A Comparison of Parallel Profiling Tools for Programs utilizing the FFT

Brian Leu
brian.d.leu@gmail.com
Applied Dynamics International
Ann Arbor, Michigan, USA

Samar Aseeri
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
samar.aseeri@kaust.edu.sa

Benson K. Muite
Kichakato Kizito
Nairobi, Kenya
benson_muite@emailplus.org

ABSTRACT

Performance monitoring is an important component of code optimization. Performance monitoring is also important for the beginning user, but can be difficult to configure appropriately. The overhead of the performance monitoring tools Craypat, FPMP, mpiP, Scalasca and TAU, are measured using default configurations likely to be choosen by a novice user and shown to be small when profiling Fast Fourier Transform based solvers for the Klein Gordon equation based on 2decomp&FFT and on FFTE. Performance measurements help explain that despite FFTE having a more efficient parallel algorithm, it is not always faster than 2decom&FFT because the complied single core FFT is not as fast as that in FFTW which is used in 2decomp&FFT.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**;
• **Networks** → *Network measurement*.

KEYWORDS

Performance, Profiling Tools, Fast Fourier Transform

ACM Reference Format:
Brian Leu, Samar Aseeri, and Benson K. Muite. 2021. A Comparison of Parallel Profiling Tools for Programs utilizing the FFT. In *IXPUG Workshop HPC Asia '21: International Conference on High Performance Computing in Asia-Pacific Region, January 20–22, 2021, Jeju, South Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Performance analysis of parallel programs can be greatly aided by using profiling tools. It can take time to learn how to use all available profiling tools, thus it is helpful to know what the different profiling tools can do. When first learning how to use distributed memory parallel computing systems, profiling tools can give the programmer insight into how well their program works, and the bottlenecks preventing good performance. The main motivation for the study is that IPM is a lightweight, portable and easy to use profiling tool that can be introduced at the early stages of teaching distributed memory parallel programming. It produces a nice text summary, with an optional graphical webpage. It can also be run in production settings, for example to supplement supercomputer center job profiling information[35]. However, it primarily profiles point to point communication and does not profile MPI_ALLTOALL which is the dominant communication call in many Fast Fourier Transform programs. The aim is therefore to find a suitable easy to use portable replacement for IPM that captures information about MPI_ALLTOALL. It is also of interest to compare portable open software to vendor provided tools. On systems with Dragonfly networks that have adaptive routing and opportunistic job placement, job interference can result in dramatic changes to program execution time - it is unclear what the signature of such interference will be when profiling with the different tools. A final aim is to see if profiling can help provide an explanation for the differences in scaling behavior of FFTE as compared to 2decomp&FFT.

The tools, IPM, Scalasca, CrayPat, mpiP, FPMPI and TAU are compared when profiling a Fast Fourier Transform solvers for the nonlinear Klein Gordon equation. The main findings are:

- Some of these tools can be challenging to setup if not already setup for the user
- For the number of cores considered here, these tools can be used in a production setting for low overhead profiling
- Vendor provided tools can allow for well integrated hardware performance monitoring that may not be otherwise easily exposed or integrated in the portable open source tools

2 TEST PROGRAM

Previous work has compared the performance of a Fast Fourier Transform (FFT) based solver for the three dimensional Klein-Gordon equation[3] using 2decomp&FFT. That study primarily focused on measuring time to solution and strong scaling results. The study also produced a simple performance model. More detailed performance models can be validated using performance measurements. In this study the numerical solution of the Klein-Gordon equation is again used as an example mini-application for which performance profiling tools can also be compared. In addition, a solver using the FFT library FFTE is included in the comparison. Earlier measurements shown in Fig. 1 showed that TAU[7, 30, 31, 42], a performance profiling tool can be configured to have low overhead even when profiling at large core counts. One of the aims of this work is to quantify the level of overhead and compare TAU to other profiling tools.

