CGPACK-PARAFEM performance assessment report

Document Information

Reference Number	POP_AR_82, Version 1.0
Author	José Gracia (HLRS)
Contributor(s)	
Date	December 6, 2017
Application	CGPACK-PARAFEM
Service Level	Performance Audit
Keywords	CGPACK, PARAFEM, MPI, Coarray Fortran

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 n°676553.*

Contents

1 Background

Application Name: CGPACK-PARAFEM
Applicants Name: Anton Shterenlikht
Applicants Affiliation: University of Bristol
Programming Language: Fortran
Programming Model: hybrid MPI + Coarray Fortran
Source Code Available: Yes
Input data:
Performance study: Audit

The customer stated, he would like to have a general in-depth audit to reveal the potential scalability issues of their application. Also, large performance variabilities between systems and software environments have been observed. The application consists of a library CGPACK, which is developed by the customer. CGPACK is used by the actual application PARAFEM, which is a well-known proxy-app in the respective community. While PARAFEM does some MPI communication of its own, CGPACK uses exclusively Coarray Fortran for all its communication. The customer is hoping to scale this hybrid application, in particular CGPACK, up to many thousands of cores.

At the time of the audit, the customer could provide compute resources only at Archer@EPCC. Therefore, performance variations across systems will not be targeted in this Audit. Archer nodes consist of two Ivy Bridge processors, each with 12 cores. The software environment consists of PrgEnv-cray/5.2.8, perftools/6.4.6, extrae/3.4.1, and papi/5.5.1.1.

Tracing of Coarray Fortran communication is not natively supported by Extrae. On some systems, Coarray is implemented on top of MPI, which would be visible to Extrae. On Archer, however, Coarray is implemented on top of DMAPP which is not supported by Extrae either. Thus only the MPI communication in the PARAFEM part is accessible to Extrae. Cray's profiler Perftools supports Coarray fully, but does not provide traces, which can be analysed with POP tools. Therefore, data was collected with Extrae for MPI traces and with Perftools for Coarray profiles in separate runs. Due to the size of trace files, the analysis was done only up to 64 nodes of Archer corresponding to 1536 cores; runs with 192 cores are used as reference.

2 Application Structure

The spatio-temporal structure of the application from an Extrae perspective (MPI communication is visible, but Coarray is not) is illustrated in Fig. 1. On 192 MPI ranks, the application runs for roughly 900 s. The execution is divided into 5 phases (bluesish colour) of roughly equal duration of several minutes. No MPI communication is taking place during these phases, however Coarray operations may happen. These phases are separated by short phases characterised by large amounts of packed collective MPI operations. The communication phases last only about 1 s to 4 s, where the duration is increasing from one to the other. Overall, the time spent in MPI communication is around 1 %, and thus almost negligible at this core count. Nonetheless, the spatiotemporal structure of the next-to last MPI communication phase is shown in Fig. 2: MPI barriers, allreduce and non-MPI burst are tightly interlaced – no particular communication pattern is easily discernible.

