Project - Methods in High Performance Computing

Perttu Jääskeläinen
perttuj@kth.se

Yujie Chen
yujiec@kth.se

Karlis Kristofers Velins
velins@kth.se

June 5, 2023

1 Introduction

This report contains the final project[1] related to the course course DD2356 - Methods in High Performance Computing, at KTH, 2023.

We chose the iPIC3D[6][4] application to evaluate performance, scalability and identify possible bottlenecks (Network, I/O, MPI), with suggestions for improvements.

The application was run on Dardel using process counts up to 1024 - to measure how well the application scales and performs when increasing parallelization. The process counts were distributed between 1-8 nodes, depending on what the total process count was. To compile the application, we utilized both Cray MPICH[1] and OpenMPI[7] to evaluate differences between two different MPI implementations. The related source code can be found on GitHub[3].

2 Group Member Contributions

In addition to the detailed lists below, all members worked together and contributed through discussions and coding session to plan, analyze and agree on conclusions and metrics measured.

- **Perttu Jääskeläinen**
  - Report Structure, Presentation
  - Batch and input file configuration
  - Performance/Scaling Measurements, Analysis (Cray, OpenMPI)

[1]https://canvas.kth.se/courses/38959/pages/final-project-requirements

- **Yujie Chen**
  - Network, I/O and MPI analysis
- **Karlis Kristofers Velins**
  - Bottleneck, Optimization analysis

3 Project Goals and Methodology

The goal of the project is to apply our knowledge from the course material in a real-world application - we do this by measuring how well the iPIC3D application performs and scales on the nodes of Dardel. We run the application by compiling with both OpenMPI and Cray MPICH for varying amounts of Dardel nodes (up to a maximum of 8 nodes with a total of 1024 MPI processes).

To analyze how well the application performs we gather metrics using the CrayPat[2] tool, which samples the program counter in intervals when the application is run - in order to identify which functions consume the most time, performance metrics, time spent on MPI and I/O and more, which are included in the output file when the execution is finished. The important metrics we want to evaluate are:

- Performance - is there a difference when increasing nodes/processes/using different compilations of the application?
- Scalability - how well does increasing processes/nodes scale the performance
- Identify bottlenecks - computation, network, MPI and/or I/O
- Suggestions for optimizations

4 Experimental Setup

To gather the metrics, we cloned the iPIC3D[4] application while signed in to Dardel and created batch files for different amounts of processes. To test the performance, we ran the **testGEM3Dsmall.inp** file with some parameters changed (to make sure to avoid known issues with CrayPat/iPIC3D).

To gather the metrics we used CrayPat[2], which was integrated into the **cmake** script which already existed in the iPIC3D application. When a run finished, a file similar to iPIC3D+xxxxx-xxxxxxxs was written to disk along with an output file. We used the iPIC3D+... file with **pat_report** to generate tables for the batch run, such as which functions took most of the execution time and what the distribution between MPI/computation was.

The hardware setup was between 1-8 Dardel nodes for different process counts. We maximized the amount of processes depending on the cores available, so we didn't, for example, always use 8 nodes regardless of process counts -

in order to minimize communication across nodes. Each performance measurement was performed using the **main** partition on Dardel. We also made some minor code modifications in order to be able to run - we removed the **WriteOutput** from **iPIC3D.cpp** to avoid I/O errors, and replaced **MPI_type_hvector** with **MPI_type_create_hvector** so that we could run with OpenMPI. All input/output, batch-, iPI3D+... and modified .cpp files can be found on our GitHub[3].

5 Results

We measured execution time for process counts 64 (1), 128 (1), 256 (2), 512 (4) and 1024 (8) processes (nodes). We did this using both Cray MPICH[1] (the default module loaded in Dardel), as well as OpenMPI[7] by re-compiling the code with the correct module loaded. For each metric, we took the average value for 3 performed runs. See section 6 for an analysis of the results.

