NEMO performance assessment report

Document Information

Reference Number	POP_AR_8
Author	Jesus Labarta (BSC)
Contributor(s)	Judit Gimenez (BSC)
Date	March 11th, 2016

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".

Table of Contents

1. Background

Applicants Name: Cyril Mazauric. BULL

Application Name: NEMO

Programming Language: ??

Programming model: MPI

Source Code Available: No

Input Data: Unknown.

Performance study: Overall analysis and demonstration of the functionalities supported by BSC tools. Identify areas of improvement.

The customer was interested in evaluating the analysis capabilities offered by the BSC tools. They installed Extrae on their platform and obtained traces of several runs of the MEMO application.

Tracing was done in burst mode with sampling for a range of processors between 24 and 480 and using the main trace control characteristics displayed in Table 1.

Trace-mode:	Burst (>1ms)
Sampling:	Yes
Hardware Counters:	PAPI_TOT_INS, PAPI_TOT_CYC, PAPI_L!_DCM, PAPI_L2_DCM, PAPI_L·_TCM, PAPI_BR_UCN, PAPI_BR_CN, PAPI_BR_MSP, PAPI_FP_INS, PAPI_LD_INS, PAPI_SR_INS, PAPI_VEC_SP, PAP_VEC_DP, RESOURCE_STALLS:SB, RESOURCE_STALLS:ROB, RESOURCE_STALLS:RS, RESOURCE_STALLS
Call Stack references	Yes (3 levels)
Ranks	24, 48, 72, 96, 240, 360, 480

Table 1: Main configuration to collect traces

The customer stated that NEMO is a memory bandwidth limited application with a poor scalability. They would like to have some hints about how to improve the scalability and how the memory pattern behaves.

In this report we focus on the analysis of the traces provided by the user, which do not allow us to study the memory pattern in detail. We consider that such more detailed analysis could be performed as part of a later Performance Plan study.

With the report, we provide some post-processed traces and specific configuration files that will allow the customer to navigate through the captured data and explore in further detail some of the observations we make.

2. Application structure

The spatio-temporal structure of the behavior of the run with for the whole run is shown in Figure 1 on a view of the length of useful computations for the 240 processes run.

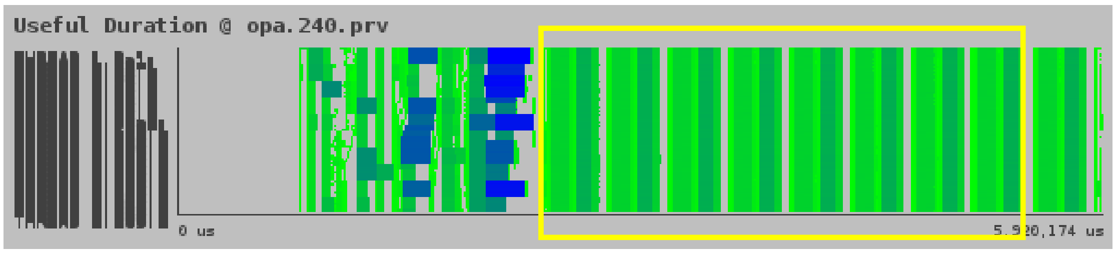


Figure 1: Useful duration view for whole trace at 240 processes

It shows an initialization phase followed by the iterative computations that constitute the region of interest (ROI) in the rectangle. The structure of the total execution for the different runs is similar across the different traces obtained.

The analysis has been done without accessing the source code. The call stack events happened to be mainly undefined. There seemed to be some issue with the instrumentation that has been fixed through the interaction between BSC and Bull. This analysis thus focuses on behavior and we leave reporting on syntactical structure to a possible Performance Plan report based on new traces.

3. ROI (Region of interest)

From the original traces we obtained cuts representing the ROI of 8 iteration of the main loop as enclosed in the rectangle in Figure 1. We show in Figure 2 the structure of three of those iterations for the case of 240 processes. The timeline on top presents the duration of the computational regions longer than 1ms.

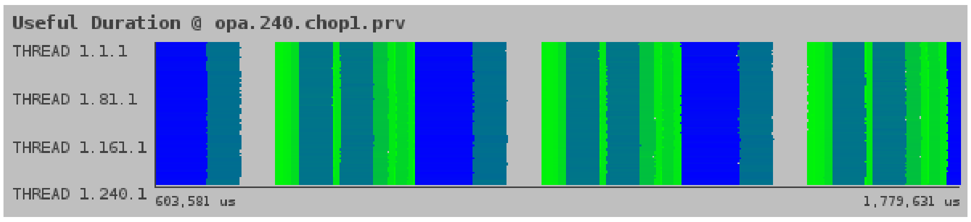


Figure 2: structure of ROI for 240 processes

4. Efficiency

Figure 3 displays the speedup computed from the execution time of the different ROIs. They show super linear speedup up to 240 processes that then stabilizes. This does not match the comment by the user when submitting the POP application stating that the application does not scale. We understand that the user statement is generic and applies to other settings (problem size, core count, …). In this draft version of the document we continue the analysis of the available traces trying to get insight on their behavior although the general context should be discussed with the user.