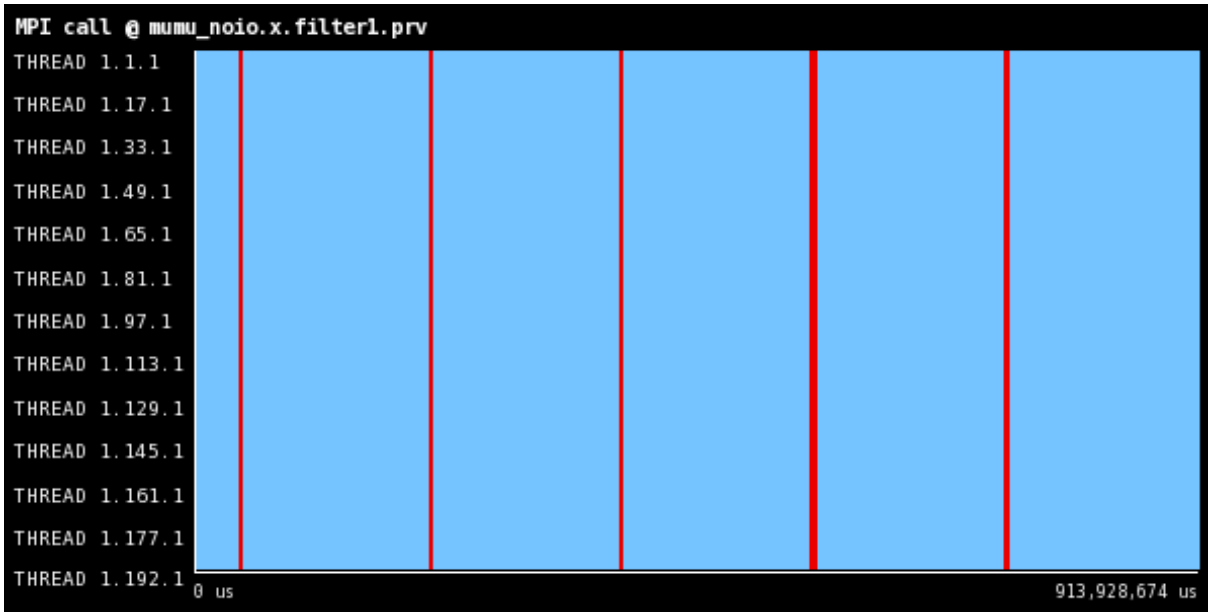

Figure 1: Timeline of the execution of the application as seen by Extrae. Each horizontal row shows the colour-coded timeline for a particular MPI rank. The 192 MPI ranks are stacked vertically with the MPI root on top. Redish colours indicate time spent in MPI operations, while bluish colours correspond to time spent in useful user code or Coarray operations (the latter are invisible to Extrae and thus perceived as useful user code). Both, time in MPI and Coarray operations will later be considered as non-useful parallel runtime overhead.

3 Focus of Analysis

The bulk of the analysis presented in this report will focus on the application as a whole. This is due to the fact that Cray's Perftools can, in practise, only provide aggregated profile data, while Extrae can provide time-resolved traces, but is unaware of Coarray operations. Some analysis done with Extrae will focus on the MPI communication phases, in particular the next-to-last one shown in Figures 1 and 2.

4 Scalability

Fig. 3 shows data of a strong scaling experiment ranging from 192 to 1536 cores. The figure shows total application execution time, as well as the breakdown of time spent in useful user code, Coarray operations, and MPI operations, respectively. Note, that the scaling curve departs significantly from an ideal one. In particular, at 1536 cores the speedup relative to 192 cores is only at 39 % of the expected. Any increase in the number of cores beyond will likely result in increasing wall-clock time. The user code as such scales well, but Coarray operations do not. In fact, at 1536 cores they take longer than user code. Also, the time spent in MPI increases with number of cores as expected from collective operations. However, for this particular setup MPI is still not relevant.

