SPH-EXA performance assessment report

Document Information

Reference Number	POP2_AR_023
Author	Fabian Orland (RWTH)
Contributor(s)	Radita Liem, Joachim Protze, Bo Wang
Date	November 6, 2019
Application	SPH-EXA
Service Level	Performance Audit
Keywords	MPI

Notices: *The research leading to these results has received funding from the European UnionâĂŹs Horizon 2020 research and innovation programme under grant agreement No n°676553.*

Contents

1 Background

Applicants Name: Florina Ciorba
Application Name: SPH-EXA
Programming Language: C++
Programming Model: MPI, (OpenMP), (OpenACC)
Source Code Available: No
Performance study: Performance audit
Application description: SPH-EXA performs hydrodynamical and computational fluid dynamics simulations using the smoothed particle hydrodynamics (SPH) method. In this method the fluid domain is discretized using a set of particles (1,000,000 in this case). Properties stored at these particles are interpolated over the fluid domain using smoothing kernel functions. Such a kernel function has an associated smoothing-length so that only a finite number of neighboring particles (here max. 500) needs to be considered in the interpolation.
Input data: Automatic generation of input conditions using parameters: -n 210 -s 80 -w -1 (ref-small) and -n 475 -s 130 -w -1 (ref-medium) . The generated scenario is a rotating square fluid patch of dimension $n \times n \times n$. The number of simulation steps is given by $s + 1$. If w is chosen larger than zero the simulation will dump particle information into a file ater every wth simulation step.
Machine Description: CLAIX-2016 cluster at RWTH Aachen University. A single node is a two-socket system each equipped with a 12-core Intel Xeon Broadwell running at 2.20 GHz.
Environment used: Intel compiler 19.0.1.144, Intel MPI Library 2018 Update 4

2 Application structure

The application consists of three phases:

- Initialization
- SPH simulation loop
- Finalization

The SPH simulation loop can be further split into different computational subparts:

1. distribution of particle data
2. synchronization of halo data
3. octree construction
4. determination of particle's neighborhood information
5. computation of density
6. computation of pressure using equation of state (EOS)
7. synchronization of halo data
8. computation of momenta and internal/kinetic energy
9. determination of stable timestep size
10. time integration of particle's positions
11. computation of total energy

