use a new page. On the top-right corner of every page write your complete name and Legi-ID. Whenever you are supposed to point out something in the code or draw on the plots provided, you are allowed to mark it directly in the exam sheets. Unless otherwise noted in the question, you should always hand-in your answers on paper!

- If something is disturbing you during the exam, or it is preventing you from peacefully solving the exam, please report it immediately to an assistant. Later notifications will not be accepted.

- You must hand in: the exam cover, the sheets with the exam questions and your solutions. The exam cannot be accepted if the cover sheet or the question sheets are not handed back. The solutions codes have to be saved in the corresponding folder (as detailed later).

- Although we suggest to solve simple questions without the use of the computer, you are allowed to use it to help you answer any question. The "exam mode" of the computers is that they are offline (no internet). You can use any text editor, GNU make and gcc.

Family Name:	
Name:	
Legi-ID:	
Computer Hostname or IP:	

Question	Maximum score	Score	TA 1	TA 2
1	15			
2	20			
3	20			
4	15			
5	40			
6	15			
7	30			
8	10			
9	15			
10	10			
11	20			

A perfect score is 180 points (out of 210 available).

With your signature you confirm that you have read the exam directives; you solved the exam without any unauthorized help and you wrote your answers following the outlined directives.

Signature:

Computer Setup

Before you start with the exam please read the following instructions carefully!

Logon to the machine by choosing the **user "Student"** and typing the **password "ethz"**. You can choose either the GNOME or KDE environment for the exam. On the next screen please enter your **nethz** user name. Afterwards, you will be asked to enter your **first and last name**. You are now logged into the Linux desktop in exam mode.

In your home directory (`~` or `/var/lib/exam/student`) you will find the two directories `resources/` and `results/`. The directory `resources/` contains the files **we** provide to you. The directory `results/` will contain **your** solutions. Only files in the directory `results/` will be fetched from your local machine and considered for the correction of the exam. Please make sure to first copy the skeleton codes we provide to your local `results/` directory **before** you start modifying them:

```
$ cd
$ cp -r resources/EXAM/skeleton_codes results/
$ cd results/skeleton_codes
```

Please do NOT copy the additional material contained in the folder `resources/SVN` (which contains manuals, class notes, exercises) to the results directory.

Inside the directory `skeleton_codes/` which you just copied to your `results/` directory, you find the following 3 project directories:

- `HPCSE12_ParallelSum` (Question 5)
- `HPCSE12_ENG_Tasks` (Question 6, Engineers only)
- `HPCSE12_MonteCarlo` (Question 7)

Each of these directories contains the skeleton codes you will need for the corresponding exercises as well as a makefile that you can use to compile your code by calling

```
$ make
```

Note that for Question 5 you can also specify different targets: 'sse', 'omp', 'threads' and 'mpi', e.g.

```
$ make sse
```

MPI

The MPI installation resides in `/usr/lib64/openmpi/`. You only need MPI in Question 5 b) where we provide a local `mpirun` script for your convenience. In the directory `HPCSE12_ParallelSum/` you can run MPI binaries by calling

```
$ ./mpirun -np [nprocesses] [executable]
```

instead of `mpirun`.

Documentation

Unzip the svn.tar.gz by calling

```
$ tar zxvf svn.tar.gz
```

This folder contains a copy of the svn directories you accessed during the semester. Within it, you will find sample codes (folder `examples/`), exercises and solutions (folder `exercises/`), lecture notes (folder `lecture_notes/`) and manuals (folder `manuals/`).

In the folder `manuals/`, you find the documentation for MPI, OpenMP, CUDA, SSE/AVX Intrinsics and C++.

