SPHYNX
Performance Assessment Report

Document Information

Reference Number	POP_PP_17
Author	Michael Wagner (BSC)
Contributor(s)	
Date	27.03.2018

Notices:
The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "676553".

Content

1. Background

Applicants Name:	Rubén Cabezon (University of Basel)
Application Name:	SPHYNX (version 1.4)
Programming Language:	Fortran
Programming Model:	MPI + OpenMP
Source Code Available:	Yes (not accessed)
Input data:	evrard collapse
Performance study:	Evaluate new version of the code.

The application was monitored by the applicant on Piz Daint at CSCS a Cray XC50 system based on Intel Xeon E5-2690 with 12 cores per node and on Mare Nostrum 4 (MN4) at BSC, a system based on Intel Xeon Platinum 8160 with 48 cores per node (2x 24 cores per chip). The applicant recorded traces on Piz Daint using 1, 2, 4, 8, and 16 nodes, i.e. 12, 24, 48, 96, and 196 cores, respectively. In addition, he recorded traces on MareNostrum 4 using 1, 2,4, and 8 nodes, i.e. 48, 96, 192, and 384 cores, respectively. All measurements are recorded in a hybrid MPI+OpenMP mode with one MPI rank per node and 12 (Piz Daint) or 48 (MN4) threads per MPI rank and node. The traces are used to study the scalability in a strong scaling setup. All traces were collected with Extrae 3.5.2 using detailed trace mode with no sampling and recording of hardware counters in three sets changing every 0.5 seconds.

This performance plan evaluates SPHYNX in version 1.4. The results can be compared with the results for version 1.3 (POP audit, POP_AR_86) and version 1.3.1 (POP performance plan, POP_PP_16). The comparison highlights parts of the improvements originating from the close collaboration.

2. Application Structure

Figure 1 depicts the timeline of the execution using 48 cores, i.e. four nodes at Piz Daint. The colour gradient from green to blue represents the duration of the compute phases. After a short initialization (orange) the measurement contains 100 iterations, whereas all iterations show relatively similar behaviour. At the end appears a short finalization phase (orange).

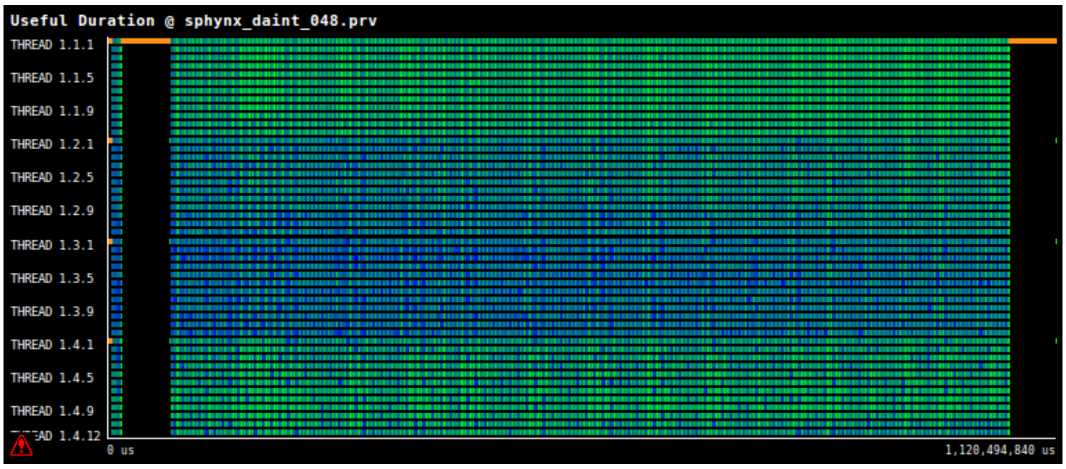
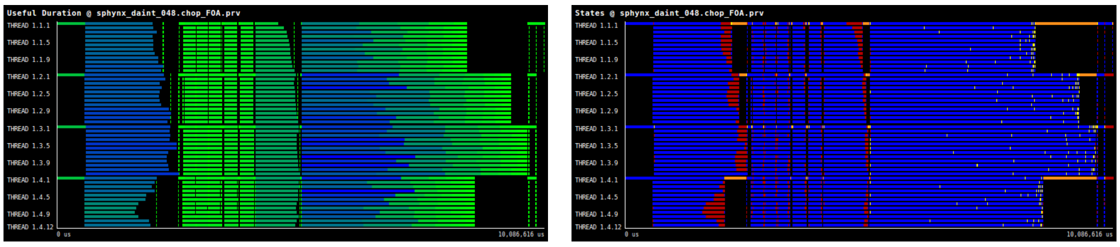


Figure 1. Application structure in a timeline view using 48 (Piz Daint) cores.