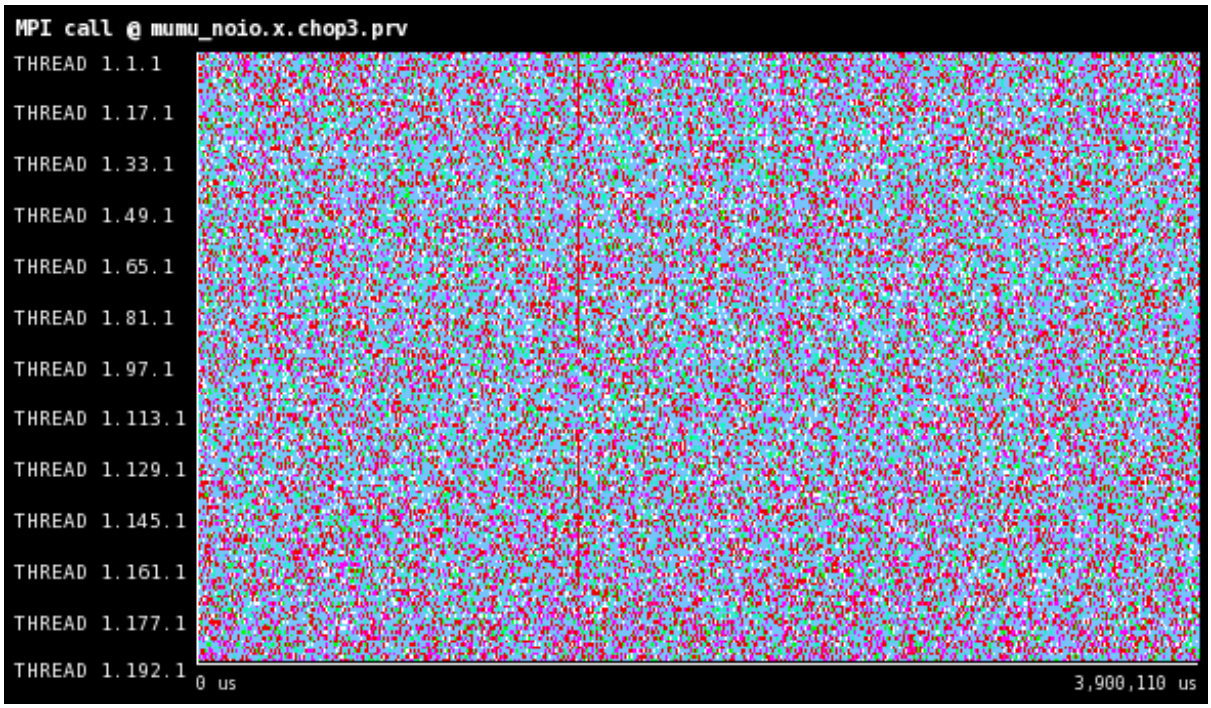

Figure 2: Timeline of a MPI communication phase as seen by Extrae. The duration of this phase is only 3.9 s. It is characterised by some amount of useful user code (shown in light blue) interlaced with large amounts of collective MPI operations, in particular MPI_Barrier (red) and MPI_Allreduce (pink).

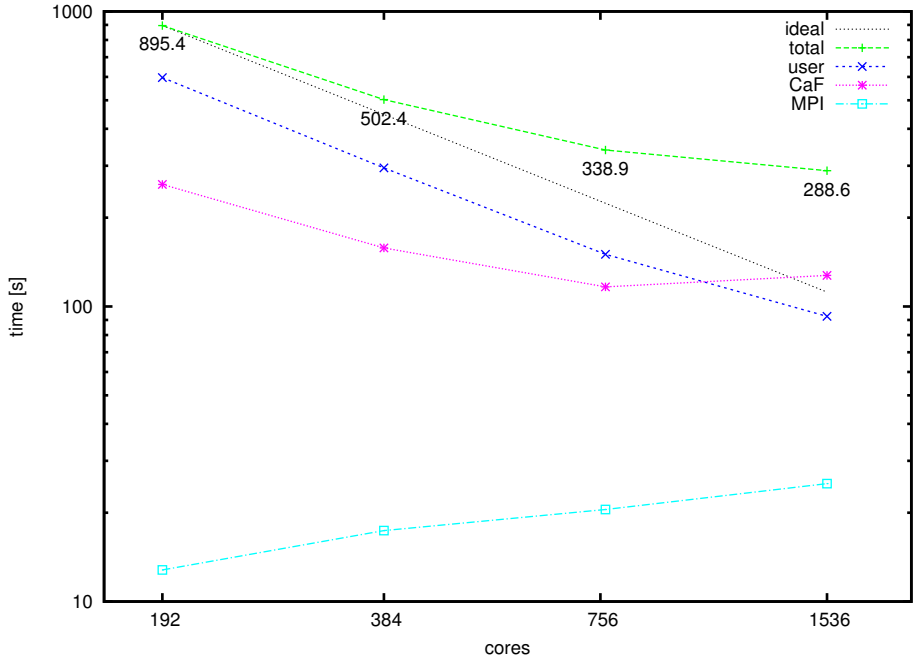

Figure 3: Strong scaling experiment. The data is plotted as reported by Perftool. The various curves show ideal scaling (dotted), total application time (green), useful user code (dark blue), Coarray Fortran operations (purple), and MPI (light blue). Note, that each data point represents a single experiment, only.