IXPUG Workshop HPC Asia '21, January 20–22, 2021, Jeju, South Korea

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

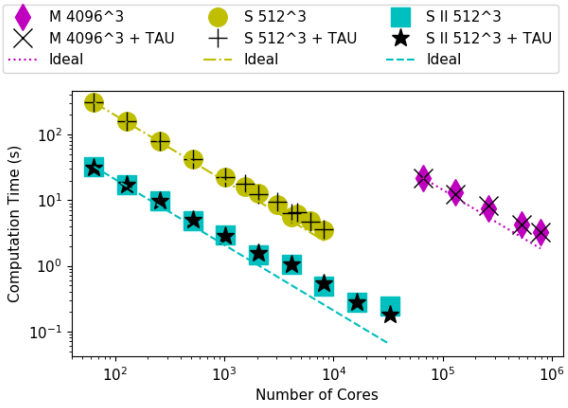


Figure 1: Strong scaling and overhead of profiling with TAU for 30 timesteps of the Klein-Gordon equation at spatial resolutions of 512^3 and 4096^3 grid points on Mira (M), Shaheen (S), and Shaheen II (S II).

3 PROFILING TOOLS, FFT LIBRARY AND TEST PLATFORMS

3.1 The Profiling Tools

A summary of parallel program profiling tools is given in Table 1. Here, a general introduction to profiling tools is given, and then more detailed information on the profiling tools used in this study follows.

A number of the profiling tools used are open source (Scalsaca, TAU, IPM, mpiP and FPMPI) and if not installed on a machine, require the user to install them. Some of these tools share components (in particular Scalasca and TAU). Many of the open source tools (such as Scalasca, and TAU) are also performance monitoring research testbeds, thus they develop quite rapidly. The open source tools tend to rely on very similar components, in particular, binutils, dyninst, libunwind and Papi.

Craypat is a vendor produced profiling tool. The developers can therefore make a tradeoff on configurability by the user, for ease of use and can take advantage of special hardware features. As an example, at present, there is no standard interface to measure energy use or input and output utilization (although there are efforts towards this such as Power API[20], the Global Extensible Open Power Manager[14], Energy Aware Runtime systems[11] and PowerAPI[10]) but Craypat can provide this information to the user while the open source tools that can do this, TAU and Extrae, need to be configured to do so.

3.1.1 IPM. [16, 48] is a lightweight profiling tool meant to be easy to use and display performance information that can also be collected at runtime. It primarily obtains performance information about MPI calls (primarily point to point communication), and can also obtain information about floating point operations if configured to obtain information using Papi[45].

3.1.2 Craypat. is part of the Cray parallel programming productivity suite. It comes installed on Cray computers and is directly integrated on each computer to enable ease of use and to take advantage of hardware features available on the computer. Craypat has a light profiling mode that allows the user to profile code without doing manual instrumentation and to enable for performance improvement by profiling program execution and then using the profiling information to improve subsequent program executions, for example by rank re-ordering. Craypat can instrument executables produced by GNU, Cray, PGI and Intel compilers. A previous study of Craypat is in [41].

Associated with Craypat is a desktop/laptop visualization package, Reveal.

3.1.3 FPMPI. [21] is an open source lightweight profiling tool that primarily targets MPI functions. It is no longer under active development, but is lightweight and still builds on modern CPU only architectures.

3.1.4 mpiP. [47] is an open source lightweight MPI call profiling library. It profiles most of the MPI 2 and MPI 3 calls, not just point to point communications as is done by IPM. Many of its developers are currently developing Openspeedshop.

3.1.5 Scalasca. [17] is a parallel profiling and tracing toolkit developed primarily by FZ Jülich. It offers an incremental performance analysis procedure that integrates runtime summaries with in-depth studies of concurrent behavior. It primarily traces and profiles C, C++ and Fortran programs. In the experiments reported here Score-P[26] was used for the instrumentation. It also has an associated performance visualization tool, Cube.

3.1.6 TAU. [7, 30, 31, 42] is an open source tracing and profiling tool that is developed and maintained by the University of Oregon and ParaTools, Inc. It has the capability to instrument Fortran, C and C++ programs using OpenMP, MPI, Cuda and OpenCL. It has also had the capability of instrumenting and profiling Java and Python programs. While TAU can do binary instrumentation, in typical use, TAU will be setup to perform specific profiling or tracing by instrumenting and then compiling the instrumented source code. This minimizes extra overhead and allows a mapping of the function calls in the output. Good use of TAU requires that the tool user have a good understanding of both the code being studied as well as of TAU. TAU also has typical setup options for tracing and profiling parallel programs. These can be setup and installed for general use. In order to instrument a program using TAU one has to first compile it for each compiler separately so that the produced executable together with the specified parameter options can be used for the profiling instrumentation. It also has an associated visualization tool Paraprof.

