High Performance Computing for Science and Engineering I

P. Koumoutsakos, M. Troyer
ETH Zentrum, CTL F 11
CH-8092 Zürich

Fall semester 2015

Exam

Issued: December 18, 2015, 09:00
Hand in: December 18, 2015, 12:00

Last Name:	
First Name:	
Student ID:	
Computer Hostname:	

With your signature you confirm that you:

- Have read the exam directives
- You solved the exam without any unauthorized help
- You wrote your answers following the outlined directives

Signature:

Grading table

Question	Maximum score	Score	TA 1	TA 2
Question 1	15			
Question 2	20			
Question 3	25			
Question 4	20			
Question 5	20			
Question 6	25			
Coding 1	35			
Coding 2	25			
Coding 3	25			
Total:	210			

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

Question 1: Monte Carlo Methods (15 points)

At a fixed temperature T the expectation value of a physical observable A can be calculated as a sum over all configurations c:

$$\langle A \rangle = \frac{1}{Z} \sum_c A(\{\vec{x}_i\}_c)\, \omega(\{\vec{x}_i\}_c), \tag{1}$$

with the Boltzmann weight

$$\omega(\{\vec{x}_i\}_c) = \exp\left(-E(\{\vec{x}_i\}_c)/k_B T\right) \tag{2}$$

where $E(\{\vec{x}_i\}_c)$ is the energy of the system in the configuration $\{\vec{x}_i\}_c$. The normalization factor is the partition function defined as $Z = \sum_c \omega(\{\vec{x}_i\}_c)$.

a) Why, in this case, is the Markov Chain Monte Carlo algorithm improving the calculation of $\langle A \rangle$ compared to simple Monte Carlo sampling?

b) In the Metropolis algorithm the probability to move from one configuration to another is $W_{a,b} = A_{a,b} P_{a,b}$, where $A_{a,b}$ is the a-priori proposal probability and $P_{a,b}$ is the a-posteriori acceptance probability. In the lecture we have seen that the acceptance probability is given by

$$P_{a,b} = \min\left[1, R_{a,b}\right] \quad \text{where } R_{a,b} = \frac{\omega(\{\vec{x}_i\}_b) A_{b,a}}{\omega(\{\vec{x}_i\}_a) A_{a,b}}. \tag{3}$$

But there are different choices for the acceptance probability, such as the heat bath method.

$$P_{a,b} = \frac{R_{a,b}}{1 + R_{a,b}} \tag{4}$$

Assuming $A_{a,b} = A_{b,a}$, prove the detailed balance condition for the heat bath method.

c) To integrate the function $\mathbb{R}^3 \to \mathbb{R} : f(\vec{x}) = \|\vec{x}\| \cdot e^{-\frac{\|\vec{x}\|^2}{2}}$ we perform a Monte Carlo integration by randomly drawing M independent points $\vec{x}_i \in [0, L]^3$ for $i = 1, \ldots, M$.

Write an equation for the expectation value of the estimator of $I = \iiint_0^L d^3\vec{x}\, f(\vec{x})$ and one for the expectation value of the estimatior of the error ΔI as a function of M, L and the random samples $\vec{x}_i$ only.

Question 2: Von Neumann Stability Analysis (20 points)

Given the 1D advection equation

$$\frac{\partial u}{\partial t} + C\frac{\partial u}{\partial x} = 0, \tag{5}$$

where C is a constant propagation speed, we want to use the Lax-Friedrichs finite difference scheme,

$$\frac{u_j^{n+1} - \frac{1}{2}(u_{j+1}^n + u_{j-1}^n)}{\delta t} + C\frac{u_{j+1}^n - u_{j-1}^n}{2\delta x} = 0, \tag{6}$$

to solve it numerically on a equidistant grid, where $\delta x = x_{j+1} - x_j$ is the grid spacing and $\delta t = t^{n+1} - t^n$ is the time step.

Under what conditions is this method stable?

Perform a von Neumann stability analysis and derive a stability condition on δt.