3. FOA (Focus of Analysis)

The iterations show only small deviations along time. To exclude variable effects from the initialization phase and towards finalization we selected an iteration from the middle of the execution as focus of analysis (FOA). Figure 2 depicts the distribution of computation phases (left) and the parallel behaviour (right) of the FOA, whereas blue signals computation phases, red intra-node (OpenMP) synchronization, and orange global inter-node (MPI) communication/synchronization.



Useful duration **Parallel behaviour**

Figure 2. Focus of Analysis (FOA) using 48 cores (4 nodes).

4. Scalability

Figure 3 highlights the scalability of the FOA. It shows the execution structure of the compute phases of the FOA for Piz Daint and MN4; whereas the time is normalized to 100% of the FOA duration. One of the effects that stands out is the increasing relative time that is spent only on the MPI ranks (master threads), i.e. most of the cores are idle. Since, the measurements on MN4 use 48 threads per MPI rank this effect is amplified.

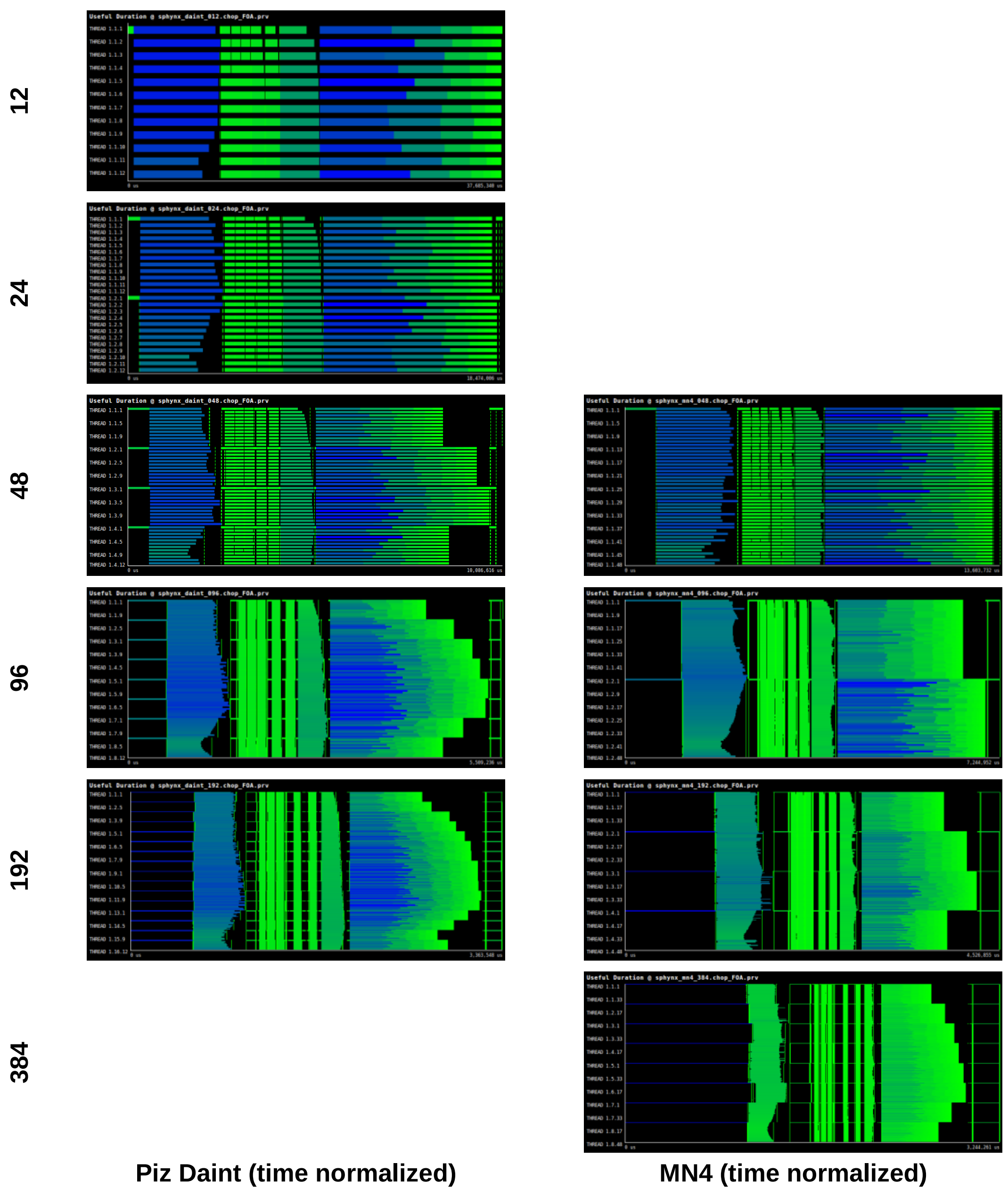
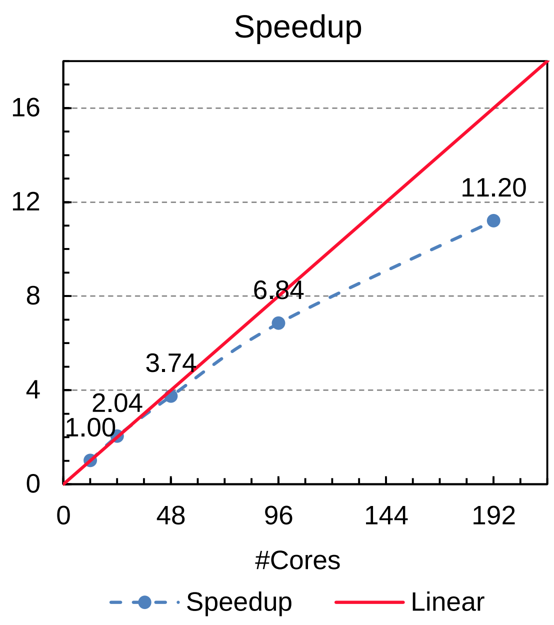
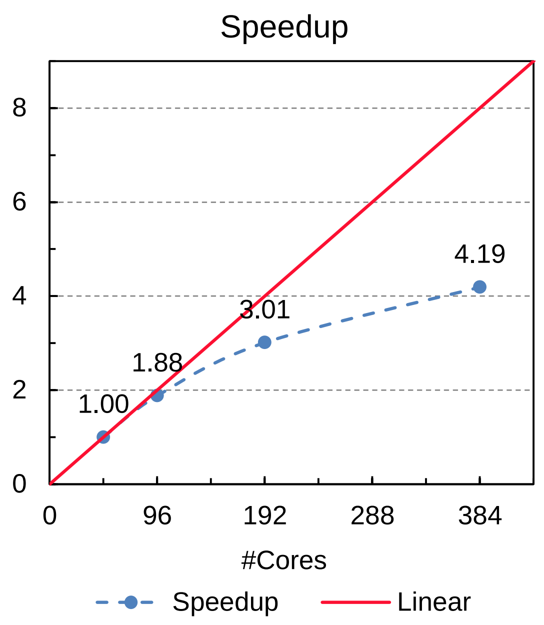


Figure 3. Scalability of FOA. Timeline of executions phases.

Figure 4 depicts the speed up of the FOA in comparison to the smallest run with 12 (Piz Daint) or 48 (MN4) cores. In a perfectly linear strong scaling execution we expect that each time the number of cores doubles, the total execution time of the FOA reduces by half (red line on the Speedup Figure 4). The overall scaling for Piz Daint is good with 11.2 out of 16 (70.0%), while the scaling for MN4 is only fair with 4.19 out of 8 (52.4%).





Piz Daint **MN4**

Figure 4. Scalability of FOA. Speedup charts.

5. Efficiency

Table 1 to Table 4 show metrics for fundamental factors and efficiencies from the FOA of the executions for Piz Daint and MN4. Values are in percentages with higher values being better.