A default profiling and tracing experiment will generate a file per process. For a large number of nodes, this can generate a significant amount of data.

3.2 The Parallel FFT Libraries

3.2.1 2decomp&FFT. [29] is a parallel three dimensional Fortran FFT library. It uses external one dimensional FFT libraries and focuses on providing transpose routines. In this study FFTW[15] was used for the one dimensional FFT. The present study used version 1.5 of 2decomp&FFT.

Table 1: An overview of parallel performance tools.

Tool	Profiling	Tracing	Open Source	Development	Support	Forum
CrayPat[12]	Yes	Yes	No	Yes	Paid	No
Extrae[6]	Yes	Yes	Yes	Active	Yes	No
FPMPI[21]	Yes	Yes	Yes	No	No	No
Hpctoolkit[1]	Yes	Yes	Yes	Active	Paid	Open
IPM[16, 48]	Yes	No	Yes	Maintain	No	No
mpiP[47]	Yes	No	Yes	Very little	No	Open
Openspeedshop[27]	Yes	Yes	Yes	Yes	Paid	Closed
PAPIex[33]	Yes	No	Yes	Yes	No	No
Scalasca[17]	Yes	Yes	Yes	Active	Free best effort	No
TAU[7, 30, 31, 42]	Yes	Yes	Yes	Active	Paid	Open
Vtune[23]	Yes	Yes	No	Yes	Paid	Open

3.2.2 FFTE. [43, 44] is a Fortran library and for parallel one, two and three dimensional FFTs. The MPI-parallel version only works correctly when the number of MPI processes divides the grid dimensions. The present study used version 7.0 of FFTE.

3.3 Test Platforms

3.3.1 Shaheen. [46] was an IBM Blue Gene P supercomputer at the King Abdullah University of Science and Technology. It had IBM PowerPC 450 4 core 0.85GHz processors, a 3D torus network and 65,536 compute cores. This machine was used for the initial studies.

3.3.2 Mira. [2] was an IBM Blue Gene Q supercomputer at the Argonne National Laboratory. It had IBM PowerPC A2 16 core 1.6GHz processors, a 5D torus network and 786,432 compute cores. This machine was used for the initial studies.

3.3.3 Shaheen II. [25] is a Cray XC-40 supercomputer at the King Abdullah University of Science and Technology. It has Intel Xeon E5-2698v3 16C 2.3GHz processors, a Dragonfly network interconnect and 197,568 compute cores.

4 PROFILING OVERHEAD MEASUREMENTS

The supplementary materials[28] includes procedures for installing IPM, TAU, mpiP, FPMPI and Scalasca on Shaheen II, some profiler outputs and code for reproducing the results. In the appendix, there are typical profiler outputs from TAU, Scalasca, Craypat, mpiP and FPMPI for both FFTE and 2decomp&FFT. When profiilng the Klein Gordon code using 2decomp&FFT, 2decomp&FFT was first compiled on its own with instrumentation, the Klein Gordon code using 2decomp&FFT was then compiled with instrumentation and using the corresponding instrumented 2decom&FFT library. The Klein Gordon code using FFTE integrated FFTE directly, and so both FFTE and the Klein Gordon code were compiled with instrumentation at the same time. In all cases GCC compilers were used, Cray and Intel compilers do show some performance improvements of upto 4% in time to solution for the uninstrumented programs on a single node, but as configuring each profiling tool with a range of compilers is time consuming, and GCC is highly portable, GCC was choosen as the compiler to do the measurements on. Profiling with TAU was done using *tau_f90.sh* with optimization *-O3* to compile and then running the code normally (for example for 64 cores using *srun -n 64 ./executable*), since TAU by default does lightweight profiling. Profiling with Scalasca was done using *scorep ftn* to compile and then running the code, for example for 64 cores with *scalasca -analyze srun -n 64 ./executable* or *scan srun -n 64 ./executable*. Profiling with Craypat-lite was done using *ftn* with optimization *-O3* to compile and then running the code normally (for example for 64 cores using *srun -n 64 ./executable*). Profiling with mpiP was done using *ftn* with optimization *-O3* to compile and link to the installed mpiP library, then running the code, for example for 64 cores using *srun -n 64 ./executable*. Profiling with FPMPI was done using *ftn* with optimization *-O3* to compile and link to the FPMPI library, and then running the code, for example for 64 cores using *srun -n 64 ./executable*. Only compiler flags were used to optimize the code.