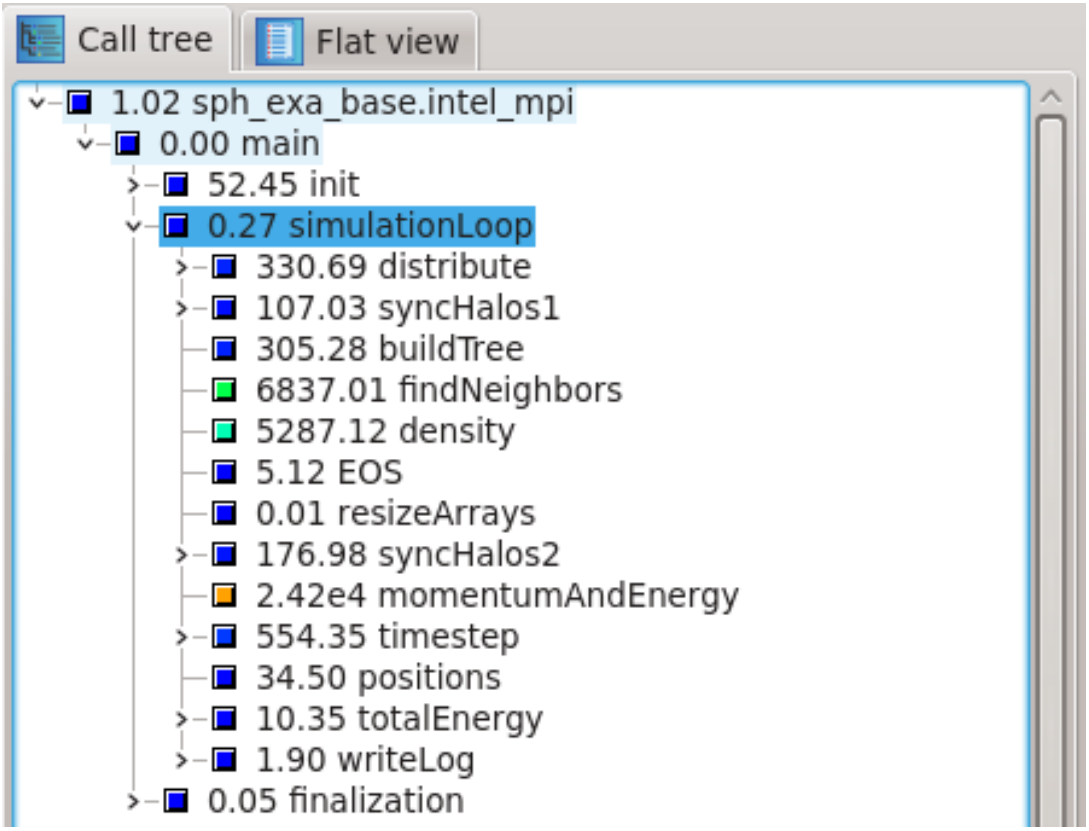

Figure 1: Call tree profile of the simulation loop showing the accumulated exclusive time spent inside different sub-functions over all processes. Data was obtained during a full run of SPH-EXA-small on one node of CLAIX2016 using 24 MPI processes.

This structure can be recognized in profiling results obtained with Score-P + Scalasca shown in figure 1. The call tree shows the accumulated time spent in the three phases init, simulation loop and finalization over all processes. Execution time is mainly dominated by the SPH simulation loop. 99.86 % of the execution time is spent inside the simulation loop. The initialization and finalization phases are negligibly short. They only account for 0.14 % and, respectively, 0.0032 % of the whole execution time. Figure 2 shows a trace of a full SPH-EXA-small execution run

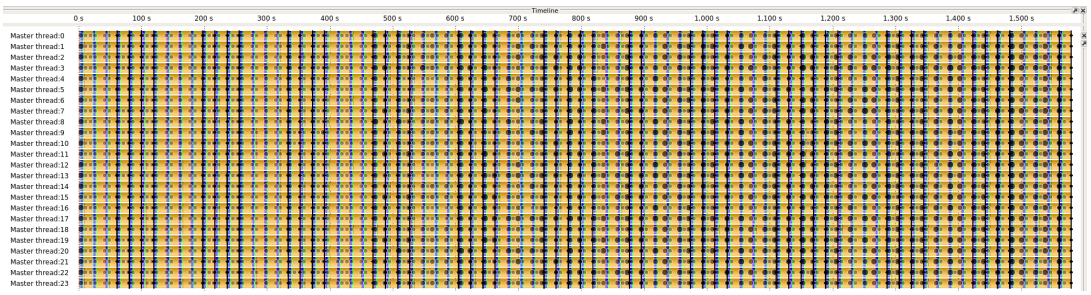

Figure 2: Visualization of tracing results obtained with Score-P + Scalasca for a full execution run of SPH-EXA-small on a single node of CLAIX-2016 using 24 MPI processes.

collected with Score-P + Scalasca on a single node of CLAIX-2016 using 24 MPI processes. Since the initialization and finalization phases are extremely short one can only recognize a regular pattern of iterations. Each vertical blue bar indicates the start of such an iteration.

3 Focus of Analysis (FOA)

Almost all time is spent in the simulation loop. Hence we focus our analysis on this part of the code. More specifically we focus on an individual iteration exemplarily. Figure 1 shows a call tree profile for the simulation loop. The applied metric is exclusive time, which is accumulated over all MPI processes in the profile. Most of the time is spent inside the `momentumAndEnergy` calculation. 63.86 % of the overall execution time is spent in this part of the code. Moreover, finding the neighbors and computing the density for each particle are also parts of the code where a significant amount of time is spent. Computing the neighborhood information accounts for 18.03 % of the whole runtime. Similarly, the density computation also takes 13.94 % of the overall execution time. With this knowledge in mind we will look specifically into these regions of the code.