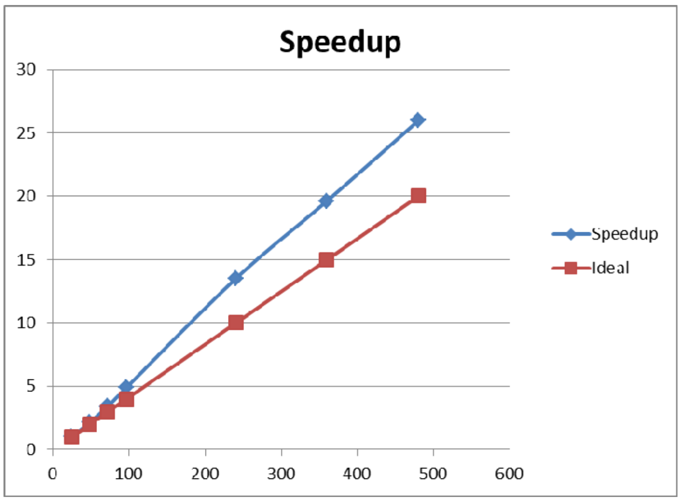


Figure 3: Speedup and efficiency derived from the ROI execution time

The analysis of the traces reports the parallel efficiency numbers in Table 2. Given that the traces were obtained in burst mode, it is not possible to differentiate between serialization and transfer effects. We thus report the aggregated communication efficiency. The table indicates that communication is the main cause of inefficiencies in the parallel execution and that the impact grows with scale, while load balance remains fairly stable.

The computational efficiency, derived from the total time in user mode at the different core counts shows a very important superlinear effect that compensates the small degradation in parallel efficiency and results in the still superlinear combined efficiency reported in Figure 3.

The instruction and IPC scaling efficiencies shown in Table 3 give more insight in the nature of the computation efficiency reported in Table 2. For IPC, this number represent a ratio of how the average IPC improves (>1) or shrinks (<1) with increased core count. For instructions it indicates whether the total number of user level instructions is the same (=1), higher (>1) or smaller (<1) than the number at the reference run (24 processes).

The IPC seems to improve significantly as core count increases and this contributes to the super linear computational efficiency. All traces were obtained with 24 processes per node, so there should not be an important difference between them in terms of memory bandwidth sharing impacting the variation in IPC. This improvement in IPC deserves further study in section 6

	24	48	72	96	240	360	480
Parallel Efficiency	0.9539	0.9501	0.9522	0.9561	0.8219	0.7816	0.7651
Load Balance	0.9928	0.9663	0.9821	0.9702	0.9812	0.9635	0.9776
Comm Efficiency	0.9608	0.9841	0.9696	0.9545	0.8377	0.8112	0.7827
Computation Efficiency	1.0	1.074	1.147	1.259	1.568	1.598	1.619
Global efficiency	1.00	1.07	1.14	1.22	1.35	1.31	1.30

Table 2: Parallel efficiencies observed in the ROI traces

	24	**48**	**72**	**96**	**240**	**360**	**480**
IPC scaling efficiency	1.000	1.0904	1.1941	1.3052	1.3202	1.3768	1.4266
Instruction scaling efficiency	1.000	0.9853	0.9601	0.9633	1.1885	1.1618	1.1358

Table 3: Other efficiencies computed for the ROI.

The total number of instructions in user mode stays stable till 96 processes but it is a bit smaller at the large core counts. We looked at this in detail and resulted to be in reality an artifact of the burst mode instrumentation used. Keeping the same threshold for all core counts implies that at large core counts some regions fall below the threshold to emit the useful duration and associated hardware counter events. The result is that some small computation regions are not separately accounted for. Although the error introduced will typically not be large, it is important to be aware of this effect. The implication on this analysis is that we do not believe that there is a significant amount of code replication even if directly applying the instruction efficiency statistic to the burst mode traces seems to indicate some replication in Table 3.

5. Load Balance

Table 2 indicates that load balance is not an important issue at least for the explored ranges of processes. Nevertheless, a bit more of detailed is given in Table 4 on the possible contributors to the observed load imbalance. We see that the imbalance in terms of instructions is extremely low. The imbalance in terms of cycles is also very small although a bit higher, essentially matching the time load balance from Table 2.

	24	48	72	96	240	360	480
Instr. LB	0.9995	0.9992	0.9992	0.9974	0.9991	0.9971	0.9914
Cycles LB	0.9934	0.9992	0.9840	0.9712	0.9811	0.9720	0.9777

Table 4: Load balance metrics

6. Serial performance

In this section we report analyses of the performance of the sequential user level code between calls to the runtime API.