Performance profiling measurements are presented in Table 2 for 30 timesteps of the numerical solution of the Klein-Gordon equation as done in [3]. Each measurement is repeated three times to give confidence in reproducibility, as suggested for example in [4]. On large core counts, this can result in wall clock times significantly less than 10 seconds. A wall clock time of 10 seconds is a good compromise between confidence that the supercomputer results are representative of typical cases and not using limited core hours only for performance measurement. In this study, the allocated available core hours were spent on setting up the tests and ensuring their correctness, so it was not possible to use wall clock times of at least 10 seconds on larger core counts. Table 2 compares Craypat-lite, Scalasca, TAU, mpiP and FPMPI on Shaheen II for a 512^3 discretization with 2decomp&FFT.

Table 2 shows that the overhead of profiling is within the noise obtained when reproducing the experiments. It also shows that FFTE and 2decomp&FFT have comparable levels of performance for an FFT of size 512^3 but with increasing core counts FFTE is typically faster than 2decomp&FFT because it uses fewer MPI_ALLTOALL calls than 2decomp&FFT. At lower core counts, transpositions are less costly and data alignment that allows the use of optimized instructions available in FFTW make the performance of 2decomp&FFT similar to that in FFTE. The reason FFTE uses fewer MPI_ALLTOALL calls is because in the fastest MPI parallel implementation of FFTE, the Fourier modes are not stored in a sorted order, requiring some programmer effort to make use of the library.

Table 2: Table comparing execution times for 30 time steps of FFT based Klein Gordon equation solvers for a 512^3 problem size using 2decomp&FFT with FFTW and using FFTE libraries. Instrumentation has been done with the Craypat-lite, Scalasca, TAU, mpiP and FPMPI tools on Shaheen II. The mean and standard deviation from 3 runs are shown.