4 Scalability

We measured scalability of SPH-EXA-small for different numbers of MPI processes on a single node and two nodes of CLAIX-2016. We run SPH-EXA-small using 6, 12, 24 and 48 MPI

Nodes	Ranks	Runtime [sec]	% MPI of runtime	speedup	efficiency
1	6	5788	1.02 %	1	1
1	12	3048	0.82 %	1.90	0.95
1	24	1584	1.98 %	3.65	0.91
2	48	860	8.59 %	6.73	0.84

Table 1: Strong-scaling measurements for SPH-EXA-small on one node of CLAIX-2016 with 6, 12 and 24 MPI processes and on two nodes with 48 processes.

processes. For speedup calculations we use the run with 6 ranks as a reference. Table 1 shows runtime, MPI runtime share, speedup and efficiency results. The efficiency value is calculated as the ratio between the runtime of the reference run and the runtime of the run with a larger number of ranks. So an efficiency value of 1 indicates perfect scalability. These results are visualized in Figure 3. Doubling the number of ranks from 6 to 12 results in a speedup of 1.90 and an efficiency value of 0.95 which shows that this is nearly optimal. Going from 6 to 24 ranks gives a speedup of 3.65 and an efficiency of 0.91. This is slighty lower than before, however, still near the optimum. Increasing the number of ranks from 6 to 48 by a factor of 8 yields a speedup of 6.73 with an efficiency of 0.84. Again the efficiency drops a little bit but is still considered to be acceptable. On a single node at most 2 % of the runtime are spent inside MPI. This significantly increases to 8.59 % on two nodes because now data needs to be transmitted over the network.

Similarly, we measured scalability of SPH-EXA-medium on multiple nodes of CLAIX-2016. The medium version of SPH-EXA was run on 10, 20 and 40 nodes with a total of 240, 480 and 960 MPI processes, respectively. Scalability results are shown in table 2 and visualized in Figure 4. We use the run on 10 nodes with 240 MPI ranks as a reference to calculate the speedup. With 480 ranks we get a speedup of 1.84 with an efficiency value of 0.92. This is a nearly optimal speedup. When using 960 ranks a speedup of 3.53 can be observed. The corresponding efficiency value is 0.88. Again this results in a speedup near the optimum. For all three runs, between 8.52 % and 16 % of the runtime is spent inside MPI communication which is mainly caused by the collective operation `MPI_Allreduce`.

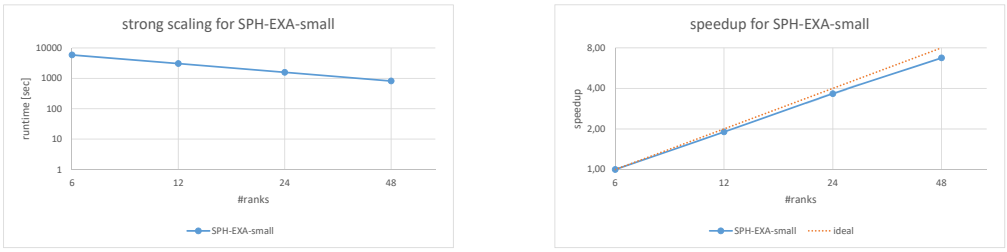

Figure 3: Visualization of strong-scaling and speedup results for SPH-EXA-small on one node of CLAIX-2016 with 6, 12 and 24 MPI processes and on two nodes with 48 processes.

Nodes	Ranks	Runtime [sec]	% MPI of runtime	speedup	efficiency
10	240	3342	8.52 %	1	1
20	480	1815	14.79 %	1.84	0.92
40	960	946	16.00 %	3.53	0.88

Table 2: Strong-scaling measurements for SPH-EXA-medium on 10, 20 and 40 nodes of CLAIX-2016 with 240, 480 and 960 MPI processes.

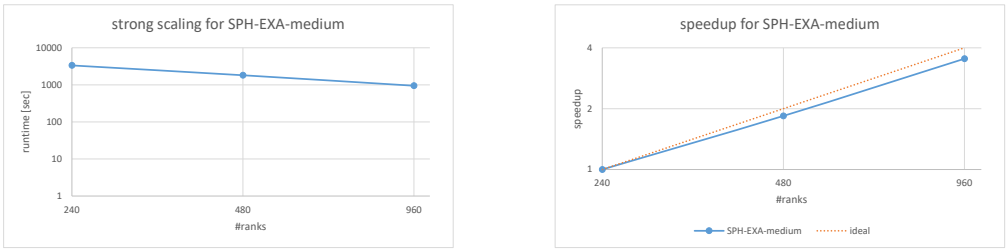

Figure 4: Visualization of strong-scaling and speedup results for SPH-EXA-mediuml on 10,20 and 40 nodes of CLAIX-2016 with 240, 480 and 960 MPI processes.