5.1 Dominant functions

We identified the following two functions that make up a majorty of the computation time:

1. Particles3D::mover_PC_AoS, primarily compute-bound. The function involves a loop that performs multiple iterations to calculate the average velocity of particles iteratively.

2. EMfields3D::sumMoments_AoS, primarily compute-bound. The main computational workload comes from the loop over the particles (nop iterations) and the subsequent calculations of moments and weights.

The dominant MPI functions are presented in figure 1, along with the time spent on computation/MPI calls in terms of percentages of total run time.

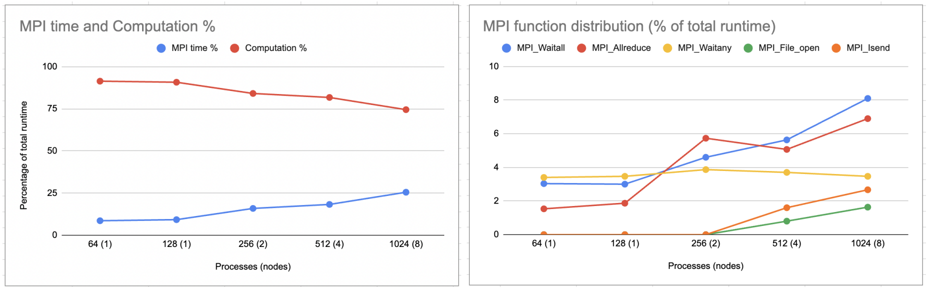


Figure 1: Distribution for MPI/computation functions during runtime, expressed as percentages of total runtime.

5.2 Communication Pattern

In this section we present a collection of figures and tables related to the communication patterns and network topology in the simulations. See the labels for

Imb.Samp%	64	128	256	512	1024
mover_PC_AoS	5.5	4.7	8.1	10.7	10.8
sumMoments_AoS	2.6	4.0	4.0	5.5	7.8

Figure 3: Process imbalance for the computation functions. The imbalance for MPI functions was insignificant and not presented here.

each figure on details of what they represent.

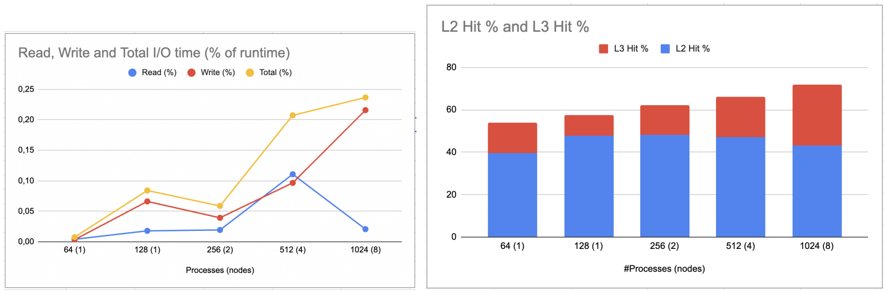


Figure 2: Distribution of I/O operations as a percentage of total runtime; L2 and L3 hit rate. I/O never becomes significant (less than 1%), and hit-rate increases as we increase the amount of nodes.

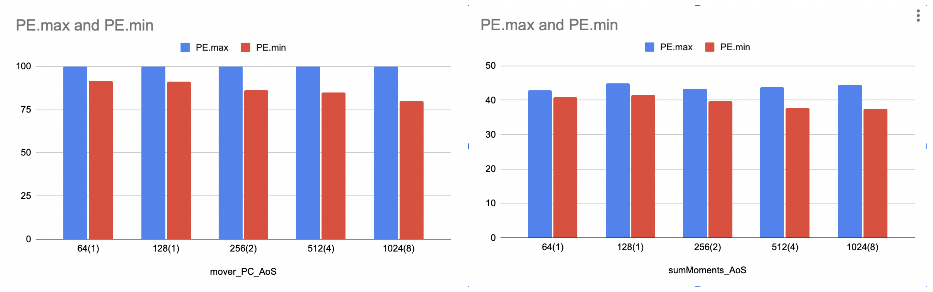


Figure 4: Max and min number of samples of PEs for two main functions (as percentage). We can see that the difference between min/max is more significant for n=512,1024 (with up to 25% difference).