The Intel Intrinsics Guide can be run from its folder within a terminal with

```
$ ./Intrinsics\ Guide\ for\ Intel\(R\)\ AVX2.sh
```

C++ documentation (cppreference.com and cplusplus.com in compressed format) is provided in the directory `resources/SVN/manuals`.

cppreference.com

unzip website:

```
$ tar zxvf cppreference_html_book_20121202.tar.gz
```

the main page is `output/en/index.html`

cplusplus.com

unzip website:

```
$ tar xvjf www.cplusplus.com.tar.bz2
```

the main page is `www.cplusplus.com/index.html` (note that the search function will NOT work).

Parallelization

Question 1: Loop Parallelization (15 points)

State whether the following code snippets can be parallelized by prefixing the loop with

```
#pragma omp parallel for
```

In case the answer is *no* give a reason. In case the answer is *yes* explain whether the code can also be vectorized.

Assume there is no aliasing between different arrays. For case c) assume that the array `idx` has been filled with a random permutation of numbers between 0 and N.

a)

```
for (int i=0; i<20; i++)
    result[i+20] = result[i] + B[i];
```

b)

```
for (int i=1; i<N-1; i++)
    result[i] = result[i+1] + result[i-1];
```

c)

```
for (int i=0; i<N; i++)
    result[i] = B[idx[i]];
```

d)

```
for (int i=1; i<N; i++) {
    result1[i]=2*A[i];
    result2[i]=result1[i-1]+B[i];
}
```

Question 2: Parallelization Bugs (20 points)

The following code snippets compile, but will show undesired runtime behavior. Find a bug in each snippet and propose a solution. Assume all headers are included correctly:

a)

```
double a=2.;
int nthreads=10;
std::vector<std::thread> threads;
for (int i=0; i<nthreads; ++i)
    threads.push_back(std::thread([&a] {
        a += 3.14/2*i;
    }));
for (std::thread& t: threads)
    t.join();
std::cout << "Result is " << a << std::endl;
```

b)

```
int nthreads=10;
double foo(); // implemented somewhere else, and thread-safe
auto ff = [] (double& x) {
                x *= foo();
            };
std::vector<std::thread> threads;
std::vector<double> vec(nthreads,3.);
for (int i=0; i<nthreads; ++i)
    threads.push_back(std::thread(ff, std::ref(vec[i])));
double b = std::accumulate(vec.begin(), vec.end(), 0.);
std::cout << "Result is " << b << std::endl;
for (std::thread& t: threads)
    t.join();
```

c)

```
std::mutex m;
int count=0;

std::thread t([&] {
    std::lock_guard<std::mutex> lock(m);
    std::cout << "Thread counts " << count++ << std::endl;
});
std::lock_guard<std::mutex> lock(m);
std::cout << "Main counts " << count++ << std::endl;
t.join();
```

d)

```
        int myid, numprocs; /* assume numprocs >= 2 */

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD, &myid);

        int left = myid - 1;
        int right = myid + 1;

        if (myid == 0) left = numprocs-1;
        if (myid == numprocs-1) right = 0;

        MPI_Status status;
        int x = myid;   // send x right, receive from left

        MPI_Ssend(&x, 1, MPI_INT, 0, right, MPI_COMM_WORLD);
        MPI_Recv(&x, 1, MPI_INT, 0, left, MPI_COMM_WORLD, &status);
        std::cout << "Process " << myid << " received " << x << std::
            endl;
```

Question 3: Parallel Scaling (20 points)

Assume we implemented a solver for the 1D diffusion equation

$$\frac{\partial u(x,t)}{\partial t} = \nu \frac{\partial^2 u(x,t)}{\partial x^2} \quad x \in [0,1[\tag{1}$$