Hints:

- $e^{i\phi} = \cos\phi + i\sin\phi$
- $e^{-i\phi} = \cos\phi - i\sin\phi$
- $\cos^2\phi + \sin^2\phi = 1$
- if $z = x + iy$, then $|z|^2 = x^2 + y^2$

Question 3: Parallel Scaling (25 points)

a) Assume we implemented a simple N^2 solver for the N-body problem. The following table reports timing results for the solver for various number of particles N on P processor cores with a fixed number of time steps:

	runtime [s]			
P	$N = 500$	$N = 1000$	$N = 1500$	$N = 2000$
1	6.00	30.00	72.00	120.00
4	1.50	7.50	18.00	30.00
9	0.75	3.50	9.00	20.00
16	0.50	2.15	6.00	12.00
24	0.40	1.50	4.50	10.00

i) Draw in the figure below at least four points of the strong scaling plot for this program. On the solution sheet show all steps of your calculations. Do *not* forget to label the axes!

ii) Draw in the figure below at least four points of the weak scaling plot for this program, using the value for $N = 500$ at $P = 1$ as reference to estimate the parallelization overhead. On the solution sheet show all steps of your calculations. Do *not* forget to label the axes!

b) i) Assume you work on Euler and you have one node with 24 cores that you can use to solve a problem in parallel for which 91% of your code is parallelizable. Can you get a speedup of 8? If so, how many cores are needed at least?

ii) Profiling a serial code for Molecular Dynamics you find that 90% of the time is spent in a large loop with independent iterations (perfectly parallelizable with N threads), another 5% is spent in a region that can be parallelized with at most 2 threads and the remaining part is purely serial.
Given Amdahl's law, what is the strong scaling for $N \to \infty$?
For what value of N is the speedup equivalent to 90% of the asymptotic maximum?

c) Suppose we have the following functional units with the given latencies in a processor:

IF	(Instruction Fetch)	2 ns
ID	(Instruction Decode)	2 ns
EX	(Execute Instruction)	3 ns
MEM	(Physical Memory Access)	6 ns
WB	(Write Back)	2 ns

i) If we use these units to build a single-cycle non-pipelined processor, how long does it take to execute a single instruction?

ii) If we use these units to build our usual 5-stage pipeline processor, what is the shortest possible cycle time, i.e. the time for executing a single stage?

iii) How long does it take to execute N instructions using this pipeline, where N is some arbitrary large number?

iv) What is the speedup of this pipelined processor over the single-cycle implementation? Assume a large number of instructions that do not cause pipeline stalls.

Question 4: Roofline Model (20 points)

Given the following serial code snippet:

```
float A[N*N], B[N*N], S[N*N];
...
const int T = 1;
for (int i=0; i<N; i++)
    for (int j=0; j<N; j++) {
        float C = A[i*N+j]*B[i*N+j];
        for (int k=0; k<T; k++)
            S[i*N+j] = 0.99*S[i*N+j]*S[i*N+j] + C;
    }
```

a) What is the operational intensity of the code? State any assumptions you made. Show your calculations.

b) For Piz Daint, the theoretical peak floating-point performance per node is 332.8 Gigaflops (in single precision, excluding the GPU) while the corresponding memory bandwidth is 51.2 GB/s.

For which values of the integer variable `T` is the code of subquestion (a) compute bound? State any assumptions you made. Show your calculations.

c) Consider the performance numbers reported for Piz Daint in the previous subquestion.

i) Draw the roofline of Piz Daint in Figure 1 (next page), label the axes and put in the theoretical maximum performance of the code for T=1 and T=21.

ii) The Intel Xeon processor of Piz Daint compute nodes is equipped with 20MB L3 cache. How would the roofline plot change if the L3 cache size was 2 times larger (i.e. 40MB)?

iii) What is the theoretical peak performance in double precision?