The observed global efficiency for Piz Daint decreases steadily from 93.9% with 12 cores to 65.8% with 192 cores. The decreasing global efficiency is mainly caused by a decreasing load balance that is already rather low for the smallest measurement causing also a rather low global efficiency for the smallest run. The low load balance is primarily caused by high idle times of the worker threads. For MN4 the global efficiency is decreasing more drastic, mainly due to the more decreasing load balance.

	12	24	48	96	192
Parallel Efficiency	**93.90%**	**87.51%**	**75.14%**	**67.77%**	**57.48%**
↳ Load Balance	98.19%	92.39%	80.77%	73.82%	64.07%
↳ Comm. Efficiency	95.63%	94.71%	93.02%	91.81%	89.72%
Computation Scalability*	**100.00%**	**109.44%**	**116.73%**	**118.46%**	**114.38%**
Global Efficiency	**93.90%**	**95.77%**	**87.70%**	**80.29%**	**65.75%**

Table 1. Time efficiencies for the FOA at Piz Daint.

	12	24	48	96	192
IPC Scalability*	100.00%	110.04%	118.33%	121.99%	121.33%
Instructions Scalability*	100.00%	99.45%	98.64%	96.98%	94.00%

Table 2. Other efficiencies for the FOA at Piz Daint.

** Reference values are based on the measurement with 48 cores.*

	48	96	192	384
Parallel Efficiency	**81.06%**	**65.96%**	**48.57%**	**34.43%**
↳ Load Balance	92.60%	68.24%	52.41%	37.55%
↳ Comm. Efficiency	87.54%	96.66%	92.68%	91.70%
Computation Scalability*	**100.00%**	**115.37%**	**125.38%**	**123.40%**
Global Efficiency	**81.06%**	**76.10%**	**60.90%**	**42.49%**

Table 3. Time efficiencies for the FOA at MN4.

	48	96	144	192
IPC Scalability*	100.00%	116.00%	127.42%	127.40%
Instructions Scalability*	100.00%	99.41%	98.83%	97.25%

Table 4. Other efficiencies for the FOA at MN4.

** Reference values are based on the measurement with 48 cores.*

6. Load Balance

The observed measurements show generally a decreasing low load balance with an increasing number of cores. Figure 5 highlights the load balance in a timeline for 384 cores at MN 4, whereas the colours represent the main execution states: compute phases (blue), MPI collective communication (orange), thread synchronization (red), thread fork/join (yellow), and idle threads (black).

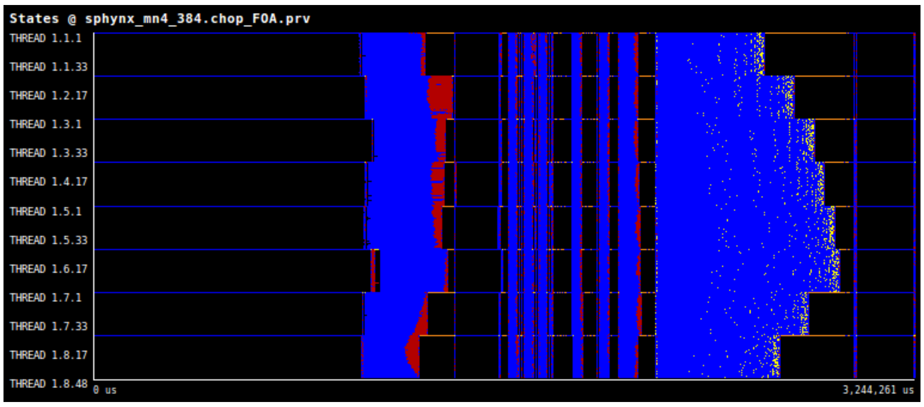


Figure 5. Load balance of the FOA for 384 cores at MN4.

The main reason for the large imbalance is the amount of time that the worker threads are idling. Tables 5 and 6 offer a bit more detail on the load balance by presenting the load balance between the nodes and within the nodes. They show a good load balance between the nodes but a low load balance within the nodes. The intra-node balance comes from two sources: the imbalance between the worker nodes, i.e. phases were all threads are working and the imbalance between the master and worker threads caused by phases where only the master thread is active and the worker threads are idling. Whereas the imbalance due to idle times is the most severe.

Thus, the amount of idle time is a primary target for optimization, allowing for a potential speedup of more than two at the given scale.