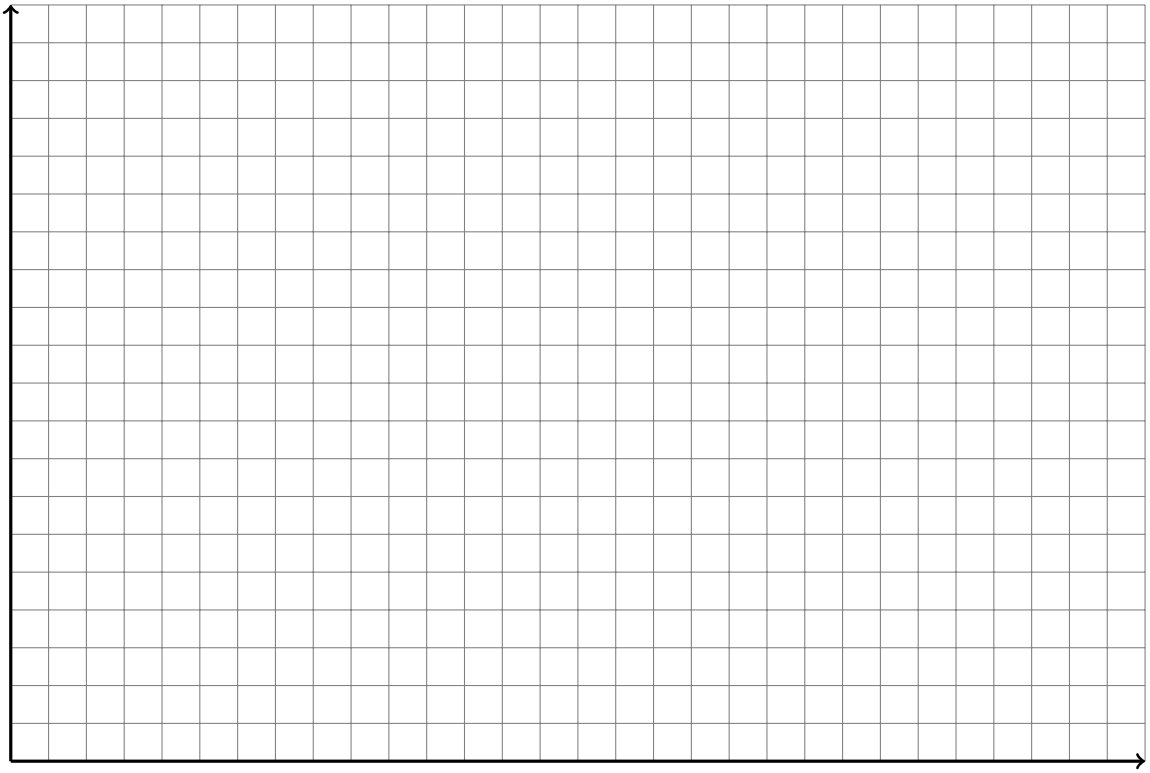
with central finite difference in space and forward Euler in time.

The following table reports timing results for the solver for various grid sizes N on P processor cores with a fixed number of time steps:

	runtime [s]			
P	$N = 1000$	$N = 2000$	$N = 4000$	$N = 8000$
1	12.0	30.0	60.0	110.0
2	6.0	15.0	32.0	60.0
4	4.0	9.0	20.0	32.0
8	3.0	6.0	13.0	24.0

a) Draw a strong scaling plot for this program using the value for $N = 1000$ at $P = 1$ as reference. Don't forget to label the axes.

b) Draw the weak scaling plot for this program, using the value for $N = 1000$ at $P = 1$ as reference. Don't forget to label the axes.



c) Profiling a serial code you find that 98% of the runtime is spent in one loop. What is the best speedup you can expect after parallelizing this loop with OpenMP and running on 48 cores? (For the best speedup we assume there is no parallelization overhead and the loop is perfectly parallelizable. The code's performance is compute bound.)

Question 4: Roofline Model (15 points)

Given the following serial GEMM code snippet:

```
1 for (int k=0; k<N; k++)
2     for (int i=0; i<N; i++)
3         for (int j=0; j<N; j++)
4             C[i+j*N] += A[i+k*N]*B[k+j*N];
```

a) Can you identify potential problems and/or difficulties in the loop structure of the code, in particular regarding parallelization and locality?

b) What is the operational intensity (single precision) for a matrix of size N, assuming there is no caching? Count floating point operations and memory accesses. (Tip: You may write on the code or draw a picture.)

c) What is the operational intensity (single precision) for a matrix of size N, assuming an infinite cache size? Count floating point operations and memory accesses. (Tip: You may write on the code or draw a picture.)

d) Complete Figure 1 by adding vertical lines corresponding to the minimum and maximum operational intensities computed in parts b) and c) for $N = 80$.