MPI ranks	192	384	768	1536
Execution time	895.4	502.4	338.9	288.6
Useful duration (average)	596.6	294.8	150.3	92.6
Useful duration (maximum)	798.4	411.8	224.9	153.7
Useful duration (sum)	$1.1 \cdot 10^5$	$1.1 \cdot 10^5$	$1.2 \cdot 10^5$	$1.4 \cdot 10^5$

Table 1: Breakdown of application time (in s) spent in the execution state *useful* user code and its total runtime for runs with increasing number of MPI ranks, respectively. The average, maximum, and sum are understood to be taken across the set of all MPI ranks, while the execution time is assumed equal for all.

Number of processes	192	384	768	1536
Global efficiency	66.6	59.4	44.0	25.8
Parallel efficiency	66.6	58.7	44.3	32.1
Load balance	74.7	71.6	66.8	60.2
Communication efficiency	89.2	82.0	66.4	53.3
Serialization efficiency				
Transfer efficiency				
Computation scalability	100.0	101.2	99.2	80.5
Instruction scalability				
IPC scalability				
Average IPC	3.1			

Table 2: Compilation of performance efficiencies as discussed throughout this report for various MPI rank counts. The values of efficiency and scalabilities are given in percent. The indentation level of the rows indicates their hierarchical composition; for instance, parallel efficiency is composed of load balance and communication efficiency.

5 Efficiency

In this section we discuss certain performance metrics which have been obtained from the instrumented benchmark runs. The results are presented in terms of a number of *efficiencies* and *scalabilities*, which are related to fundamental performance issues. As usual efficiencies and scalabilities can take values between 0% and 100%. These metrics are all derived from the time the application spends in different execution states. For an MPI code, these states are *useful* and *MPI runtime*. In the former case, the cores are executing user code, i.e doing useful work, while in the latter case the application is executing code which is related to the parallel runtime system, as for instance doing communication or synchronising processes, and thus not directly contributing to the application's business logic. In this particular case, we have two parallel runtime systems, MPI and Coarray; time spent in either one is considered non-useful parallel runtime overhead.

Table 1 gives details on the time the application is spending in these states within the focus of analysis across different number of MPI ranks. From these basic measurements we can derive a set of performance metrics as presented in Table 2 and discussed in the following. The terms average (avg), sum and maximum (max) are understood to be taken across the set of MPI ranks, respectively. Further details on performance metrics used in the POP project can be found on the project website, in particular [1, 2, 3]. In general, efficiencies are considered acceptable only if their value is above a threshold of 80 %.

Parallel Efficiency (PE) The parallel efficiency is defined as the ratio of the average useful time over the total execution time.

$$\textbf{Parallel Efficiency} = \frac{avg(\text{useful time})}{max(\text{execution time})} \tag{1}$$

The higher the parallel efficiency is, the more time the application spends doing user code, rather than executing MPI/Coarray functions or paying parallelisation overheads in the runtime system. The parallel efficiency is found to be around 67 % for the reference run using 192 cores. This value is already considered unacceptably low – roughly 33 % of the application time is lost doing communication or synchronisation. With increasing number of cores the parallel efficiency continues to drop significantly. Note, that the parallel efficiency can be expressed as the product of load balance and communication efficiency, i.e. PE = LB × CE, both of which are introduced below.

Load Balance Efficiency (LB) The load balance efficiency is defined as the ratio of the average useful time over the largest useful time found across ranks.

$$\textbf{Load Balance} = \frac{avg(\text{useful time})}{max(\text{useful time})} \tag{2}$$

The higher the load balance efficiency is, the lower is the variation of the time spent doing useful work among different MPI ranks relative to largest value of useful time across MPI ranks; work is distributed in a more balanced way. The observed load balance efficiency is around 75 % for the reference run and drops gradually for increasing number of cores. Again, these values fall below acceptable threshold. The application's distribution of work and communication cost is thus not well balanced across MPI ranks on average. Further discussion on load balance can be found in Section 6.