5 Efficiency

Several metrics[1] are defined in the POP methodology. These metrics cover different aspects commonly causing inefficiencies in parallel programs. Their values range from 0 to 1, where higher values are better. They are organized in a hierarchical way as follows: At the top of the hierarchy is the **Global Efficiency**. It is the product of the **Parallel Efficiency** and the **Computational Efficiency**.
The **Parallel Efficiency** measures how well the application is parallelized in terms of data distribution among processes and communication between them. It is the product of two sub-metrics, namely **Load Balance Efficiency** and **Communication Efficiency**. **Load Balance**

[1] https://pop-coe.eu/node/69

is the ratio between average computation time (across all processes) and maximum computation time (across all processes). It measures how evenly computational work is distributed among processes. **Communication Efficiency** is another composite efficiency which combines the **Serialization Efficiency** and the **Transfer Efficiency**. **Serialization Efficiency** is the ratio between the maximum computation time on an ideal network and the total runtime on an ideal network. It is a measure for waiting times within communications. The **Transfer Efficiency** covers inefficiencies caused by data transfers. It is defined as the ratio between the total runtime on an ideal network and the total runtime on the real network.
For the **Computational Efficiency** metric values are compared to a reference case in terms of scalability. In this case always the lowest processes count is used as a referenced. The **Computational Efficiency** is also a composite efficiency. It is composed of the **Instruction Efficiency** and the **IPC Efficiency**. The **Instruction Efficiency** is given as the ratio between total number of useful instructions in the reference case and the total number of useful instructions with more processes than in the reference case. Similarly, the **IPC Efficiency** is the ratio of IPC in the reference case to the IPC in a case with more processes.
Table 3 illustrates the resulting metrics for an SPH-EXA-small run on a single node of CLAIX-

	6 ranks	12 ranks	24 ranks	48 ranks
Global Efficiency	0.97	0.99	0.98	0.98
Parallel Efficiency	0.97	0.99	0.98	0.97
Load Balance Efficiency	0.98	0.99	0.98	0.98
Communication Efficiency	0.99	0.99	0.99	0.98
Serialization Efficiency	0.99	0.99	0.99	0.99
Transfer Efficiency	0.99	0.99	0.99	0.99
Computation Efficiency	1	0.99	0.97	0.96
IPC Efficiency	1	1	0.98	0.97
Instruction Efficiency	1	0.99	0.99	0.986

Table 3: POP metrics reported by the Cube tool based on profiling data obtained with Score-P + Scalasca for an execution run of SPH-EXA-small on a single node of CLAIX-2016 using 6-24 MPI processes and on two nodes with 48 processes.

2016 with 6, 12 and 24 MPI processes and on two nodes with 48 processes. Metrics are computed for the whole simulation loop. Overall the application achieves very high values near the optimum of 1.00 for each of the defined metrics.

We also computed the metrics for a run of SPH-EXA-medium on 10, 20 and 40 nodes of CLAIX-2016 using 240, 480 and 960 MPI processes. The results are shown in table 4. When running SPH-EXA-medium with larger numbers of MPI processes we notice lower **Global Efficiency** values compared to the SPH-EXA-small runs. Using 240 MPI processes the application achieves a **Global Efficiency** of 0.92. With 480 ranks this efficiency value drops to 0.85. In the POP methodology we consider efficiencies above 0.8 as acceptable. Nevertheless, we should identify the cause of inefficiency in this case by further looking into the hierarchy of metrics. The **Computational Efficiency** is nearly optimal with 0.99. **IPC Efficiency** is perfect and there is only a very small inefficiency in the instruction scaling. However, the **Parallel Efficiency** is only at 0.86. Looking at the different submetrics the **Parallel Efficiency** is composed of we recognize that the **Load Balance Efficiency** is the lowest with 0.92. With 960 ranks a **Global Efficiency** of 0.83 can be measured. Compared to the run with 480 ranks the **Load Balance Efficiency** and the **Transfer Efficiency** are slighty lower by 1 %. However, the **Communication Efficiency** and the **Serialization Efficiency** are slightly higher. The **Computation**