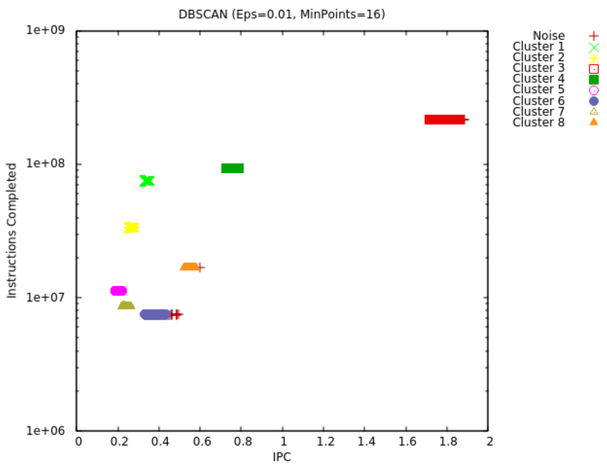


Figure 4: Clustering scatterplot for ROI @ 240 processes

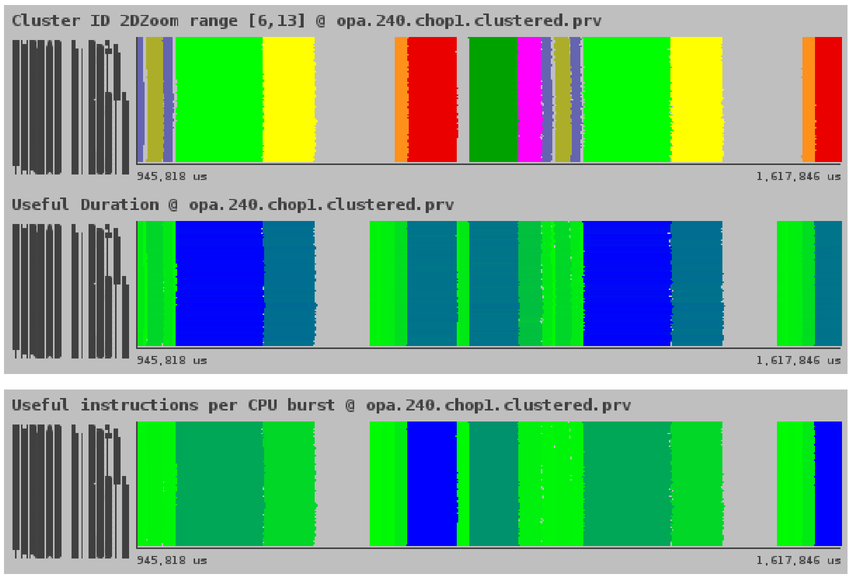

Cluster ID 2DZoom range [6,13] @ opa.240.chop1.clustered.prv
945,818 us
1,617,846 us
Useful Duration @ opa.240.chop1.clustered.prv
945,818 us
1,617,846 us
Useful instructions per CPU burst @ opa.240.chop1.clustered.prv
945,818 us
1,617,846 us

Figure 5: Cluster timeline for two iterations (top). The lower two timelines show respectively the useful duration and instructions in the corresponding regions of sufficient granularity

Figure 4 shows the structure of the ROI in terms of computational cost and sequential performance (#instructions vs. IPC axes). We also show in Figure 5 the spatio-temporal distribution of the detected clusters using the same coloring scheme (zoomed to a bit over a couple of iterations). For additional information we display in the same scale the useful duration and instruction count timelines.

The only cluster with an IPC above 1 is cluster 3. Cluster 4 is close to 1, but all other clusters have pretty poor IPC. Cluster numbering correlates to their total elapsed time so we can see that the most time consuming regions (cluster 1, light green; cluster 2, yellow) is not the ones that have more instructions, but their IPC is very poor.

When jointly analyzing all core counts we see the evolution of the centroids of each cluster (Figure 6) we see the reduction in instruction count that should be expected from a strong scaling experiment. A trend for all clusters to improve their IPC when scaling is also apparent. This justifies the IPC efficiency reported in Table 3.