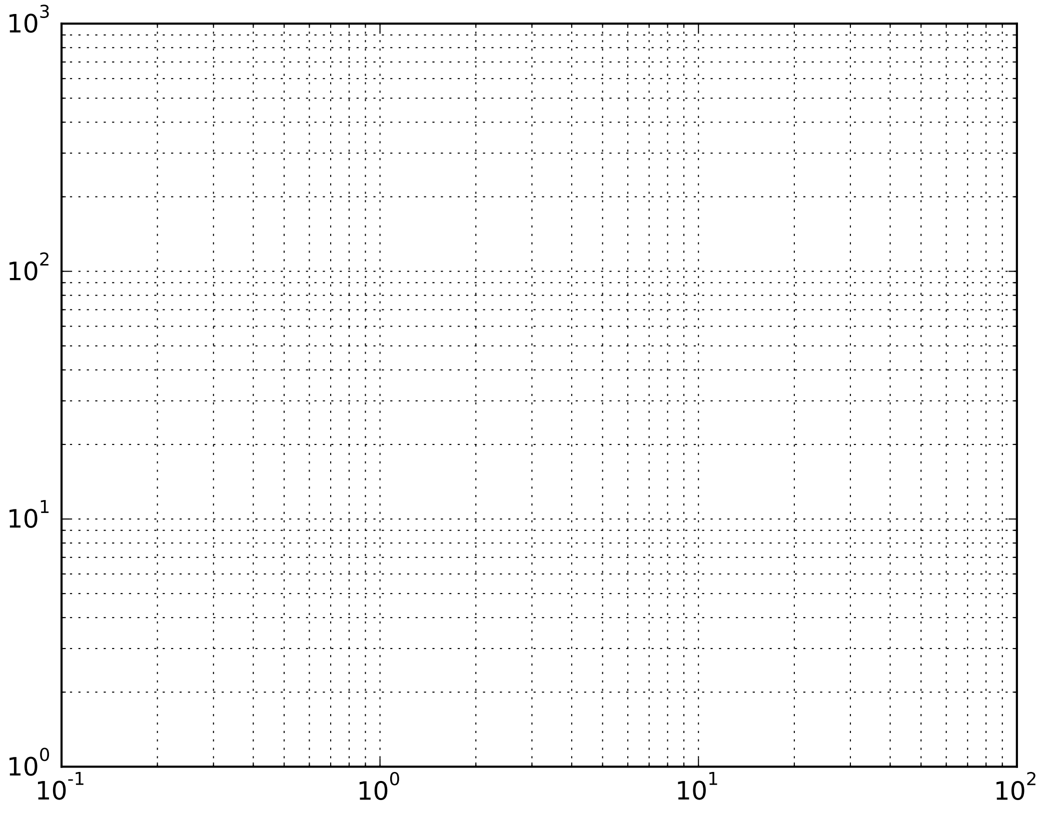


Figure 1: Roofline for Piz Daint (CPU only).

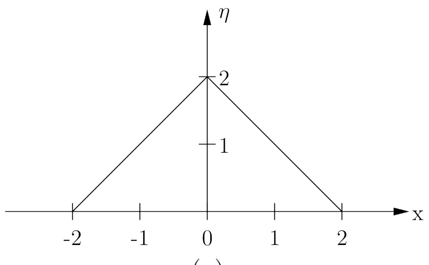
Question 5: Diffusion and its Discretization (20 points)

a) Describe at least two advantages and two drawbacks of using particle-based methods compared to grid-based methods for solving the diffusion equation.

b) Consider the method of particle strength exchange in 1D, which is given as

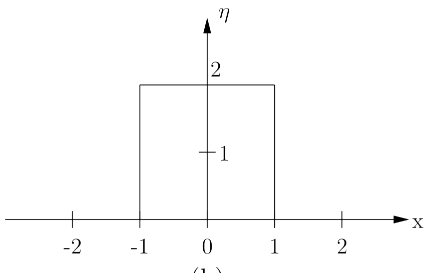
$$u_p^{n+1} = u_p^n + \frac{\nu \delta t}{\varepsilon^2} \sum_{q=0}^{N-1} \left(u_q^n - u_p^n\right) V_q \eta_\varepsilon(x_q - x_p), \tag{7}$$

where u_p denotes the strength of particle p and V_p its volume. Moreover, N is the number of particles, ν the diffusion coefficient, $\delta t = t^{n+1} - t^n$ the time-step length and ε the mollification length. Furthermore, the following kernels $\eta(x)$ are given:



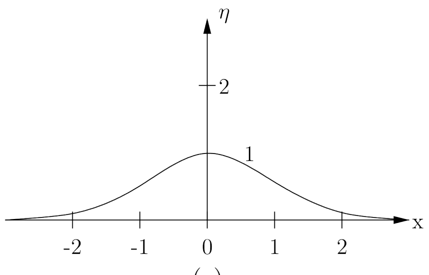
(a)

$$\eta(x) = \begin{cases} 2(1 - \frac{1}{2}|x|) & \text{if } |x| \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$



(b)

$$\eta(x) = \begin{cases} 2 & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$



(c)

$$\eta(x) = \mathrm{e}^{-\frac{x^2}{2}} \tag{10}$$

i) In terms of moments, is any of the given kernels better than the other ones? Briefly explain your answer.

ii) Consider kernel (b) and N equidistantly positioned particles p at locations $x_p = p\delta x$, where $p = 0, ..., N-1$ and $\delta x = \varepsilon$. Derive the equivalent finite-difference scheme. Make sure that you normalize the kernel appropriately. Here, we need discrete normalization which reads as $\sum_i \eta(i) i^2 = 2$.

Question 6: Parallel Bugs (25 points)

The following code snippets compile, but will show undesired runtime behavior (except for some garbled output). Assume all headers are included correctly.

a) Identify and explain any bugs in the following multithreading code using SSE intrinsics. Propose a solution.

```
constexpr size_t nThreads = 16;
std::vector<std::thread> threads(nThreads);

float * data;
// allocate 16-Byte aligned memory for 32 float values
posix_memalign(reinterpret_cast<void**>(&data), 16, 32*sizeof(float));

for (size_t i=0; i<32; ++i)
    data[i] = drand48();

for (size_t t=0; t<nThreads; ++t)
    threads[t] = std::thread(
        [&]() {
            float * const myData = data + 2*t;
            __m128 c4 = _mm_set1_ps(M_PI);
            __m128 r4 = _mm_load_ps(myData);
            __m128 a4 = _mm_mul_ps(c4, _mm_mul_ps(r4, r4));
            _mm_store_ps(myData, a4);
        });
for (auto& t : threads)
    t.join();
```

b) Identify and explain any bugs in the following OpenMP code. Propose a solution.

```
#define N 1000

struct data member[N];
int good_members[N];
int pos = 0;

void find_good_members() {
#pragma omp parallel for
    for (i=0; i < N; i++) {
        if (is_good(member[i])) {
            good_members[pos] = i;

#pragma omp atomic
            pos ++;
        }
    }
}
```

c) Identify, explain and solve the bug in the following OpenMP code. You are allowed only to modify and add OpenMP pragmas.

```
#pragma omp parallel
{
    if( omp_get_thread_num() % 2 ){
#pragma omp for schedule(dynamic)
        for( int i=0; i < N; ++i ){
            // ...
        }
    }
}
```

d) In the following MPI code, the process with rank 0 sends the value of the index i to the MPI process with rank i. Identify and explain the race condition in the code. Write in one or two sentences a possible solution to fix the problem, without modifying the asynchronous communication pattern.

```
int myid, numprocs;

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

if (myid == 0) {
    MPI_Request request[numprocs-1];
    MPI_Status status[numprocs-1];

    for (int i = 1; i < numprocs; i++) {
        MPI_Isend(&i, 1, MPI_INT, i, 123, MPI_COMM_WORLD, &request[i
            -1]);
    }
    MPI_Waitall(numprocs-1, request, status);

}
else {
    MPI_Status status;
    int received;
    MPI_Recv(&received, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &status);
    std::cout << "Process  " << myid << " received " <<  received <<
         std::endl;
}

MPI_Finalize();
```

Coding 1: MPI Datatypes (35 points)