	240 ranks	480 ranks	960 ranks
Global Efficiency	0.92	0.85	0.83
Parallel Efficiency	0.92	0.86	0.86
Load Balance Efficiency	0.95	0.92	0.91
Communication Efficiency	0.97	0.93	0.94
Serialization Efficiency	0.98	0.94	0.97
Transfer Efficiency	0.99	0.99	0.98
Computation Efficiency	1	0.99	0.96
IPC Efficiency	1	1	0.99
Instruction Efficiency	1	0.99	0.97

Table 4: POP metrics reported by the Cube tool based on profiling data obtained with Score-P + Scalasca for an execution run of SPH-EXA-medium on 10, 20 and 40 nodes of CLAIX-2016 using 240, 480 and 960 MPI processes.

Efficiency loses 3 % due to an increased number of useful instructions and a slightly lower **IPC Efficiency**.

6 Load Balance

As already shown by table 4 the **Load Balance Efficiency** is at 0.92 when running SPH-EXA-medium on 20 nodes of CLAIX-2016 with 480 MPI processes and at 0.91 when running on 40 CLAIX-2016 nodes with 960 MPI processes. In both cases the inefficiency in load balancing has the largest impact on the **Global Efficiency** of 0.85 and 0.83 respectively compared to the other SPH-EXA runs. Figure 5 shows a trace of a single iteration of an SPH-EXA-medium run on twenty node of CLAIX-2016 with 24 MPI processes each. The iteration was chosen randomly in the middle of the execution. The start of the iteration is indicated by the cyan blue intervals on the left. These correspond to the time spent inside the `distribute` function call of the `distributedDomain` object, where particle data is distributed across MPI ranks. There does not seem to be a huge load imbalance for this function based on the results shown in Figure 5. Computing the **Load Balance Efficiency** for the `distribute` function (for all iterations) yields a value of 0.95 which supports the first visual impression.
The next part of the execution is the synchronization of halo data (light brown) followed by `MPI_Recv` (red) and `MPI_Waitall` (dark red). Based on the trace data visualized in Figure 5 it looks like there are also some load balance issues. Again computing the **Load Balance Efficiency** for the `synchronizeHalos` part results in 0.74. However, since synchronizing halo data only accounts for 0.57 % of the whole runtime of the simulation loop, this notable inefficiency should not have a huge impact on the overall load balance.
After halo data is exchanged a tree structure is build as a support structure for finding the neighbors in the next step. In Figure 5 the `buildTree` function is colored in magenta. There is a significant load imbalance in building the tree. There are lots of processes which spent approximately up to three times more time with building the tree than others. A fair efficiency value of 0.50 for the load balance for this part of the code confirms the visual finding.
As a consequence the processes that spent more time with building the tree can also start later with finding the neighbors (dark blue) and computing the density (light green). The individual values for the **Load Balance Efficiency** are 0.87 for the `findNeighbors` function and 0.95 for