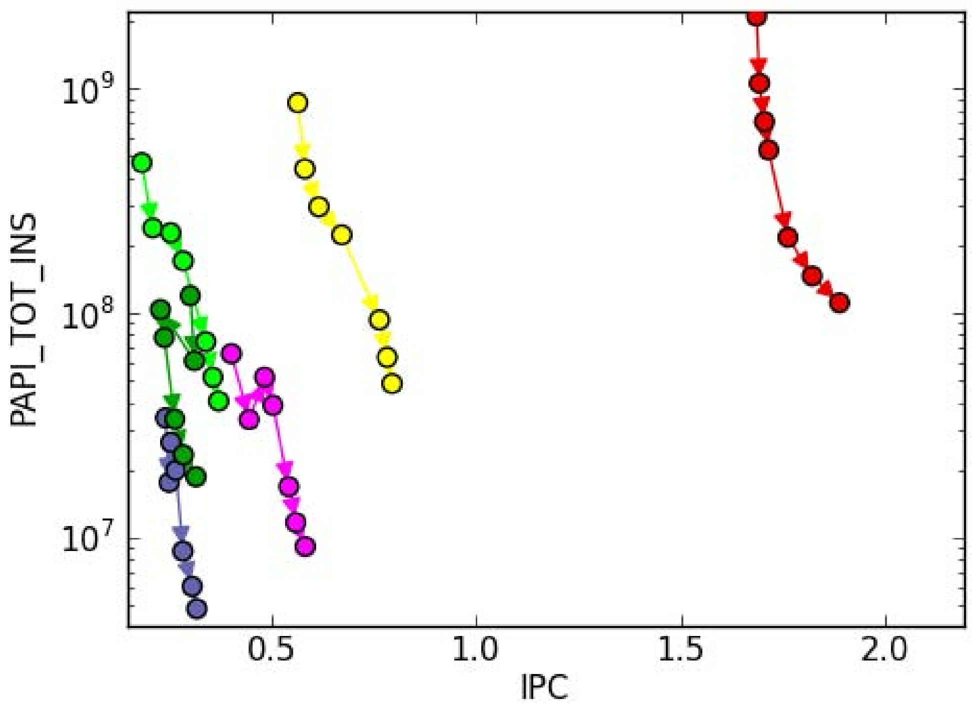


Figure 6. Evolution of #instructions vs. IPC for the main clusters when scaling from 24 to 480 processes.

To better understand the poor IPCs as well as its evolution along time we can use the sampled information available in the trace. The instantaneous evolution of MIPS along time for a full iteration of the algorithm is shown in Figure 7. We correlate it with other views showing the internal structure at a coarser granularity view. The MIPS scale goes from 1 to 6850. We see an overall fairly low ratio that corresponds to the low IPC shown for the individual clusters in Figure 6. Cluster 3 is the only one reporting a high MIPS value. The next area with higher MIPS at the end of the region is an area of intense communication. It is often the case in MPI intensive regions that a fair IPC (or MIPS) result from the busy wait loops waiting for message arrivals so the high value in this region does not necessarily represents fast progress in the computation. As we can see, the clusters with average IPC below 0.5 in Figure 6 have in reality a non-constant IPC along time, with some areas of very poor performance.