Communication Efficiency (CommE) The communication efficiency is defined as the ratio of the maximum useful time over the maximum execution time for any rank.

$$\textbf{Communication Efficiency} = \frac{max(\text{useful time})}{max(\text{execution time})} \tag{3}$$

The higher the communication efficiency is, the lower is the fraction of time spent in MPI/Coarray operations assuming absence of load imbalances. We find values of 89 % for the reference one, which is considered acceptable. However, with increasing number of cores communication efficiency falls significantly and is low for runs beyond 384 cores. Further discussion on communication can be found in Section 8.

Computation Scalability (CompS) The computation scalability is defined as the ratio of the sum of the useful time in the reference run over the sum of the useful time for a specific rank count.

$$\textbf{Computation Scalability} = \frac{sum(\text{useful time})_{\text{reference}}}{sum(\text{useful time})} \tag{4}$$

The higher the computation scalability is, the lower is the increase of computation time in user code with increasing number of MPI ranks. Essentially, this quantifies the efficiency of parallel decomposition of the problem. The computational scalability is in all cases close to 100 %, which is excellent. Thus the parallelisation algorithm does not introduce additional instructions in the user code with respect to increasing the number of MPI ranks. Only at the largest scale of 1536 cores, the computational scalability seems to drop to 80 %. Further discussion on computation performance can be found in Section 7.

Global Efficiency (GE) The global efficiency is defined as the product of the parallel efficiency and the computational scalability.

$$\textbf{Global Efficiency} = \text{Parallel Efficiency} \times \text{Computational Scalability} \tag{5}$$

The higher the global efficiency is, the higher is the overall scaling efficiency when increasing number of MPI ranks. This metric is similar to the scaling efficiency derived from weak or strong scaling experiments, but applied to the specific focus of analysis. The application shows unacceptably low global efficiency of at most 67 % at 192 cores. The low value can be blamed fully on the low parallel efficiency.

6 Load Balance

As stated above, the application suffers from load imbalance. In this particular case, imbalances do not seem to stem from imbalance computation, but rather from imbalance in the time spent on communication, in particular Coarray. Due to the sub-optimal analysis capabilities it is difficult to see what is really going on. Apparently, the bulk of imbalance arises due to Coarray operation `cosum` in the subroutine `cgca_clvg` of the Fortran module `cgca_m3clvg`. For instance, at 1536 cores time on this operation spans from 27 s to 129 s with an average of 73 s for 665 invocations. The next biggest imbalance arises in `pgas_sync_all` operations. In this case, it cannot be blamed on particular subroutines or modules.

Computational load imbalances arise in the function/module `xx14` with execution times spanning from 11 s to 153 s with an average of 93 second.

7 Computational Performance

In this section we report on the performance of useful computation regions, i.e. actual user code. All modern CPUs in principle allow to execute more than one instruction per clock cycle. In practice, peak CPU performance is never achieved. Using hardware counters, we have measured the instructions per cycle (IPC) within the useful regions of the application.

For the case of 192 cores, the average IPC is 3.05, which is an excellent value for this processor. The computational scalability (CompS) metric introduced in Section 5 shows, that the time doing useful computations does not change significantly with the number of MPI ranks. Therefore, the parallelisation does not introduce additional workload in terms of CPU time to solve the underlying algorithm. Therefore, it is reasonable to assume the IPC remains high for core counts beyond the reference run.

For completion, it is noted, that computational scalability can be expressed in terms of instructions scalability (measuring how numbers of instructions change) and IPC scalability (measuring how IPC changes).

8 Communications

The communication efficiency is acceptable for lowish number of of cores, but declines with increasing core counts. Proper analysis of this is difficult without access to trace data. However, already the scalability experiment shown in Fig. 3, suggest that Coarray synchronisation becomes dominant; there is little indication that the time spent in actual Coarray data transfers increases significantly. The phases of MPI activity, however, are characterised by large amounts

of collective operations such as barriers and allreduce. As expected for collective MPI operations, these do not scale well with increasing number of cores. Therefore it is expected, that MPI operations will eventually become a performance scalability issue.

