P. Koumoutsakos, M. Troyer
ETH Zentrum, CLT F 12
CH-8092 Zürich

Fall semester 2014

Set 11 - Exam collections

Issued: December 5, 2014
Hand in: December 12, 2014, 8:00am

In this exercise sheet we propose some exercises that were asked in previous exams. Try to solve them using only the material that we allow during the exam (see lecture homepage) to train yourself.

For this week you will not receive any hint, but your are welcome to ask all your questions when we discuss the solutions.

Disclaimer: Some of the questions have been simplified to fit the topic of this year lecture.

Question 1: Performance of Cell Lists (HS 2013)

a) Assume you are working in a 2D computational domain with $(x, y) \in [0, 1]^2$ and your pairwise interactions have a cutoff of 0.08. What is the maximum number M of cells that you could use?

b) Assume you have N particles and that the cost (or time) to compute a single pairwise interaction between them is c_{pw}. This means that the time required to compute x pairwise interactions is xc_{pw}. What is the cost t_{naive} to compute the pairwise interactions between all particles? Consider that each particle does not interact with itself and that the interactions are not symmetrical.

c) If we were to use a cell list with periodic boundary conditions, we would first need to go through all particles to create the cell list and then we would only compute interactions of a particle with particles in its own cell and neighboring ones. Assume that the N particles are nicely distributed over all cells so that each cell contains $N_c = N/M$ particles. Furthermore assume that the cost to put a single particle in a cell when creating the cell list is c_{cl} and the cost to compute a single pairwise interaction between 2 particles is again c_{pw}. What is the cost t_{cl} of creating the cell list and of computing pairwise interactions between all particles within the interaction range using a cell list, considering no self-interactions and no symmetry?

d) Derive for what number of particles N it is better not (!) to use a cell list using your results from questions 1b and 1c. You can assume $M > 9$ and that we create the cell list at each time-step.

e) Cell list can be used to effectively handle a cutoff in the Lennard-Jones potential for the N-body problem, and similarly to handle the cutoff in the kernel of the Particle Strength Exchange (PSE) method. The benefit of using a cell list in the PSE method is however larger than in the N-body problem. Why?

Question 2: Roofline Model (HS 2012)

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 $N = 80$.

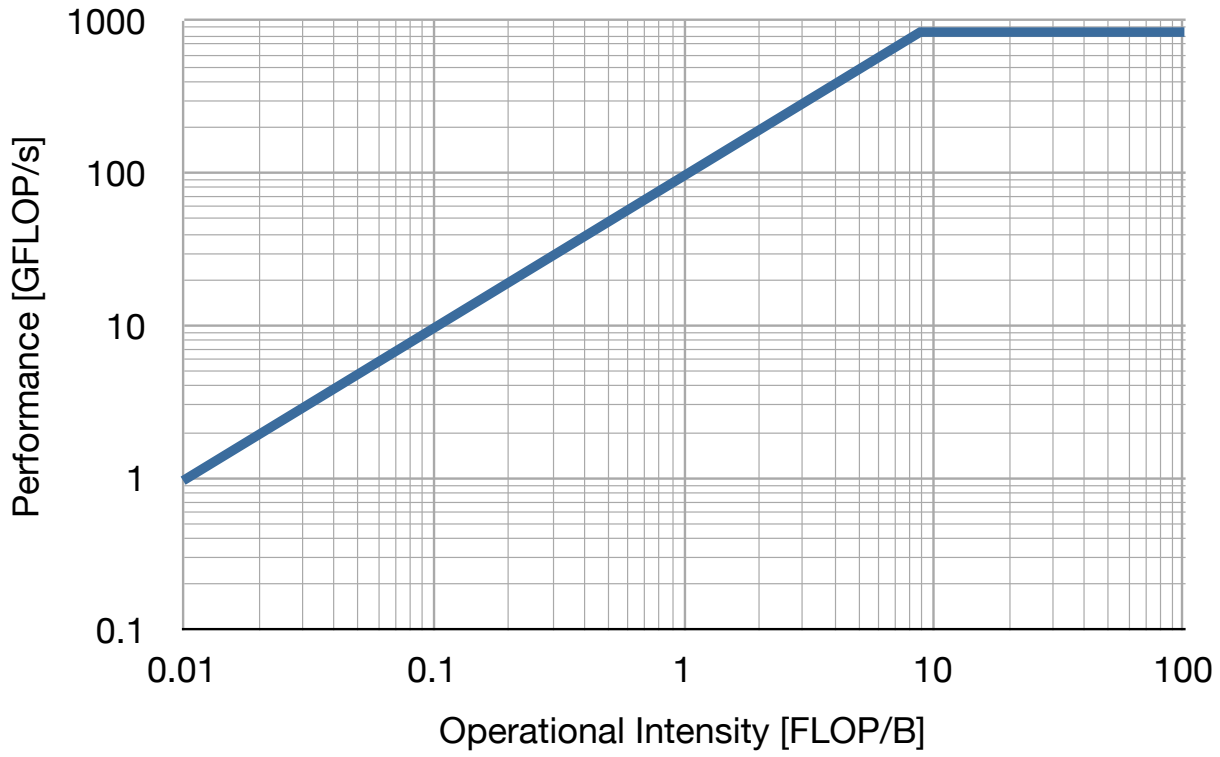

Figure 1: Roofline for Brutus.

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.

Question 3: Parallel sum (HS 2012)

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)

```

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


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 using 4 threads
    ...


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

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

c) using MPI. You may assume that N is a multiple of the number of processes. In order to compile and run MPI code, you will first need to load the corresponding module:

```
$ module load mpi/mpich-x86_64
```

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>
#include <iostream>

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

```
16
17      // SET UP COMMUNICATION.
18      // DETERMINE num_processes AND OUR rank
19      // ...
20
21
22      // initialize local parts of the vectors and do the
          sum z = x + y
23      assert( N % num_processes == 0 );
24      int nlocal = N / num_processes;
25      std::vector<float> x(nlocal,-1.2), y(nlocal,3.4), z(
          nlocal);
26      for( int i = 0; i < nlocal; i++ )
27          z[i] = x[i] + y[i];
28
29      if( rank == 0 )
30      {
31          std::vector<float> fullz(N);
32
33
34          // COLLECT ALL PARTS INTO fullz
35          ...
36
37
38          // print result checksum
39          std::cout << std::accumulate( fullz.begin(), fullz
              .end(), 0. ) << std::endl;
40      }
41      else
42      {
43          // SEND LOCAL PART z TO ROOT PROCESS
44          ...
45      }
46
47      // CLEAN UP
48      ...
49  }
```

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

Summary

Summarize your answers, results and plots into a PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.