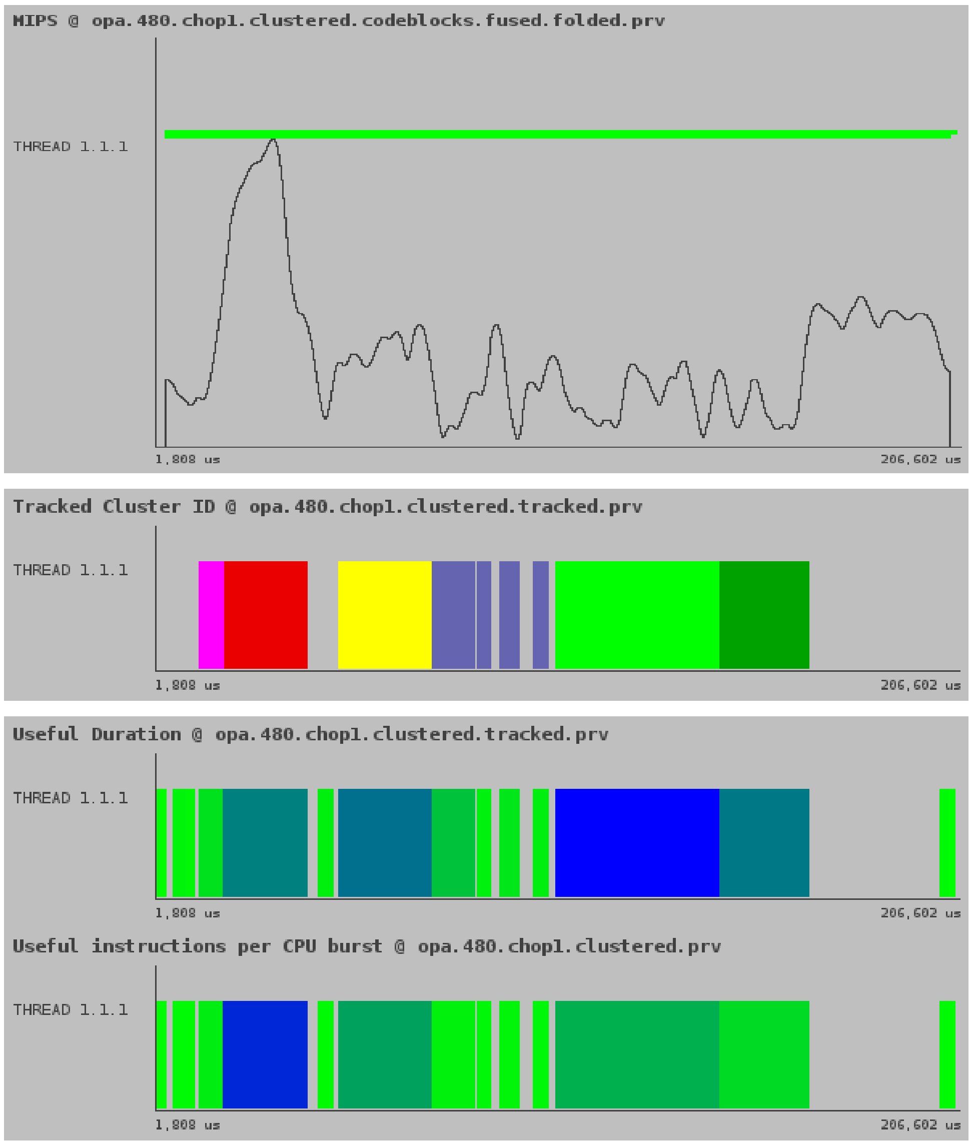


Figure 7: Instantaneous MIPS evolution for one full iteration (Top view). Additional synchronized views of cluster id, duration or total instructions within the different regions

We can go into further detail for the regions (clusters) of Figure 7. Let us focus on cluster 1 (bright green) as it represents an important fraction of the computation in each process. Its detailed MIPS evolution is presented at the top of Figure 8 for the 24 processes run (left) and for the 480 processes run (right). The timelines are not in at the same scale as the region is is significantly faster for 480 processes than for 24 processes. The vertical scale is the same and goes from 1 to 2075. Four phases of different behaviour can be identified in it, with some transitions between them. Even if four phases appear at both core counts, it is apparent that their behaviour is quite different. While phase 1 has a similar MIPS in both cases, phase 2 and 3 get s significantly better MIPS in the 480 run.