Cores	FFT	No profiling	Craypat-lite	Scalasca	TAU	mpiP	FPMPI
64	2decomp&FFT	32.54 ± 0.10s	31.92 ± 0.38s	25.35 ± 0.02s	31.72 ± 0.50s	31.67 ± 0.10s	31.97 ± 0.41s
64	FFTE	30.68 ± 0.05s	30.69 ± 0.03s	30.67 ± 0.01s	30.45 ± 0.36s	29.90 ± 0.33s	31.11 ± 0.05s
128	2decomp&FFT	17.43 ± 0.17s	17.57 ± 0.24s	17.72 ± 0.94s	17.45 ± 0.17s	17.64 ± 0.03s	17.33 ± 0.07s
128	FFTE	17.76 ± 0.03s	18.26 ± 0.08s	18.89 ± 0.001s	18.06 ± 0.16s	18.62 ± 0.02s	18.17 ± 0.09s
256	2decomp&FFT	9.72 ± 0.26s	9.71 ± 0.03s	10.20 ± 0.07s	9.67 ± 0.01s	9.88 ± 0.23s	9.75 ± 0.06s
256	FFTE	9.73 ± 0.02s	9.73 ± 0.01s	9.81 ± 0.01s	9.78 ± 0.06s	9.70 ± 0.01s	9.93 ± 0.22s
512	2decomp&FFT	4.93 ± 0.07s	5.17 ± 0.26s	5.12 ± 0.04s	5.04 ± 0.13s	5.15 ± 0.17s	5.01 ± 0.03s
512	FFTE	5.09 ± 0.03s	4.99 ± 0.21s	4.97 ± 0.03s	4.91 ± 0.04s	4.86 ± 0.13s	4.76 ± 0.01s
1024	2decomp&FFT	2.93 ± 0.02s	3.15 ± 0.29s	2.87 ± 0.06s	2.89 ± 0.03s	3.10 ± 0.10s	2.92 ± 0.03s
1024	FFTE	2.48 ± 0.03s	2.24 ± 0.02s	2.59 ± 0.29s	2.24 ± 0.01s	2.21 ± 0.05s	2.30 ± 0.08s
2048	2decomp&FFT	1.45 ± 0.017s	1.67 ± 0.05s	1.64 ± 0.02s	1.57 ± 0.03s	1.51 ± 0.13s	1.86 ± 0.20s
2048	FFTE	1.30 ± 0.10s	1.62 ± 0.45s	1.29 ± 0.07s	1.26 ± 0.02s	1.19 ± 0.03s	1.67 ± 0.51s
4096	2decomp&FFT	1.04 ± 0.03s	1.01 ± 0.04s	0.91 ± 0.03s	1.05 ± 0.08s	0.93 ± 0.06s	0.91 ± 0.04s
4096	FFTE	0.66 ± 0.03s	0.659 ± 0.002s	0.64 ± 0.02s	0.63 ± 0.01s	0.67 ± 0.03s	0.70 ± 0.01s
8192	2decomp&FFT	0.487 ± 0.002s	0.50 ± 0.02s	0.54 ± 0.02s	0.54 ± 0.02s	0.56 ± 0.11s	0.49 ± 0.02s
8192	FFTE	0.50 ± 0.01s	0.53 ± 0.05s	0.53 ± 0.10s	0.47 ± 0.02s	0.48 ± 0.03s	0.51 ± 0.01s
16384	2decomp&FFT	0.28 ± 0.01s	0.27 ± 0.01s	0.35 ± 0.08s	0.28 ± 0.01s	0.28 ± 0.01s	0.28 ± 0.02s
16384	FFTE	0.255 ± 0.002s	0.26 ± 0.02	0.33 ± 0.06s	0.28 ± 0.03s	0.30 ± 0.10s	0.30 ± 0.04s
32768	2decomp&FFT	0.24 ± 0.04s	0.22 ± 0.04s	0.25 ± 0.06s	0.18 ± 0.01s	0.21 ± 0.03s	0.25 ± 0.06s
32768	FFTE	0.22 ± 0.04s	0.18 ± 0.02	0.19 ± 0.01s	0.16 ± 0.01s	0.31 ± 0.21s	0.18 ± 0.02s

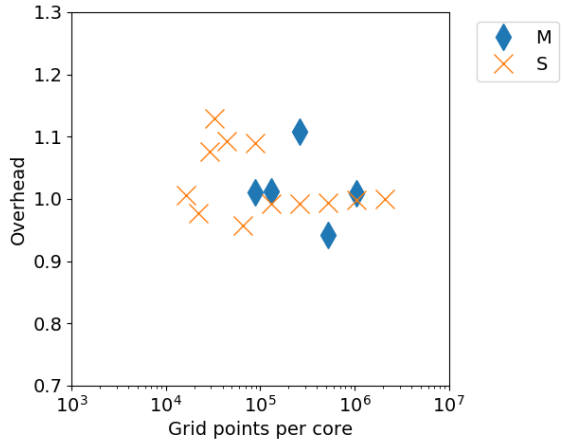
5 DISCUSSION

This study has primarily focused on measuring the ease of use and overhead of different performance profiling tools. The algorithms used in the tools and their implementation determine the amount of overhead, but the typical user of the tools is primarily interested in the measurement and its associated cost, rather than an in depth performance tool developers knowledge of the tools.

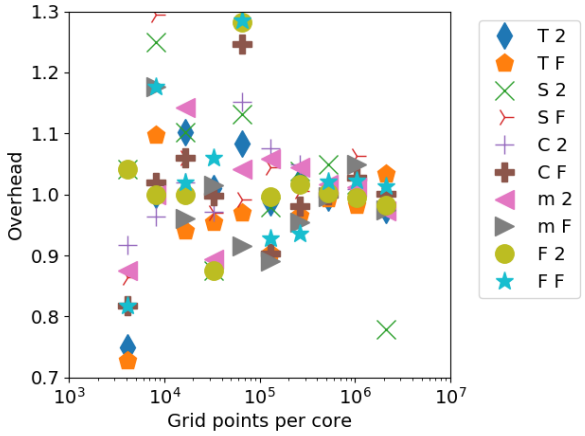
One can consider a derived overhead metric of a tool as the ratio of the computation time without profiling to that with profiling. Overheads are presented in Figures 2a and 2b for Mira, Shaheen and Shaheen II respectively. Measurements on Mira and Shaheen were not repeated multiple times, and since these are no longer operational, it is not possible to repeat the measurements. The torus networks on Mira and Shaheen isolate communication interference between running programs, but this does not occur on Shaheen II, which has more variability for time to solution of communication intensive programs. The figures show that default configurations of the used profiling tools have low overhead for the measured core counts for an application which primarily has collective communication calls. While the profiling tools can give information about job placement, in dragonfly networks, they do not give information about communication interference due to other jobs, such information may be helpful in job placement, job scheduling and in modifying routing communication algorithms. It may also be of help in evaluating network designs such as the Slim Fly topology[8] for the effects of job interference on variability in time to completion of communication intensive programs running on a portion of a supercomputer.