For completion, it is noted that formally the communication efficiency (CommE) can be decomposed further. On one side, time in MPI/Coarray can be due to being engaged in actual message transfers which is captured by *transfer efficiency (TE)*. On the other hand, MPI ranks might be blocked in MPI/Coarray calls, such as for instance `MPI_Recv`, or waiting for communication partners to enter the corresponding MPI call, for instance `MPI_Send`. This kind of waiting time and other similar effects are captured by the *serialisation or synchronisation efficiency (SE)*. We can thus define communication efficiency as the product CommE = TE × SE. Note, that disentangling transfer and serialisation, requires to simulate the application's communication on an ideal network which is assumed to have negligible latency and infinite bandwidth. The useful time will be equal on real and simulated network. However, this kind of analysis is not possible in this Audit as time-resolved traces are not available

9 Summary of observations

In this report we have analysed the application CGPACK-PARAFEM, which is a hybrid application using MPI and Coarray Fortran operations alike. Analysing this application was challenging due to the fact, that this particular combination of programming models is not supported by standard tools when time-resolved traces are required. The report is based on performance data collected on Archer at EPCC for core counts between 192 and 1536.

The computational or single-core performance of the application is excellent as witnessed by the high average IPC value. Also, increasing the number of cores had no influence on the duration of computational phases up to 768 cores. Only at 1536, there is evidence suggesting that the computation work load increases as seen by the drop of computational scalability down to mere 80 %.

On the other side, the application spends significant time in parallel runtime systems doing communication and synchronisation. Already at the low end of core counts, i.e. at 192 cores, the application pays significant parallelisation overheads (roughly one third of the execution time). The situation only worsens with increasing number of cores. This low parallel efficiency stems mainly from insufficient load balance, both in user computations and the communication operations. With increasing core counts, however, the insufficient communication efficiency starts to dominate parallel efficiency. Most likely the cause is increasing synchronisation cost as opposed to actual data transfer cost. These observations above refer to Coarray operations as the code spends little time only in MPI on the scales considered. It is clear however, that the MPI communication time, in particular for collective operations, will become dominant for larger core counts.

In order of expected impact we recommend:

Reduce computational load imbalance The distribution of the actual computational workload is not well balanced. Addressing this is crucial as this kind of imbalance will have an adverse effect independently of the actual parallel programming model used. Further analysis might be required to assess whether the number of instructions is unbalanced, or otherwise if the efficiency of calculation is non-homogeneous (e.g. NUMA effects, etc).

Reduce impact of Coarray operations The time spent doing Coarray operations, in particular the collective `cosum`, is significant. The same is true for synchronisation overhead associated with Coarray operations. It might be worth evaluating alternatives to collective

operations, such as hybrid approaches with local aggregation through shared-memory, or asynchronous operations which would allow hiding most of communication latency behind computation.

Reduce impact of collective MPI operations While not critical at the scales under consideration, MPI collective will become a bottleneck at higher core counts. In particular, many barrier operations have been observed. In most cases, these can be replace with other less intrusive synchronisation mechanisms, if they are necessary at all.

Analyse drop of computational scalability at high core counts The computational scalability, while initially excellent, shows a marked drop at 1536 cores. First of all, it should be confirmed that it is not only a statistical artefact. In case it is real, overall performance will be impacted at higher core counts.

List of Figures

List of Tables

Acronyms and Abbreviations

- MPI: Message Passing Interface

References

References

[1] POP standard metrics for parallel performance analysis, https://pop-coe.eu/node/69, accessed: August 2017.

[2] Efficiency metrics in a POP performance audit, https://pop-coe.eu/sites/default/files/pop_files/metrics.pdf, accessed: August 2017.

[3] Paraver efficiencies guide, https://pop-coe.eu/sites/default/files/pop_files/paraverefficenciesguide.pdf, accessed: August 2017.