The communication topology was distributed among the X/Y/Z axises in the following structure for varying process counts (with PeriodicityX/Z=1, Y=0):

Topology Map	64	128	256	512	1024
Processors grid (X/Y/Z)	4x4x4	8x4x4	8x8x4	8x8x8	16x8x8

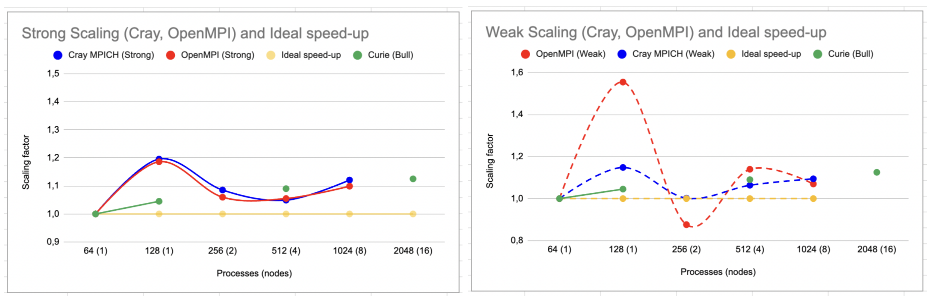


Figure 5: Performed Strong/Weak Scaling tests on Dardel. The results indicate bad scaling for iPIC3D for Cray MPICH, with slightly better results for OpenMPI - with quite similar results to those acquired on Curie. We have some stronger deviations for the weak scaling (for OpenMPI) which could likely be avoided by performing further scaling tests and re-computing deviations/averages. The scaling factor used was computed using S = 2*T(n)/T(n-2).

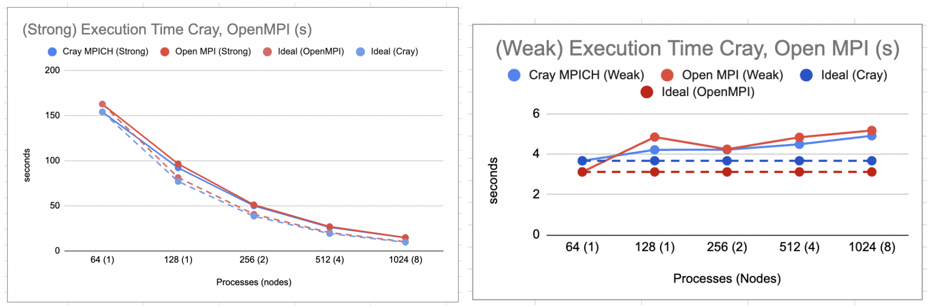


Figure 6: Execution times for weak/strong scaling tests. We can see that while it is not optimal, it performs quite well when considering network and cross-node constraints - with Cray outperforming OpenMPI.

5.3 Scaling Efficiency - OpenMPI vs. Cray MPICH

In this section we present figures relating to scaling tests performed on Dardel - both for OpenMPI and Cray.

6 Analysis

6.1 Identified Bottlenecks and Proposed optimizations

Function Profiling: The table "Profile by Function" shows each function's percentage of samples and imbalance. We used the 256 nodes results for this specific analysis. The "USER" group has the highest sample percentage (80.8%), indicating that most of the time is spent on user-defined functions rather than system functions. We will shift our focus to the 2 largest bottleneck functions:

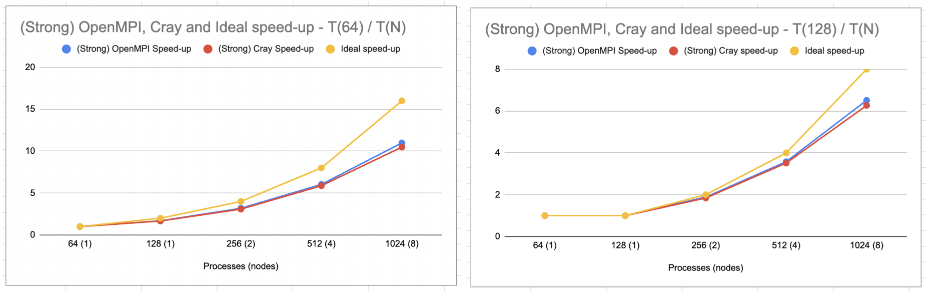
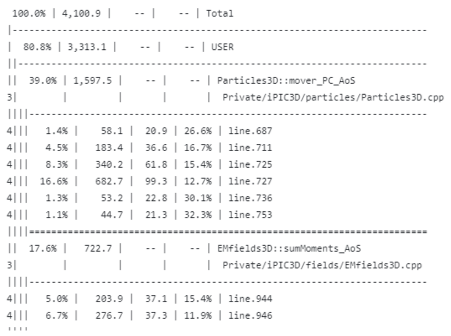


Figure 7: Speed up of the execution time compared to the ideal case - using both T(64) and T(128) as the base (i.e. 1 node).

Particles3D:mover-PC-AoS and EMfields3D:sumMoments-AoS



```
 100.0% | 4,100.9 |     -- |     -- | Total
|----------------------------------------------------------------------------
|  80.8% | 3,313.1 |     -- |     -- | USER
||---------------------------------------------------------------------------
||  39.0% | 1,597.5 |     -- |     -- | Particles3D::mover_PC_AoS
3|        |         |        |        |  Private/iPIC3D/particles/Particles3D.cpp
||||-------------------------------------------------------------------------
4|||   1.4% |    58.1 |  20.9 | 26.6% | line.687
4|||   4.5% |   183.4 |  36.6 | 16.7% | line.711
4|||   8.3% |   340.2 |  61.8 | 15.4% | line.725
4|||  16.6% |   682.7 |  99.3 | 12.7% | line.727
4|||   1.3% |    53.2 |  22.8 | 30.1% | line.736
4|||   1.1% |    44.7 |  21.3 | 32.3% | line.753
||||=========================================================================
||  17.6% |   722.7 |     -- |     -- | EMfields3D::sumMoments_AoS
3|        |         |        |        |  Private/iPIC3D/fields/EMfields3D.cpp
||||-------------------------------------------------------------------------
4|||   5.0% |   203.9 |  37.1 | 15.4% | line.944
4|||   6.7% |   276.7 |  37.3 | 11.9% | line.946
''''
```

Figure 8: Profile by Group, Function, and Line

Particles3D:mover-PC-AoS: This function is used to move particles in a three-dimensional space. A large bottleneck is in the sampled field calculation. It is already using pragma simd which is enabling vectorization. Our proposed optimization is to introduce loop unrolling to process multiple iterations of the loop body in a single iteration. Loop unrolling reduces loop overhead and allows for better utilization of SIMD instructions

```
1: for $i = 0$ to $num\_field\_components$ do
2:     $sampled\_field[i] weights_c \times field\_components_c[i]$
3: end for
```

EMfields3D:sumMoments-AoS: This function calculates moments of particle data and accumulates them into field arrays. The function itself is very basic with only 80 iterations, but since it gets called many times they stack up and contribute to the overall performance decrease. An optimization we propose is loop vectorization. The aim would be to align data and enable compiler optimizations to allow for loop vectorization. We could achieve this using the pragma omp simd flag like it was done in the sampled-field calculation.

```
for $c = 0$ to 7 do
    for $m = 0$ to 9 do
        $momentsArray[c][m]velmoments[m] \times weights[c]$
    end for
end for
```

6.2 Result Analysis

6.2.1 MPI, Network, I/O and Process Imbalance

For both functions for process imbalances (seen in figure 4), the imbalance tends to increase as the number of processes increases. This means that in larger node configurations, certain processes have relatively higher execution time or resource utilization for the functions compared to other processes.