Mira and Shaheen have different types of torus networks, with Shaheen having processors about half as fast as Mira. Figures 2a and 1 show that when one considers grid points per core, scaling on Shaheen is better than Mira because the processor speed, memory bandwidth and network bandwidth on Shaheen are in better balance than on Mira. Similarly, Figure 1 shows that the parallel efficiency at high core counts on Shaheen II is not as good as Shaheen, nevertheless, the time to solution on Shaheen II is less than on Shaheen.

Craypat-lite and FPMPI automatically provide short summary text files with aggregate information at the end of each profiling experiment. mpiP also provides a summary text file, however this contains node level information, which while useful for understanding performance imbalance, can become quite lengthy when using thousands of cores. Scalasca, TAU and Craypat-lite also produce comprehensive measurement files that can be postprocessed to obtain further information. For Scalasca and TAU, a postprocessing step is needed after job completion to obtain a summary file with aggregate job information. In a production setting, it may be helpful to do this automatically. The resulting profiling information can be used to suggest code improvements as well as monitor the health of the parallel computer. For the current experiments, the summary file provided by FPMPI was the simplest to use as it gave the number of MPI_ALLTOALL calls which are most useful when understanding the performance of communication in the parallel FFT, which dominates the wall clock time. TAU, Scalasca and Craypat-lite also provide useful information on MPI_ALLTOALL to allow one to determine the reasons for the performance differences between the solver using FFTE and the solver using 2decomp&FFT. The summary information from mpiP is less useful for this as it is aggregated by MPI rank, rather than by MPI call.



(a) Calculated overhead of profiling with TAU for 30 timesteps of the Klein-Gordon equation on Mira (M) and Shaheen (S) for 4096^3 and 512^3 grids respectively.



(b) Calculated overhead of profiling with Klein Gordon solver for 30 timesteps on Shaheen II. The first character in the legend indicates the profiling tool, C: Craypat-lite, F: FPMPI, m: mpiP and T: TAU. The second character in the legend indicates the FFT library, F: FFTE and 2: 2decomp&FFT

Figure 2: Overhead and differences in repeatability on torus (Mira and Shaheen) and dragonfly (Shaheen II) networks

6 SUMMARY

Craypat-lite, Scalasca, mpiP, FPMPI and TAU offer reasonable summarized default lightweight profiling options for parallel programs which can be obtained by compiling the programs with appropriate wrappers. Reports produced using Craypat-lite have the most comprehensive information, though it is also possible to obtain more comprehensive information from TAU and Scalasca by changing the runtime configuration. On the core counts used here, all these programs have low enough overhead to be used in a production setting, allowing for monitoring of program performance. In parallel programs, a call graph of function information or timing of MPI routines is provided by most of the tools. Learning to use these is a transferable skill. The vendor provided performance tools may be more comprehensive and are automatically configured for the user, but since users typically use more than one kind of computer, the open source tools are preferable for initial training. For the beginning user, it is helpful to pre-install a lightweight low overhead configuration of a productivity tool for production setting use.

7 FURTHER WORK

Further work involves profiling other parallel FFT libraries, such as [5], [9], [13], [18], [19], [24], [32], [36], [37], [38], [39] and [40]. In addition profiling and tracing tools such as Extrae, PapiEx, Intel Vtune and OpenSpeedshop will be included in a more detailed comparison. The effects of job placement and communication interference on result reproducibility will also be studied, an area for which there are still challenges[22]. Finally, there are other numerical methods that can be used to solve the Klein Gordon equation[34], it would be interesting to use these other methods to aid in performance prediction and optimal matching of algorithm to hardware architecture.