	12	24	48	96	192
Total load balance	98.19%	92.39%	80.77%	73.82%	64.07%
↳ Inter-node	100.00%	99.19%	90.58%	90.68%	91.10%
↳ Intra-node	98.19%	93.14%	89.17%	81.41%	70.32%

Table 5. Load balance for the FOA at Piz Daint.

	48	96	192	384
Total load balance	92.60%	68.24%	52.41%	37.55%
↳ Inter-node	100.00%	96.50%	91.95%	94.12%
↳ Intra-node	92.60%	70.72%	56.99%	39.89%

Table 6. Load balance for the FOA at MN4.

7. Computing Performance

The observed computing performance at Piz Daint averages between 0.47 instructions per cycle (IPC) with 12 cores and 0.57 with 192 cores; at MN4 between 0.55 with 48 cores and 0.70 with 384 cores. In general, the application achieves a fair computing performance for the respective machines. However, this depends strongly on the underlying algorithmic structure of the code.
Figure 6 compares the computing performance at Piz Daint with 192 cores (top) and MN4 with 384 cores (bottom), whereas the scale of the colour gradient is the same for both timelines. Except the first phase that is executed only on the master threads, the computing performance is slightly higher at MN4.

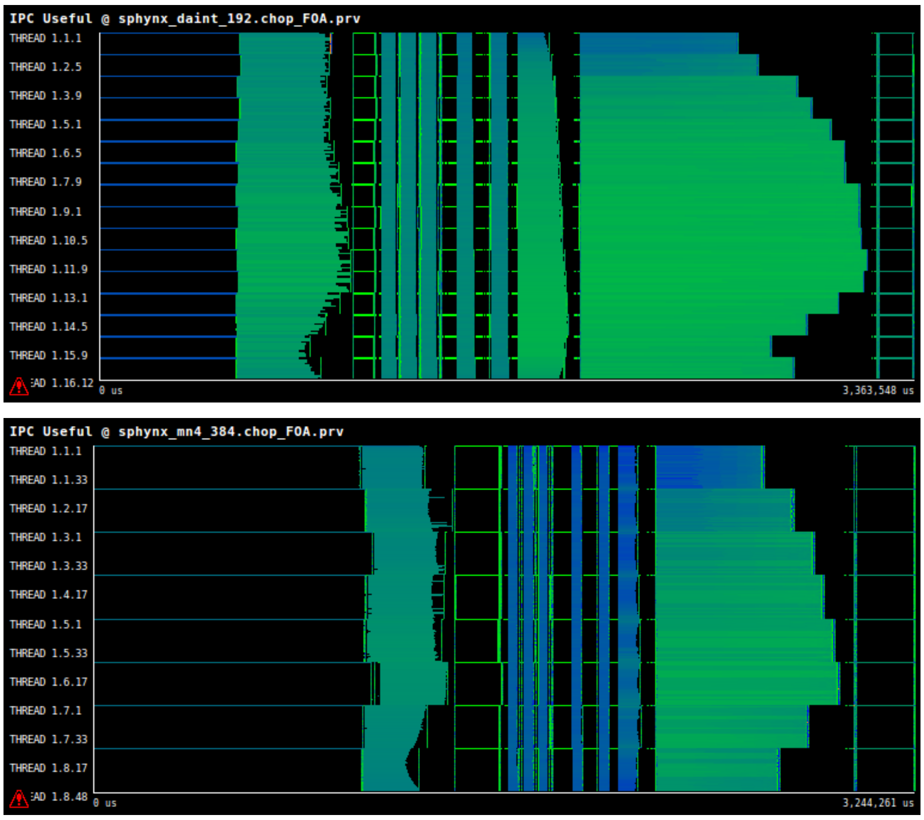


Figure 6. Comparison of computing performance for 192 cores at Piz Daint (top) and 384 cores at MN4 (bottom).

8. Communications

The MPI communication consists of two types of calls in the FOA: 34 calls to MPI_Allreduce (pink) and a final call to MPI_Barrier (red). The communication is only executed by the master threads, i.e. by one process per node. The code achieves in general a good communication efficiency (compare Tables 3 and 5). They includes only the time for the actual communication. In addition, since all these calls are globally synchronizing, they absorb the load imbalance of the previous compute phases, which increases the overall time spent in communication.

The communication efficiency slightly decreases with an increasing number of cores mainly because the total amount of data that is transfers does not scale ideally. Nonetheless, the overall communication efficiency is good, especially, considering that the typical use case of the application uses week scaling, i.e. the problem size scales with the number of cores.