A distributed N-Body solver employs particles with positions (x_i, y_i, z_i) and masses $m_i = m$ for $i = 0, 1, \ldots, N-1$. Since the masses stay constant during the simulation, the MPI ranks need to exchange only the updated positions. Use the skeleton code `coding_1_mpi_datatypes/mpi_datatypes.cpp` to show how you can send only the positions of the particles in the given data structure using MPI Datatypes.

a) Complete the Makefile and the first part of the skeleton: create an MPI Datatype containing only the positions of all particles. Use the Datatype to send all positions from rank 0 to rank 1 with a single Send/Recv pair.

b) Complete the second part of the skeleton. This part uses a different data structure to represent the particle's positions. Create an MPI Datatype for the positions of all particles in this data structure. Receive the positions on rank 1 using the new data structure from rank 0 using the old data structure. *Note:* The Send command for rank 0 is the same as the previous task.

Write your solution in
`~/results/coding_1_mpi_datatypes/mpi_datatypes.cpp`
and adapt the `Makefile` *in the same folder to compile your program using* `make`.

Coding 2: Particle-To-Mesh Operations (25 points)

We study a continuous field of pollutants $u(x)$ carried by N particles. Assume a $1D$ domain $[0,1]$, that is discretized by an equispaced grid $x_i^m = i \cdot \delta x$, where $\delta x = 1/M$ and $i = 0, 1, \ldots, M-1$. The particles are randomly distributed in the domain $[0,1]$. The pollution concentration carried by particle j, located at position x_j^p, is denoted by u_j^p.

We wish to interpolate the field carried by the particles onto the grid $u_i^m = u(x_i^m)$ by using the scheme

$$u_i^m = \sum_{j=0}^{N-1} u_j^p W(\lambda_j), \tag{11}$$

where $\lambda_j = \frac{x_j^p}{\delta x} - i$ and $W(\lambda_j)$ is some interpolation kernel. For this problem we use the M_4' kernel, which is given as

$$W(\lambda) = \begin{cases} 1 - \frac{5}{2}\lambda^2 + \frac{3}{2}|\lambda|^3 & \text{if } 0 \leq |\lambda| < 1, \\ \frac{1}{2}(2 - |\lambda|)^2(1 - |\lambda|) & \text{if } 1 \leq |\lambda| < 2, \\ 0 & \text{if } |\lambda| \geq 2. \end{cases} \tag{12}$$

The skeleton code provided for this problem includes the following:

- structure `Particle` to store position and pollution concentration of a particle,
- mesh defined as a vector of size M storing pollution concentrations $u_i^m = u(x_i^m)$, which are initially set to zero,
- vector of particles of size N, where particles are seed at random positions x_j^p with concentration $u_j^p = \sin\left(2\pi x_j^p\right)$ depending on their location,
- sequential algorithm implemented in `main` for particle to mesh interpolation.

a) Implement the computation of the weights for the M_4' kernel according to equation (12).

b) The sequential particle-to-mesh (P2M) algorithm interpolates pollution concentration values from the particles to the mesh via a scattering approach, using the M_4' kernel and considering periodic boundaries.

   Use manual C++11 threads to parallelize the P2M algorithm.

Write your solution in
`~/results/coding_2_particle_to_mesh_operations/p2m.cpp`
and adapt the `Makefile` *in the same folder to compile your program using* `make`.

Coding 3: Diffusion Statistics (25 points)

In `coding_3_openmp_diffusion/diffusion2d_openmp.cpp` you find a simplified version, with slightly modified initial conditions, of the solution OpenMP code for the 2D diffusion problem of homework 3.

a) Provide a OpenMP parallel implementation of the diffusion kernel in function `advance()` .

b) Implement the sequential diagnostics function `compute_max_density()` which prints the maximum density value and its location.

c) Provide a parallel OpenMP implementation of the previous code in the function `compute_max_density_omp()`. Note: try to keep the number of memory accesses close to that of the sequential version.

Write your solution in
`~/results/coding_3_openmp_diffusion/diffusion2d_openmp.cpp`
and adapt the `Makefile` *in the same folder to compile your program using* `make`.

Good luck!