ACKNOWLEDGMENTS

We thank José Gracia, Chirstoph Niethammer, Sameer Shende, Daisuke Takahashi and Brian Wylie for helpful discussions. B.L.'s work was primarily done while affiliated with the University of Michigan. B.K.M. was partially supported by HPC Europa 3 (INFRAIA-2016-1-730897) and the Estonian Center for Excellence in IT (TK148), and his work was mostly done while affiliated with the Institute of Computer Science at the University of Tartu. The computational resources used to build and test the programs were:

- Shaheen and Shaheen II operated by the KAUST Supercomputing Laboratory.
- Hazelhen at HLRS.
- K computer that was operated by RIKEN.
- Rocket and Vedur operated by the University of Tartu HPC center.
- Mira that was operated by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. 2010. HPCTOOLKIT: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 685–701. https://doi.org/10.1002/cpe.1553

[2] Argonne National Laboratory. 2020. Mira. https://www.alcf.anl.gov/alcf-resources/mira

[3] S. Aseeri, O. Batrasev, M. Icardi, B. Leu, A. Liu, N. Li, B. K. Muite, E. Müller, B. Palen, M. Quell, H. Servat, P. Sheth, R. Speck, M. Van Moer, and J. Vienne. 2015. Solving the Klein-Gordon Equation Using Fourier Spectral Methods: A Benchmark Test for Computer Performance. In *Proceedings of the Symposium on High Performance Computing* (Alexandria, Virginia) *(HPC '15)*. Society for Computer Simulation International, San Diego, CA, USA, 182–191. http://dl.acm.org/citation.cfm?id=2872599.2872622

[4] Samar Aseeri, Benson K. Muite, and Daisuke Takahashi. 2019. Reproducibility in Benchmarking Parallel Fast Fourier Transform Based Applications. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering* (Mumbai, India) *(ICPE '19)*. Association for Computing Machinery, New York, NY, USA, 5–8. https://doi.org/10.1145/3302541.3313105

[5] Alan Ayala, Stanimire Tomov, Azzam Haidar, and Jack Dongarra. 2020. *heFFTe: Highly Efficient FFT for Exascale.* Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-030-50371-0_19

[6] Barcelona Supercomputing Center. 2019. Extrae instrumentation package. https://tools.bsc.es/extrae

[7] David E. Bernholdt, Benjamin A. Allan, Robert Armstrong, Felipe Bertrand, Kenneth Chiu, Tamara L. Dahlgren, Kostadin Damevski, Wael R. Elwasif, Thomas G. W. Epperly, Madhusudhan Govindaraju, Daniel S. Katz, James A. Kohl, Manoj Krishnan, Gary Kumfert, J. Walter Larson, Sophia Lefantzi, Michael J. Lewis, Allen D. Malony, Lois C. Mclnnes, Jarek Nieplocha, Boyana Norris, Steven G. Parker, Jaideep Ray, Sameer Shende, Theresa L. Windus, and Shujia Zhou. 2006. A Component Architecture for High-Performance Scientific Computing. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 163–202. https://doi.org/10.1177/1094342006064488

[8] M. Besta and T. Hoefler. 2014. Slim Fly: A Cost Effective Low-Diameter Network Topology. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 348–359. https://doi.org/10.1109/SC.2014.34

[9] Anando G. Chatterjee, Mahendra K. Verma, Abhishek Kumar, Ravi Samtaney, Bilel Hadri, and Rooh Khurram. 2018. Scaling of a Fast Fourier Transform and a pseudo-spectral fluid solver up to 196608 cores. *J. Parallel and Distrib. Comput.* 113 (2018), 77 – 91. https://doi.org/10.1016/j.jpdc.2017.10.014

[10] Maxime Colmant, Romain Rouvoy, Mascha Kurpicz, Anita Sobe, Pascal Felber, and Lionel Seinturier. 2018. The Next 700 CPU Power Models. *Journal of Systems and Software* 144 (July 2018), 382–396. https://doi.org/10.1016/j.jss.2018.07.001