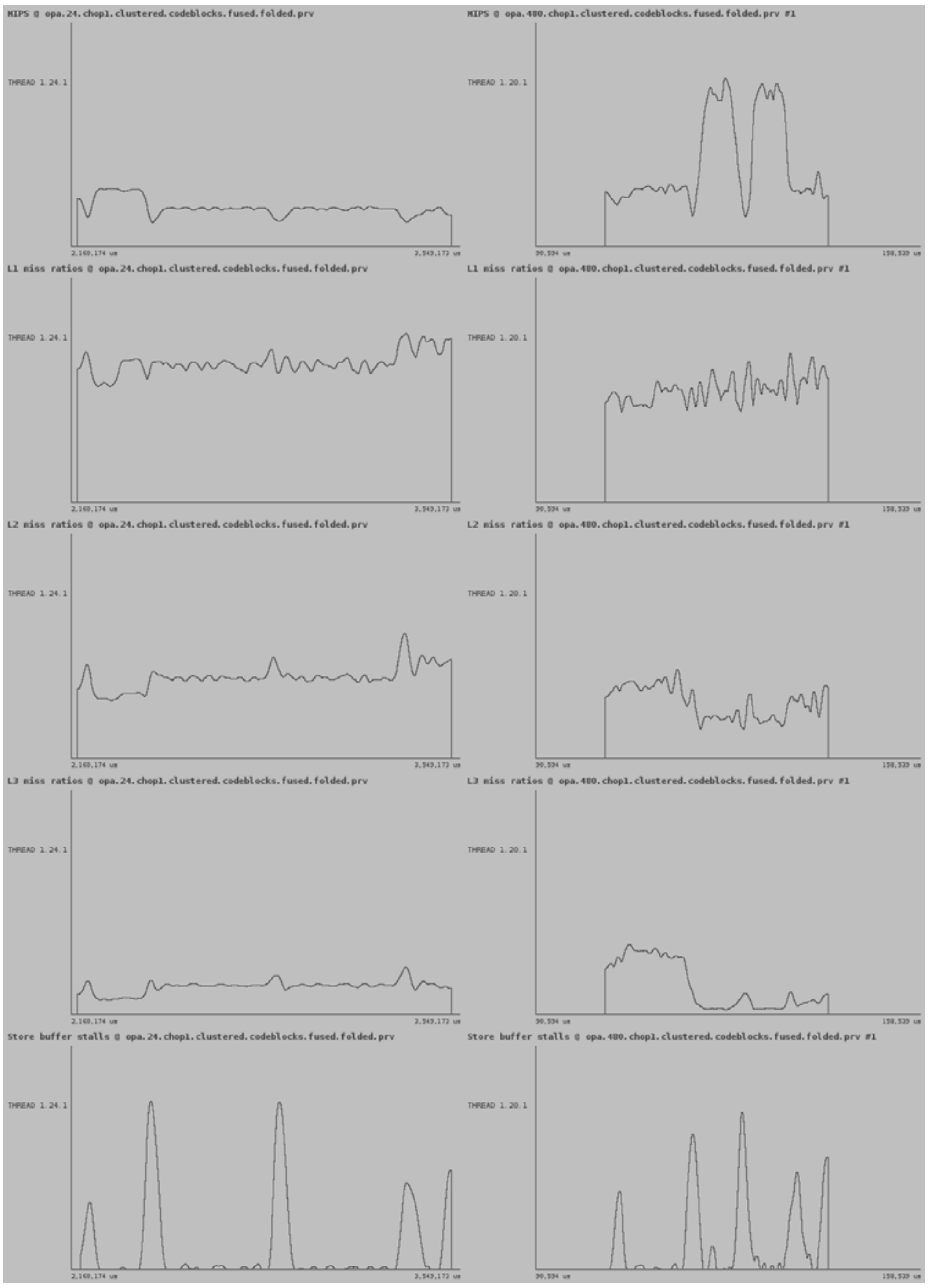


Figure 8: Instantaneous metrics for cluster 1 for 24 (left) and 480 (right) processes. From top to bottom: MIPS, L1, L2, L3 and store stalls.

Below the MIPS, the figure shows the miss ratios (misses per 1000 instructions) for

L1, L2 and L3 at the same Y-scale. L2 and L3 expose the difference in behaviour between phases identified in the MIPS timeline. For 24 processes we see relatively uniform miss ratios between all phases (the first one having a bit less L2 and L3 misses). About half of the L1 misses result in L2 misses and about one third of them result in L3 misses and this justifies the very low MIPS in all phases. For 480 processes the initial phase has a very high L3 miss ratio (actually most of the L1 cache misses also result in L2 and L3 misses) but the other phases have significantly less L3 misses. This explains the MIPS improvement for these phases with respect to the 24 processes case. Overall, this also explains the improvement in IPC observed in Figure 6 when scaling the number of processes under the strong scaling mode.

The last timeline in Figure 8 shows the stalls caused by the store buffer. They seem to be highly correlated with the very low MIPS in the transition between phases.

We have not seen the source code, but form the behavioural structure observed we can advance some suggestions of directions for improvement to further explore :

- The low MIPS and high store stalls between phases could correspond to initializations of arrays or vectors done at the beginning of the phase. If that was the case, one possibility to consider would be to use tasks (OpenMP, OmpSs) to overlap these operations with the computation that precedes it. In a heterogeneous platform, a little core could perform such bookkeeping activities, preparing the path for big cores to proceed at full speed.
- The program is memory bandwidth limited for small core counts, but the tracking study shows that it is possible to fit the data in L3 cache if sufficient number of cores under strong scaling is used. In a co-design context one could think of increasing L3 cache to enter before the superlinear zone, specially for some phases of the code. Overdecomposition and proper scheduling of blocks of successive phases migh be another alternative to investigate.
- The execution of the program is very synchronous, all processes entering the same behavioural phases at about the same time. This generates a lot of contention on memory bandwidth in regions like phase 1 in cluster 1 with 480 processes while other regions may have less memory bandwidth demand. Again over-decomposition and scheduling schemes that interleave blocks eager for a given resource with other less eager of such resource might be an interesting approach.

Even if highly speculative at this point in the analysis, this might be a potential a target for later Performance Plan POP studies. Analysing the memory access pattern during the whole cluster would be a potential activity addressed to better understand such behaviour and provide more informed vision on the potential usefulness of some of the proposed directions to explore.