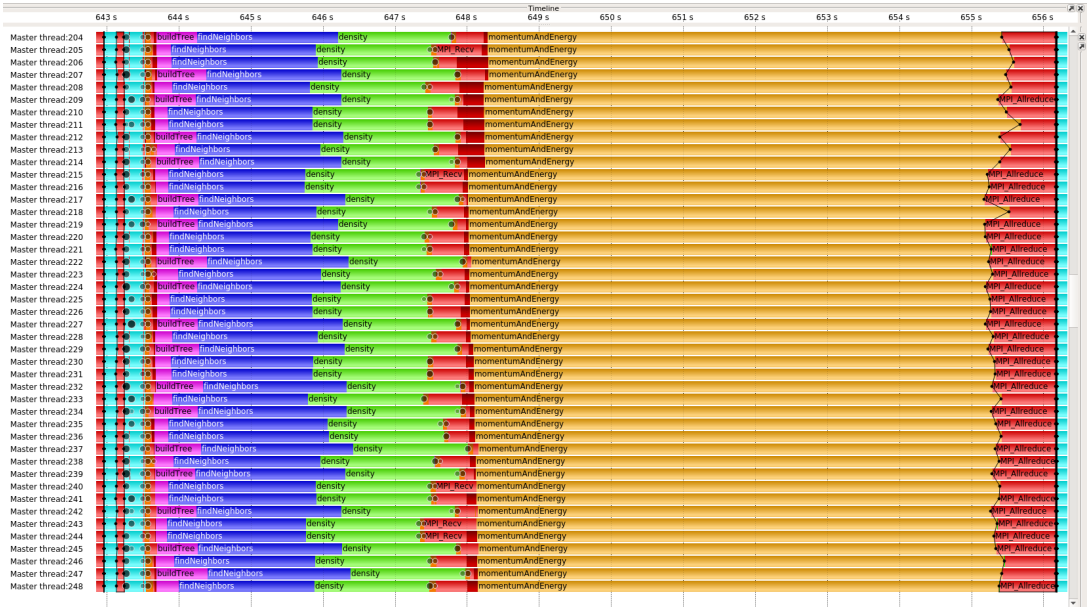

Figure 5: Trace of a single simulation loop iteration for an SPH-EXA-medium run on 20 nodes of CLAIX-2016 using 480 MPI processes. Functions are color coded. MPI calls are colored red while application code is colored in different colors other than red.

the `computeDensity` function. Both are good values, especially the second, but have room for some optimization. However, this might be difficult since particles dot not have all the exact same number of neighbors. So some load imbalance might be inherent to the problem itself and can probably not be avoided.

After the density computation pressure values are determined by an equation of state. Since the EOS part has very short execution times it cannot be recognized in the trace. After that another synchronization step is required and halo data is exchanged again (light brown) which is then followed by some `MPI_Recv` (red) and `MPI_Waitall` (dark red). Comparing both synchronization passes in Figure 5 the second one looks quite imbalanced compared to the first. Due to the imbalance caused by the `buildTree` function processes that finished earlier are spending more time waiting in their `MPI_Recv` calls because other processes are still computing the density and may not have posted the corresponding `MPI_Isend`. Scalasca's automatic trace analysis provides a metric for late MPI senders. It reports that 2.12 % of the overall runtime can be attributed to `MPI_Isend` calls that happen after the corresponding `MPI_Recv` was posted in the second halo synchronization pass. Furthermore, it indicates that the `MPI_Isend` calls do not happen in the same order as they are expected by the corresponding `MPI_Recv` calls. An efficiency value of 0.78 for the second call of `synchronizeHalos` indicates that there is some optimization still possible.

Next, momentum and energy are computed (orange). On the right of Figure 5 one can recognize some smaller load imbalances. Some processes finish earlier than others and call `MPI_Allreduce` as part of the `timestep` function following the momentum and energy computation. The **Load Balance Efficiency** for computing momentum and energy is 0.85, which is still acceptable but shows that there are small load imbalances.

Updating the positions and computing the total energy takes nearly no time and is hence, not visible in the trace in Figure 5. So they can safely be neglected in terms of load balancing.

Table 5 summarizes the computed load balance metric for the different stages of the simulation loop.

function	Load balance efficiency	runtime contribution
distribute	0.95	5.89 %
synchronizeHalos1	0.74	0.57 %
buildTree	0.50	2.63 %
findNeighbors	0.87	14.47 %
computeDensity	0.95	11.54 %
computeEquationOfState	0.81	0.01 %
synchronizeHalos2	0.78	0.49 %
computeMomentumAndEnergy	0.85	52.73 %
computeTimestep	0.94	0.01 %
computePositions	0.88	0.07 %
computeTotalEnergy	0.91	0.01 %

Table 5: Load balance efficiency values computed for the different stages of the simulation loop. Data was obtained during a run of SPH-EXA-medium on 20 nodes of CLAIX-2016 with 480 MPI processes.

