Prof. Dr. Peter Bastian, Santiago Ospina De Los Ríos
IWR, Universität Heidelberg

Submission date: 2024-01-16

Exercise 1 *N-body: MPI+X+SIMD*

In the lecture you have seen a MPI parallel solver for the n-body problem. In this exercise we want implement a version that combines all modes of parallelisms seen in the lecture by extending it.

1. Try to build and run the `nbody_mpi.cc` file. You will need a working MPI implementation like `OpenMPI` or `MPICH` for this. After successfully building the executable you can run the program using

```
mpirun -np <number_of_processes> ./nbody_mpi <basename> <nbodies> <timesteps> <timestep> <every>
```

2. Modify the MPI implementation to use Structure of Arrays (SoA). **Note:** You need to change how the messages sent and reveiced by MPI.

3. Extend the n-body code to be MPI+X by combining message passing and shared memory parallelism. That is, each MPI process uses multiple threads to work on its part of the problem. Use the shared memory library of your preference.

4. Optimize for an specific target hardware architecture.
   In Moodle (section **Scientific Computing Group Compute Node**) you should have gotten access to a compute node with a two socket AMD Epyc 7713 (codename *Milan*) together with usage instructions. Compute nodes of this kind are now commonly used for scientific computing purposes, so use this node as your target architecture in the following sub-tasks. Figure 1 shows the core and memory topology of Milan. To compile code in Milan load the required modules as shown below

```
# loads gcc-11 and openmpi and oneapi-tbb
source ~/source_gcc11.sh
# compile source code using openmpi
mpic++ <compilerflags> nbody_mpi.cc
```

   The goal of thi exercise is to get the highest performance on this compute node. Thus, find a strategy for mapping shared memory threads and MPI processes that favors your target hardware architecture. Report your findings and make sure that the results are still correct after optimizations (checking that output files look similar in paraview than in sequential version is sufficient).

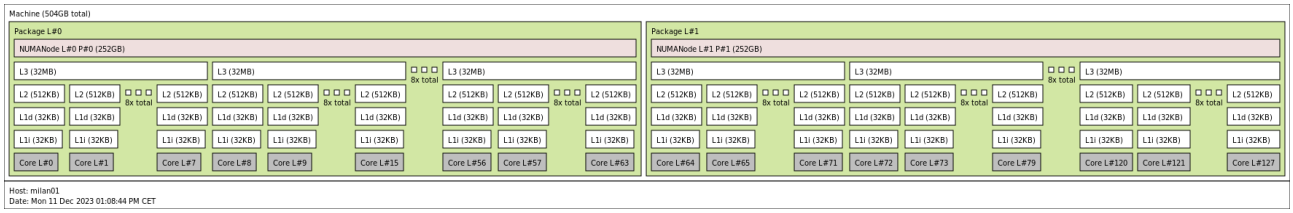

Figure 1: Memory and CPU topology of Milan with NPS1. NUMA Node Per Socket (NPS) is a configurable parameter in this architecture that changes the topology. This specific figure has NPS=1 and Simultaneous Multithreading (SMT) disabled.

The following are some ideas of how to acheive the desired maximum performance:

(a) Pinning: Learn how to pin threads to physical cores by using `likwid`, `hwloc` or by directly invoking the Operating System[2] (e.g., `sched_setaffinity` in Linux).

[2]Pinning is only possible if the operating system allows you to do so. For example, darwin, the kernel of macOS has no such feature. On the other hand, Windows and Linux allow this.

(b) SIMD: Extend the new SoA n-body code to have MPI+X+SIMD. For this, vectorize the body acceleration updates with SIMD instructions as we have done in exercise sheet 4. Use your own implementation or one of the reference solutions. Make sure that the performance for a single thread is acceptable. Use pinning if necessary.

(c) NUMA Domain: First optimize the shared memory implementation for a single memory domain. For this, consider the case where only one MPI process is executed and all children threads are pinned to cores on the same NUMA domain. Then, find the configuration of your threading library that yiels the maximum througput. Note that the latency among all the cores of may vary even within the same CPU (See figure 2), so use pinning to minimize the commuication overhead between threads.

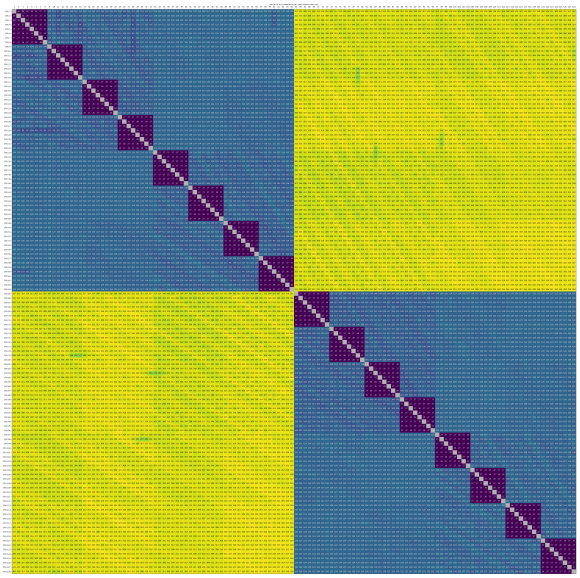

Figure 2: Latency of a Compare and Swap instruction between cores of AMD Epyc Milan. Each value represents the latency of sending and receiving a message between two cores. It depends on whether the cores are in the same chiplet (27ns), in different chiplet but the same socket (125ns), or in different sockets (315ns).

(d) NUMA: Extend the last case to the full compute node. Consider the situation where *one* MPI process is completely assigned to a node with more than one NUMA domain, e.g., there is only one MPI process running for the two sockets of a compute node like Milan. Find a configuration for the MPI processes and child threads that maximizes throughput. To assess different alternatives use pinning or use a NUMA-aware partitioner/scheduler (e.g., `oneapi::tbb::affinity_partitioner`) and initialize the memory using a first touch approach if necessary.

The best 2 groups with highest performance for roughly 100'000 masses on 128 CPU cores of Milan will get a prize!

Note: The compute node that we provided for you has no scheduling system! This means that you need to communicate with other groups in the discussion forum if you want to make a performance measurment. Try to run relatively short simulations and do not disturb other group measurments. Use `htop` to monitor the usage of the machine.

(0+4+6+10 Points)