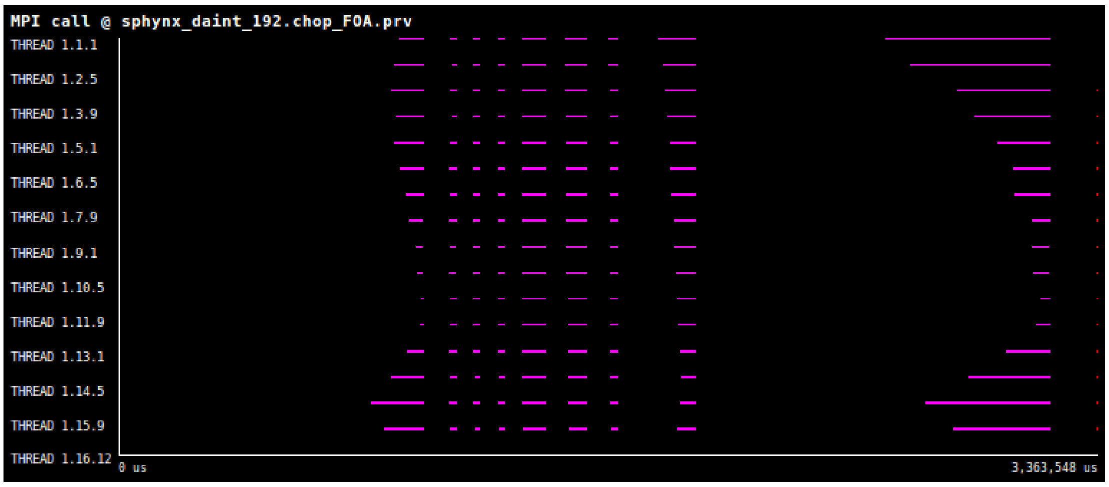


Figure 7. MPI communication in the FOA for 192 cores (Piz Daint).

9. Threading

As stated before, the main reason for the lowered parallel efficiency is the low and further decreasing load balance, which is primarily caused by the intra-node load balance due to the high idle times on the worker threads.

The main reason for the large idle times are phases that only involve the master thread on each node. This is due to the structure of the code that does not apply an OpenMP parallelization for all parts of the code. The amount of idle time due to lacking OpenMP parallelization is a primary target for optimization, allowing for a potential speedup of more than two at the given scale.

10. Accelerators

This section does not apply for this audit.

11. I/O

This section does not apply for this audit.

12. Summary and Suggestions

In this performance plan we analysed the performance of SPHYNX version 1.4 with a focus on scalability. We analysed traces based on 12, 24, 48, 96, and 192 cores at Piz Daint and 48, 96, 192, and 384 cores at MN4. Overall, the application achieves a good scalability in the given range at Piz Daint but a rather low scalability at MN4. The application realizes almost perfect scalability in the computation, i.e. the workload is almost perfectly distributed with increasing core counts. In addition, the application shows a good communication efficiency (MPI parallelization).

We found the main reasons for the limited scalability the decreasing load balance due to increasing idle times on the worker threads (OpenMP parallelization).

- The OpenMP parallelization is the primary target for optimization. Due to phases that only involve the master thread of each node, i.e. no OpenMP parallelization is used, the application wastes up to 30% and 60% of the total runtime on idling threads for Piz Daint and MN4, respectively.
- The main difference between the behaviour on Piz Daint and MN4 is the higher amount of idle time at MN4 since it involves 48 OpenMP threads per MPI rank in comparison to 12 threads per MPI rank at Piz Daint.
- With the current state of the development we recommend to reduce the ratio of OpenMP threads per MPI rank on MN4, e.g. use 4 MPI ranks with 12 OpenMP threads each per node.
- The computing performance at MN4 is slightly higher as at Piz Daint: 0.70 IPC vs. 0.57 IPC for the largest run respectively. The increased computing performance correlates strongly with the respectively smaller workload per thread, i.e. better cache utilization, at MN4.
- In comparison to the previous assessment of SPHYNX version 1.3.1, version 1.4 improves the speedup of 48 to 192 cores from 2.5 to 3.0 (ideally 4) mainly due to improving the load balance from 50.0% to 64.1%.