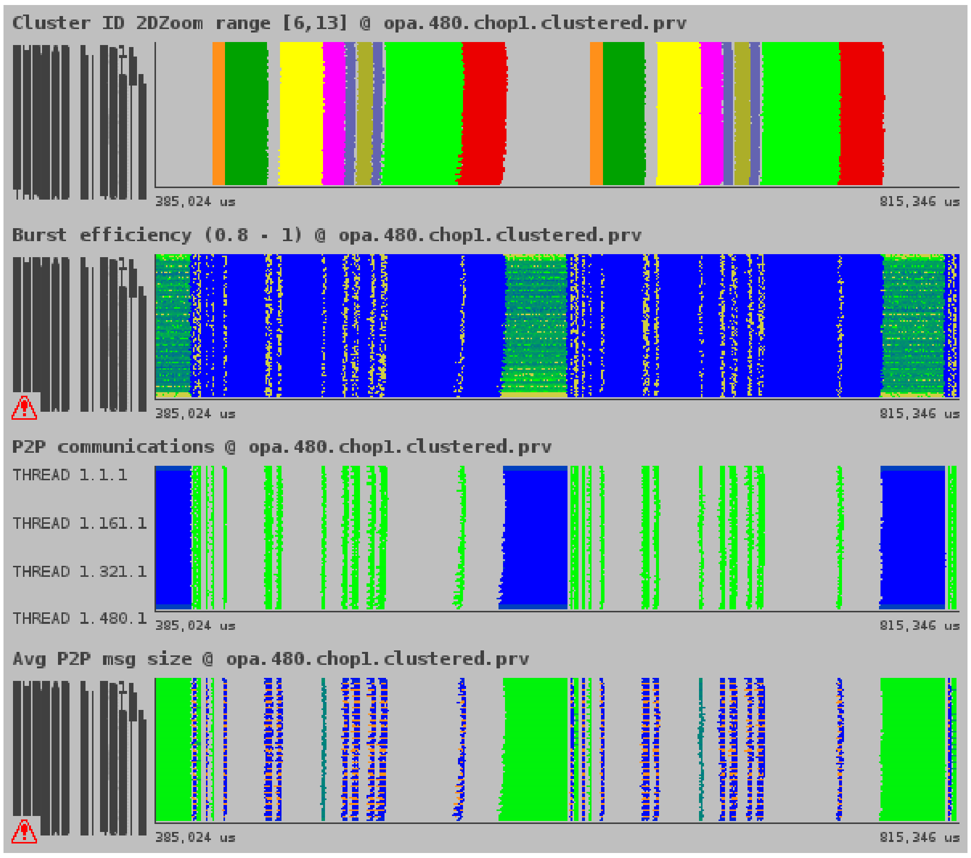
The call-stack samples were not properly translated in the trace generation process most probably due to the binary not having been built with the –g option. Most of them show up as “undefined” and we thus cannot relate the observed behaviour to the syntactic structure. Obtaining an additional trace with a binary compiled with –g would provide the information required to further investigate code refactoring options.

We also observed that the trace lacked some other stall counters that would help give an even more comprehensive view of the behaviour. The availability of such counters often depends on the processor and PAPI support in the specific installation where the traces were obtained. It might be interesting to identify what the actual issue with counters like ROB or SB stalls was for them to appear as cero counts in the trace.

Another observation on the raw trace is that it seems to expose a certain level of correlation of samples across processes in a node. Even if we have been able to obtain information for the analysis, it would be interesting to reduce such correlation to improve the exploration of the sampled space. This is interesting feedback for the tool developers.

7. Communications

The level of detail supported by burst mode traces is lower than that of fully detailed traces. We can nevertheless analyse some statistics for the regions containing communication calls intermixed with short computations.


Cluster ID 2DZoom range [6,13] @ opa.480.chop1.clustered.prv
385,024 us
815,346 us
Burst efficiency (0.8 - 1) @ opa.480.chop1.clustered.prv
385,024 us
815,346 us
P2P communications @ opa.480.chop1.clustered.prv
THREAD 1.1.1
THREAD 1.161.1
THREAD 1.321.1
THREAD 1.480.1
385,024 us
815,346 us
Avg P2P msg size @ opa.480.chop1.clustered.prv
385,024 us
815,346 us

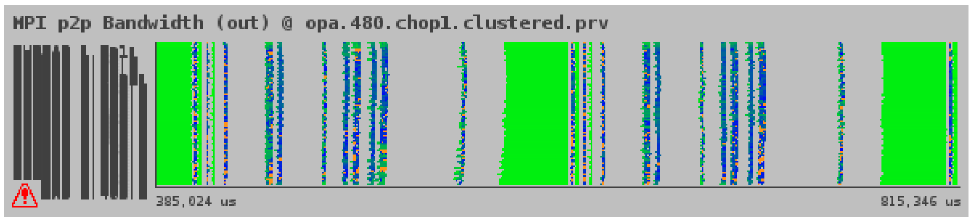


Figure 9: Communication metrics

We show different metrics in Figure 9 starting from top to bottom with the identified clusters id to act as a reference. The next view shows the percentage of time in user mode code in each interval. For long computation regions, the efficiency is 1 (dark blue). For regions with fine grain sequences of communication and computation, the colour shows the average percentage of time the process was in user level code, the rest being in MPI. The wide light green region in the middle of the timeline represents a region with an average efficiency of 80%. The next timeline shows the number of MPI point to point calls in each interval. The value for the just identified region is 960, and the average granularity between these communications is 64 microseconds. All other regions with MPI calls have a much small number of MPI calls (<25) between two long computation bursts. The next view shows the average number of bytes and the last timeline the effective bandwidth in communication intervals.