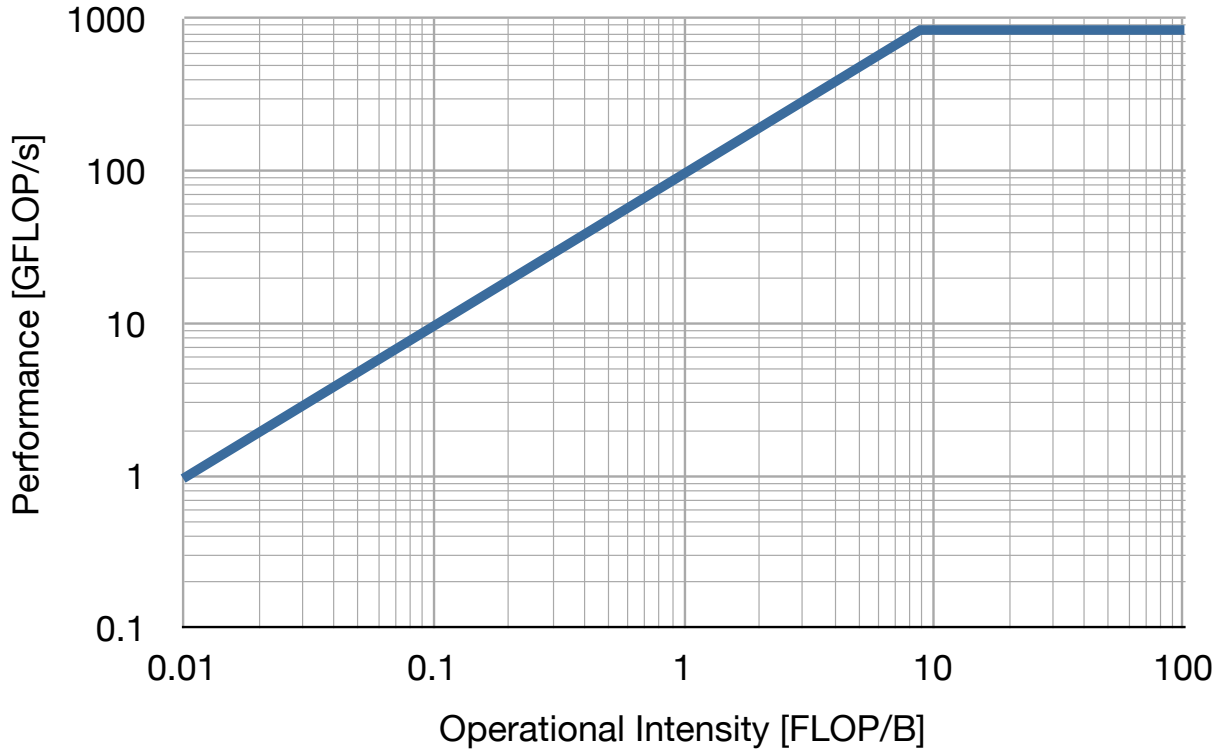


Figure 1: Roofline for Brutus.

e) What is the maximum performance you can reach with the two operational intensities computed for $N = 80$ given that a Brutus node has a bandwidth of 96 GB/s and a peak single precision floating point performance of 844.8 GFLOP/s? Show your calculations.