7 Serial performance

Based on the **Computational Efficiency** reported in table 3 we can measure the serial performance of SPH-EXA-small. As a reference case we use the run with 6 MPI ranks. Optimally, the **Computational Efficiency** stays 1 when increasing the number of ranks. However, the efficiency slightly decreases down to 0.96. This indicates that the number of instructions increases as the number of processes is increased.
Moreover, we can look at the *instructions per cycle (IPC)* values hidden behind the **IPC Efficiency**. The theoretical optimum for the processors of Claix-2016 is 4 instructions per cycle. Typically, this value cannot be achieved for real applications. Based on POP experience any value above 1 is good. IPC values for the simulation loop as well as for the most time-

nodes	ranks	avg. IPC	findNeighbors	computeDensity	computeMomentumAndEnergy
1	6	1.63	1.66	1.27	1.70
1	12	1.63	1.65	1.27	1.70
1	24	1.60	1.61	1.26	1.68
2	48	1.58	1.58	1.26	1.67

Table 6: Instructions per cycle for different runs of SPH-EXA-small on one node of CLAIX-2016 with 6, 12 and 24 MPI processes and on two nodes with 48 processes.

consuming parts of the code are shown in table 6. For all runs the average IPC is between 1.58 and 1.63 which is very good. For the most time-consuming part of the code, the computation of `momentumAndEnergy`, IPC values between 1.67 and 1.70 can be observed. Similarly, for the `findNeighbors` function the IPC is between 1.58 and 1.66. Only for the computation of the density the IPC is slightly lower with 1.26 and 1.27. However, computing the density is computationally much simpler than computing momentum and energy. It is basically a weighted sum of the scalar masses of neighboring particles. In contrast to that computing momentum works on 3D data and requires generally more operations on the same data than computing the density. Still the IPC is above 1 for the density computation. So this is still a good result.
For SPH-EXA-medium we get similar results. The run with 240 ranks is taken as a reference for the other runs. As table 4 shows the **Computational Efficiency** slightly drops to 0.99

nodes	ranks	avg. IPC	`findNeighbors`	`computeDensity`	`computeMomentumAndEnergy`
10	240	1.56	1.56	1.25	1.67
20	480	1.56	1.57	1.26	1.69
40	960	1.55	1.57	1.27	1.70

Table 7: Instructions per cycle for different runs of SPH-EXA-medium on 10,20 and 40 nodes of CLAIX-2016 with 240, 480 and 960 MPI processes.

when using 480 MPI ranks. This is because the **Instruction Efficiency** is also slightly lower than 1 with 0.99 indicating that slightly more instructions were needed for this run than for the reference run. For 960 ranks we get a **Computational Efficiency** of 0.96. The reasons for this is again the **Instruction Efficiency** which shows that slightly more instructions were performed compared to the reference run. Furthermore, table 7 shows IPC values for important parts of the code similar to table 6. For all three runs the average IPC is 1.55 or 1.56 which is only slighty lower than for the runs with SPH-EXA-small. For finding the neighbors the IPC is around 1.56. For the density computation the IPC is roughly 1.25 which is again a little bit lower than for other parts of the computation but still greater than 1. Finally, for the computation of momentum and energy the IPC is highest with values between 1.67 and 1.70. Again these values are slighty lower than the ones of SPH-EXA-small but they are significantly larger than 1 in the computationally most demanding part of the code which is very good.

8 Communications

At the start of each simulation loop iteration the application uses two collective `MPI_Allreduce` operations inside the `distribute` function. One determines the maximum smoothing length globally and the other one how many particles are in each spatial bucket.
Moreover, a lot of MPI communication happens inside the `synchronizeHalos` function. This synchronization is necessary directly after distributing the particle data and after computing the density because the density values are required to compute momentum and energy. For exchange of halo data at the end of the `distribute` function each process computes a list of ranks which it needs to send data to and a list of ranks from which data will be received. After that in the `synchronizeHalos` call each process will asynchronously send data to its corresponding neighbor processes and then recieves the required data from neighboring processes.
For the runs of SPH-EXA-small we measured a **Communication Efficiency** between 0.98 and 0.99 (see table 3) which is already nearly optimal. The **Communication Efficiency** drops slightly for the runs of SPH-EXA with a medium sized workload. We measured values between 0.94 and 0.97. This is also almost optimal. The application already uses asynchronous communication. Every send operation is a `MPI_Isend`. Moreover, a distributed graph communicator is created so that the MPI implementation can optimize for this special topology. MPI ranks that are neighbors of each other in the distributed graph are also physically placed closed together onto compute nodes. Therefore, the communication matrix in Figure 6 shows that each process communicates only with a small number of neighboring processes. The coloring is mostly symmetric. Looking at the data row by row shows that all the rows look quite similar. This indicates that each process sends an equal number of messages to other processes. Computing a balance metric for the bytes sent in MPI Point-to-Point communication based on profiling data obtained with Score-P/Scalasca yields a value of 0.86. This shows that the processes do not send the exact same amount of bytes but this imbalance is acceptable.