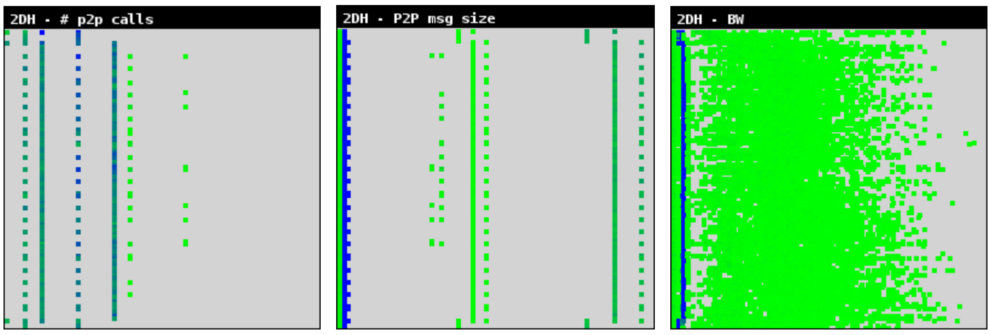


Figure 10: Histograms of number of point to point calls in a communication interval, average message size (focusing average on messages less than 32KB) and effective bandwidth achieved (horizontal scale from 0 to 500 MB/s)

For a more quantitative analyst of the distribution of these metrics we include in Figure 10 histograms of the number of point to point calls, message size and effective bandwidth. The histograms on number of point to point calls and message sizes show that these values are not perfectly balanced with differences between processes that show some structure, possibly derived from the partitioning of a regular domain. The effective bandwidth histogram reflects much higher variability, and values distributed in a range between0 and 500 MB/s. The dominant mode at a low value of 20MB/s corresponds to the long region of frequent comunications mentioned before. Given that 80% of the time in that region was computation and 20% MPI, we can infer that the actual bandwidth achieved during those computations was in the order of 100MB/s. This value may not be very large and might be interesting to look at the detailed communication behaviour in that phase. This would require fully detailed traces.

An aggregated plot of the average amount of bytes sent by each process is shown in Figure 11 as a function of the number of processes. The model (green line) of the communication surface between domains considering squared patches in a 2D physical system model matches fairly well the measured values. As the computation of for the subdomain patch depends on its area, the volume to surface ratio gets worse with scale and this justifies the communication efficiency numbers in Table 2 getting worse with increased core count.

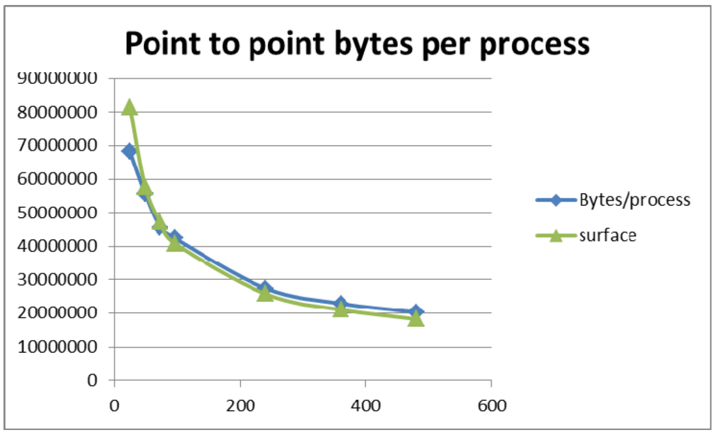


Figure 11: Average bytes sent per process vs number of processes

A final comment relates to the variability of total bites sent by the different processes for a given core. This was shown to happen in Figure 10. Such variability is higher for small core counts. The corresponding load balance metric varies from 0.87 for 24 cores to 0.97 for 480. The effect can be attributed to the decomposition, as the larger the process count, more of them are internal and this have neighbors in all directions, while for few processes many of them do not need to communicate in some of the directions.

8. Summary observations/recommendations

In this assessment we have analyzed burst traces with sampling information of runs between 24 and 480 processes of the NEMO code. We focused on squeezing the information obtainable from the raw data in the traces provided by the customer. This allows us to show the type of analyses supported by BSC tools, but also to identify aspects of the performance of the application that could be amenable to improvements.

Some general observations from the analysis are:

- Even if the traces do not have a complete information (missing values for some hardware counters, executable not built with –g option so no symbolic information is available,…) it has been possible to demonstrate several analyses supported on this type of traces. In particular overall parallel efficiency models, hardware counters metrics, tracking performance of sequential computation parts as core count increases or reporting the instantaneous evolution of hardware counter derived metrics.
- Many of the relevant regions of the code (clusters) have a very poor IPC. In reality, some of them seem to have internal phases with very poor values alternating with phases a bit better. The fact that these phases seem to put different pressure on the architectural resources (eg. Memory bandwidth) suggest that code restructurings (scheduling/synchronizations) that could shift the phases of different processes in time might help achieve a better utilization of the architecture, in particular the memory subsystem.
- We have identified some methodological issues to improve the precision of certain metrics when scaling burst mode traces. This feedback will have to be considered as recommendations on how to instrument future experiments or to develop some extensions to the Extrae library.

We have not performed the request form the customer to analyze the memory access pattern as this information was not available in the traces. We suggest performing such activity as part of a Performance Plan study where new traces should be obtained by the customer. Studies for counters not available on these traces, or memory access pattern could be done as part of a performance plan study. In such study we should also use look at call stack references and some of the hardware counters that were not properly read in these traces.