Programming

Question 5: Parallel sum (40 points)

Parallelize the following vector addition:

```
for (int i=0; i<N; i++)
    z[i] = x[i] + y[i];
```

The directory `skeleton_codes/HPCSE12_ParallelSum/` contains a skeleton code for each of the following questions. The solutions should be put directly into these files.

a) using OpenMP. You can use the following code as a starting point and compile it by calling

`$ make omp`

```
// Skeleton code for HPCSE Exam, 18.12.2012
// Profs. P. Koumoutsakos and M. Troyer
// Question 5a)

#include <vector>
#include <numeric>
#include <iostream>


int main( int argc, char** argv )
{
    // vector size
    const int N = 1600000;

    // initialize vectors
    std::vector<float> x(N,-1.2), y(N,3.4), z(N);


    // DO THE SUM z = x + y
    ...


    // print result checksum
    std::cout << std::accumulate(z.begin(), z.end(), 0.) << std::endl;
}
```

Listing 1: Skeleton code `HPCSE12_ParallelSum/omp.cpp`

b) using C++11 threads. Create 4 threads such that N is a multiple of the number of threads. You can use the following code as a starting point and compile it by calling

`$ make threads`

```
// Skeleton code for HPCSE Exam, 18.12.2012
// Profs. P. Koumoutsakos and M. Troyer
// Question 5b)

#include <vector>
#include <numeric>
#include <iostream>
#include <thread>

```

```
10
11 int main( int argc, char** argv )
12 {
13     // vector size
14     const int N = 1600000;
15
16     // initialize vectors
17     std::vector<float> x(N,-1.2), y(N,3.4), z(N);
18
19
20     // DO THE SUM z = x + y using 4 threads
21     ...
22
23
24     // print result checksum
25     std::cout << std::accumulate(z.begin(), z.end(), 0.) << std::endl;
26 }
```

Listing 2: Skeleton code `HPCSE12_ParallelSum/threads.cpp`

c) using SSE for single precision. You may assume that N is a multiple of 4 and that the vectors `x, y, z` are aligned. You can use the following code as a starting point and compile it by calling

```
$ make sse
```

```
1  // Skeleton code for HPCSE Exam, 18.12.2012
2  // Profs. P. Koumoutsakos and M. Troyer
3  // Question 5c)
4
5  #include <vector>
6  #include <numeric>
7  #include <iostream>
8  #include <x86intrin.h>
9  #include "aligned_allocator.hpp"
10
11
12 int main( int argc, char** argv )
13 {
14     // vector size
15     const int N = 1600000;
16
17     // initialize 16 byte aligned vectors
18     std::vector< float, hpc12::aligned_allocator<float,16> > x(N,-1.2)
           , y(N,3.4), z(N);
19
20
21     // DO THE SUM z = x + y
22     ...
23
24
25     // print result checksum
26     std::cout << std::accumulate(z.begin(), z.end(), 0.) << std::endl;
27 }
```

Listing 3: Skeleton code `HPCSE12_ParallelSum/sse.cpp`

d) using MPI. You may assume that N is a multiple of the number of processes. You can use the following code as a starting point, compile and run it by calling

```
$ make mpi
$ ./mpirun -np 4 ./mpi
```

```cpp
// Skeleton code for HPCSE Exam, 18.12.2012
// Profs. P. Koumoutsakos and M. Troyer
// Question 5d)

#include <vector>
#include <numeric>
#include <cassert>
#include <mpi.h>

int main( int argc, char** argv )
{
    // vector size
    const int N = 1600000;
    int num_processes, rank;

    // SET UP COMMUNICATION.
    // DETERMINE num_processes AND OUR rank
    ...


    // initialize local parts of the vectors and do the sum z = x + y
    assert( N % num_processes == 0 );
    int nlocal = N / num_processes;
    std::vector<float> x(nlocal,-1.2), y(nlocal,3.4), z(nlocal);
    for( int i = 0; i < nlocal; i++ )
        z[i] = x[i] + y[i];

    if( rank == 0 )
    {
        std::vector<float> fullz(N);


        // COLLECT ALL PARTS INTO fullz
        ...


        // print result checksum
        std::cout << std::accumulate( fullz.begin(), fullz.end(), 0. )
             << std::endl;
    }
    else
    {
        // SEND LOCAL PART z TO ROOT PROCESS
        ...
    }

    // CLEAN UP
    ...
}
```