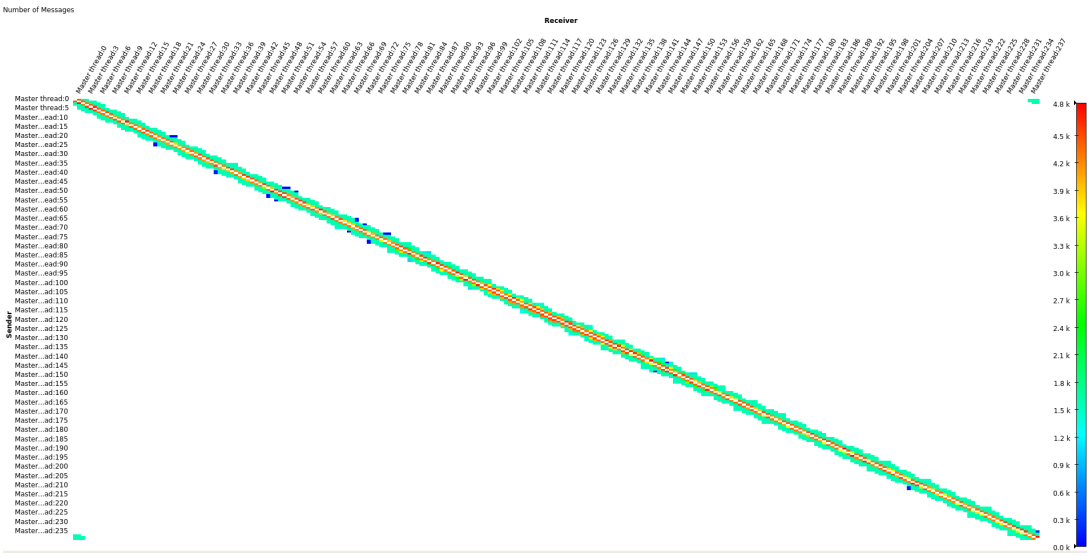

Figure 6: Visualization of communication between individual processes for a run of SPH-EXA-medium on 10 nodes of CLAIX-2016 with 240 MPI processes.

9 Summary of observations

In this report we analysed different execution runs of the SPH-EXA application on the CLAIX-2016 cluster. For SPH-EXA-small the scalability up to 48 MPI ranks is mostly nearly optimal with efficiency values of 0.84 and higher. A similar scalability behavior is measured for SPH-EXA-medium. Up to 480 MPI ranks on 20 nodes speedup is nearly optimal with an efficiency of 0.92. For 960 ranks on 40 nodes we observed a speedup of 3.53.
All the calculated POP metrics achieve nearly optimal values of 0.97 and higher for SPH-EXA-small. However, for SPH-EXA-medium we observed a small load imbalance which decreases the **Global Efficiency** to 0.85 and 0.83 respectively. This inefficiency is mostly caused by the `buildTree` function which only achieves a **Load Balance Efficiency** of 0.50. One should have a look into that because this severe load imbalance happens quite early in each iteration and thus negatively impacts the performance of the whole iteration. The synchronization of halo data also achieves load balance efficiency values below 0.80 with 0.74 for the first synchronization pass of each iteration and 0.78 for the second. Here the tools indicate that the `MPI_Isend` do not happen in the same order as they are expected by the sequence of `MPI_Recv` on the receiving processes.
In terms of communication over all execution runs SPH-EXA achieves very good values between 0.93 and 0.99 for the **Communication Efficiency**.
Regarding serial performance we observed IPC values significantly higher than 1 over the whole simulation. Especially in the most time-consuming parts like `momentumAndEnergy`, `findNeighbors` and the `density` computation comparable IPC values are observed. Only for the density computation these values are a little bit lower but still above 1.