The cache hits rates presented in figure 2 indicate that increasing the number of nodes generally leads to a higher cache hit rate. This is because more cache space is available, reducing the need to access the main memory. With more nodes, each node has its own cache (L2 and L3 cache), providing more space to store data and instructions, increasing the likelihood of cache hits.

The dominant MPI functions in terms of execution time were mostly the same regardless of process count, but deviated slightly for larger counts. For process counts = 512/1024, MPI_File_open and MPI_Isend started to become significant, while MPI_Waitall and MPI_Allreduce and MPI_Waitany were all significant for all counts. This is understandable since the computation is distributed among more processes while the problem size stays the same - resulting in a higher percentage for MPI calls of total run-time as we increase the amount of processes.

6.2.2 Performance, Scalability and Speed-up

The scaling factor results presented in figure 5 are quite similar to previous results on Curie[5], but not optimal - sometimes having an increased runtime of 20%. The execution times for the different tests presented in figure 6 indicate that the execution time is almost optimal - with Cray slightly outperforming OpenMPI. Since Dardel is configured as an HPE Cray supercomputer, these metrics make sense. However, we did notice some deviations in run time for the different test runs we did, so it would be interesting to run further tests (if given more time, ex. average of 10 runs) to get a better grasp of the average run-time and identify deviations.

Similarily, Cray outperforms OpenMPI in speed-up (presented in figure 7), with both not being fully optimal. This also makes sense, since we're dealing with MPI/network overhead for larger process counts - so ideal speed-up is not very likely. However, we argue that the speed-up performs well, since it is within 10-20% of the ideal speed-up for the tests performed.

7 Conclusions

In conclusion, we successfully evaluated the scalability and performance of the iPIC3D application on Dardel, identified bottlenecks, and proposed optimizations. Cray seems to be preferred over OpenMPI for larger scale problems which require more nodes (outperforms OpenMPI for almost all process counts). Both Cray and OpenMPI perform worse than ideal in speed-up/scaling, while optimally we would like for it to grow sublinearly (for our scaling factor) or superlinearly (for speed-up) - but this seems consistent with previous tests done for the iPIC3D application (on Carie (Bull)[5]). Since we encountered deviations for runtimes in certain tests, it would also be interesting to further run thests to compute deviations and averages more accurately.

The findings can guide future improvements to enhance the scalability, efficiency, and overall performance of the iPIC3D application on high-performance computing systems like Dardel. We would also like to test the scaling tests for even larger problem sizes/nodes to see if OpenMPI keeps up with the better scaling factor compared to Cray MPICH.

References

[1] *Cray MPICH - default and preferred MPI implementation on Cray systems.* URL: `https://docs.nersc.gov/development/programming-models/mpi/cray-mpich/`.

[2] *Craypat - performance analysis tool provided by Cray.* URL: `https://docs.nersc.gov/tools/performance/craypat/`.

[3] *Final Project - GitHub repository.* URL: `https://github.com/perttuj/dd2356-high-performance-computing`.

[4] *iPIC3D - three-dimensional parallel PIC (Particle-in-cell) code.* URL: `https://github.com/KTH-HPC/iPIC3D`.

[5] Stefano Markidis. *The iPIC3D code and its applications to 3D magnetic reconnection.* URL: `https://indico.fysik.su.se/event/4460/attachments/`.

[6] Stefano Markidis, Giovanni Lapenta, and Rizwan-uddin. "Multi-scale simulations of plasma with iPIC3D". In: *Mathematics and Computers in Simulation* 80.7 (2010). Multiscale modeling of moving interfaces in materials, pp. 1509–1519. ISSN: 0378-4754. DOI: `https://doi.org/10.1016/j.matcom.2009.08.038`. URL: `http://www.sciencedirect.com/science/article/pii/S0378475409002444`.

[7] *OpenMPI - open source Message Passing Interface implementation developed and maintained by a consortium of academic, research and industry partners.* URL: `https://www.open-mpi.org/`.