Listing 4: Skeleton code `HPCSE12_ParallelSum/mpi.cpp`

Question 6: Task Based Parallelism (15 points)

The following sequential code is given:

```
1  // Skeleton code for HPCSE Exam, 18.12.2012
2  // Profs. P. Koumoutsakos and M. Troyer
3  // Question 6
4
5  #include <iostream>
6  #include <thread>
7  #include <chrono>
8  #include <unistd.h>
9
10 int alpha()                          {sleep(1); return 10; }
11 int beta()                           {sleep(1); return 20; }
12 int gamma()                          {sleep(1); return 50; }
13 int delta()                          {sleep(1); return 80; }
14 int epsilon(int x, int y, int z)     {sleep(1); return x*y+z; }
15 int zeta(int x, int y, int z)        {sleep(1); return x-y+z; }
16 int ita(int x, int y)                {sleep(1); return x/y; }
17
18 int main()
19 {
20     int A, B, C, D, E, F, G;
21
22     std::chrono::time_point<std::chrono::high_resolution_clock> t0, t1;
23     t0 = std::chrono::high_resolution_clock::now();
24
25     A = alpha();
26     B = beta();
27     C = gamma();
28     D = delta();
29     E = epsilon(A, B, C);
30     F = zeta(B, C, D);
31     G = ita(E, F);
32
33     t1 = std::chrono::high_resolution_clock::now();
34     double elapsed_time = std::chrono::duration<double>(t1-t0).count();
35
36     std::cout << "G = " <<  G << std::endl;
37     std::cout << "Elapsed = " << elapsed_time << " seconds\n";
38
39     return 0;
40 }
```

Listing 5: Skeleton code `HPCSE12_ENG_Tasks/tasks.cpp`

The code can be found in `skeleton_codes/HPCSE12_Tasks/tasks.cpp`. The program prints the value of variable G in approximately 7 seconds, since `sleep(1)` inserts a delay of 1 second to the calling thread. The goal of this question is to minimize the execution time of the code without affecting its correctness. In addition, the following constraints must be met:

1. The `alpha,beta,...,ita` routines cannot be modified.
2. The declaration of the variables `A,B,...,G` (line 20) must remain in `main()`.
3. Usage of global variables is not allowed.

a) Identify the dependencies in the code and draw a small diagram showing them.

b) Parallelize the program using multithreading (either C++11 threads or OpenMP). The solution should be implemented in `skeleton_codes/HPCSE12_Tasks/tasks.cpp`.

Question 7: Monte Carlo Sampling (30 points)

The code in `skeleton_codes/HPCSE12_MonteCarlo` (shown below) uses Monte Carlo to compute the integral of the multidimensional function

$$f(x_1, \ldots, x_4) = \prod_{i=1}^{4} \sin\left(x_i + \frac{\pi}{2} i\right) \tag{2}$$

in the box $x_i \in [-2, 3/2)$.

```
// Skeleton code for HPCSE Exam, 18.12.2012
// Profs. P. Koumoutsakos and M. Troyer
// Question 7

#include <iostream>
#include <algorithm>
#include <array>
#include <cmath>
#include <random>

#define DIM 4

inline double func(std::array<double,DIM> const& x)
{
    double ret = 1.;
    for (std::size_t i=0; i<DIM; ++i)
        ret *= std::sin(x[i] + M_PI/2.*(i+1));
    return ret;
}


int main(int argc, char ** argv) {
    const std::size_t M = 100000;
    const double a=-2., b=1.5;

    const double volume = std::pow((b-a), DIM);

    std::mt19937 eng(11);
    std::uniform_real_distribution<double> dist(a,b);
    auto rng = std::bind(dist, std::ref(eng));

    long double sum=0.;
    for (std::size_t m=0; m<M; ++m) {
        std::array<double,DIM> x;
        std::generate(x.begin(), x.end(), rng);
        long double val = func(x);

        sum += val;
    }

```

```
41      double mean = sum / M;
42      double integral = volume * mean;
43      std::cout << "Result is " << integral << " +/- " << "unknown" << std
            ::endl;
44  }
```

Listing 6: Skeleton code `HPCSE12_MonteCarlo/mc_integrate.cpp`

a) Implement the error estimation, and report the result for $M = 100'000$ samples.
Write the code in `skeleton_codes/HPCSE12_MonteCarlo/mc_integrate.cpp` and report the result on the solution sheet for this question.

b) If you want to reduce the error by a factor of 8, how many samples M do you have to take?

c) Parallelize your code with OpenMP or C++11 threads.
Write the code in `skeleton_codes/HPCSE12_MonteCarlo/mc_integrate.cpp` (the same file as in part *a*).

Applications

Question 8: Scaling of Monte Carlo Methods (10 points)

a) Discuss whether Monte Carlo with direct sampling is perfectly parallel.

b) Discuss whether Markov chain Monte Carlo is perfectly parallel.

Question 9: Sparse Linear Algebra (15 points)

In computations with sparse matrices we can save memory and increase performance by storing the matrix in special formats.

a) Given the following sparse matrix:

$$\begin{pmatrix} 5 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 7 \\ 0 & 1 & 0 & 6 & 3 \end{pmatrix} \tag{3}$$

Write the matrix in CSR format.

b) What is the maximum number of non-zero elements (nnz) in a sparse $m \times n$ matrix for CSR to use less memory than dense storage format? Assume the matrix elements are stored as 64-bit `double` and the indices as 32-bit `int`.

c) Explain one reason why it could be advantageous for small matrices to use dense matrix format for matrix-vector multiplication even if the matrix is sparse.

Question 10: Molecular Dynamics (10 points)

In the N-Body problem we simulate the time evolution of N point particles with pairwise interactions. As in the exercises, we consider the Velocity-Verlet algorithm to solve the problem numerically. Due to the force calculations, the computational effort grows as $\mathcal{O}(N^2)$ with increasing number of particles N.

Sketch (in a few sentences and a figure) how you could reduce the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ in the case of short range interactions.

Question 11: Diffusion (20 points)

The linear diffusion equation in 1D is written as

$$\frac{\partial u(x,t)}{\partial t} = \nu \frac{\partial^2 u(x,t)}{\partial x^2}, \quad -\infty < x < \infty, \quad 0 < t < \infty \tag{4}$$

$$u(x,0) = u_0(x), \quad -\infty < x < \infty \tag{5}$$

where ν is the diffusion coefficient and u_0 is the initial condition. For $\nu > 0$ the solution to this problem after a long time approaches $\lim_{t\to\infty} u(x,t) = 0$.

We also define the Fourier transforms of an arbitrary function $f(x)$ as

$$\mathcal{F}(f)(k) = \hat{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{x=-\infty}^{\infty} f(x) e^{-ikx} \, \mathrm{d}x \tag{6}$$

$$\mathcal{F}^{-1}(\hat{f})(x) = f(x) = \frac{1}{\sqrt{2\pi}} \int_{k=-\infty}^{\infty} \hat{f}(k) e^{ikx} \, \mathrm{d}k. \tag{7}$$

a) Solve equation (4) in Fourier space for $\hat{u}(k,t)$, the Fourier transform of $u(x,t)$.

b) The diffusion equation (4) is solved in real space by approximating the space derivative with a central second order finite difference scheme and integrating the time with an (implicit) backward Euler method:

$$u_j^{(n+1)} = u_j^{(n)} + \frac{\delta t\, \nu}{\delta x^2}\left[u_{j+1}^{(n+1)} + u_{j-1}^{(n+1)} - 2u_j^{(n+1)}\right]. \tag{8}$$

Prove that the scheme in eq. (8) is unconditionally stable.

Good luck!