[11] Julita Corbalan and Luigi Brochard. 2019. EAR: Energy management framework for supercomputers. https://www.bsc.es/sites/default/files/public/bscw2/content/software-app/technical-documentation/ear.pdf

[12] Cray. 2020. Cray Performance Measurement and Analysis Tools User Guide (7.0.0) S-2376. https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide/craypat

[13] Lisandro Dalcin, Mikael Mortensen, and David E. Keyes. 2019. Fast parallel multidimensional FFT using advanced MPI. *J. Parallel and Distrib. Comput.* 128 (2019), 137 – 150. https://doi.org/10.1016/j.jpdc.2019.02.006

[14] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. 2017. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. In *High Performance Computing*, Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes (Eds.). Springer International Publishing, Cham, 394–412.

[15] M. Frigo and S. G. Johnson. 2005. The Design and Implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[16] Karl Fürlinger, Nicholas J. Wright, and David Skinner. 2010. Performance Analysis and Workload Characterization with IPM. In *Tools for High Performance Computing 2009*, Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 31–38.

[17] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. 2010. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 702–719. https://doi.org/10.1002/cpe.1556

[18] A. Gholami, J. Hill, D. Malhotra, and G. Biros. 2016. AccFFT: A library for distributed-memory FFT on CPU and GPU architectures,. http://arxiv.org/abs/1506.07933

[19] J.H. Göbbert. 2017. nb3dfft. https://gitlab.version.fz-juelich.de/goebbert/nb3dfft

[20] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti, and J. H. Laros III. 2016. Standardizing Power Monitoring and Control at Exascale. *Computer* 49, 10 (Oct 2016), 38–46. https://doi.org/10.1109/MC.2016.308

[21] William Gropp and Kristopher Buschelman. 2020. FPMPI. https://www.mcs.anl.gov/research/projects/fpmpi/WWW/index.html

[22] Sascha Hunold, Alexandra Carpen-Amarie, and Jesper Larsson Träff. 2014. Reproducible MPI Micro-Benchmarking Isn't As Easy As You Think. In *Proceedings of the 21st European MPI Users' Group Meeting* (Kyoto, Japan) *(EuroMPI/ASIA '14)*. Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1145/2642769.2642785

[23] Intel. 2019. Vtune. https://software.intel.com/en-us/vtune

[24] Andreas Jocksch, Matthias Kraushaar, and David Daverio. [n.d.]. Optimized all-to-all communication on multicore architectures applied to FFTs with pencil decomposition. *Concurrency and Computation: Practice and Experience* 31, 16 ([n. d.]), e4964. https://doi.org/10.1002/cpe.4964

[25] KAUST Supercomputing Laboratory. 2020. Shaheen II. https://www.hpc.kaust.edu.sa/content/shaheen-ii

[26] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. 2012. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*, Holger Brunst, Matthias S. Müller, Wolfgang E. Nagel, and Michael M. Resch (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–91.

[27] Krell Institute. 2020. OpenSpeedshop. https://openspeedshop.org/

[28] Brian Leu, Samar Aseeri, and Benson Muite. 2020. *Programs and Sample data for "A Comparison of Parallel Profiling Tools for Programs utilizing the FFT"*. https://doi.org/10.5281/zenodo.4032591

[29] N. Li and S. Laizet. 2010. 2DECOMP&FFT – A highly scalable 2D decomposition library and FFT interface. In *Cray User Group 2010 conference*.

[30] K. A. Lindlan, J. Cuny, A. D. Malony, S. Shende, B. Mohr, R. Rivenburgh, and C. Rasmussen. 2000. A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*. 49–49. https://doi.org/10.1109/SC.2000.10052

[31] A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile. 2005. Performance technology for parallel and distributed component software. *Concurrency and Computation: Practice and Experience* 17, 2-4 (2005), 117–141. https://doi.org/10.1002/cpe.931

[32] A.V. Mohanan, C. Bonamy, and P. Augier. 2018. FluidFFT: common API (C++ and Python) for Fast Fourier Transform HPC libraries,. https://arxiv.org/abs/1807.01775

[33] Philip Mucci. 2020. PAPIex. https://bitbucket.org/minimalmetrics/papiex-oss/src/master/

[34] B. K. Muite and Samar Aseeri. 2020. Benchmarking Solvers for the One Dimensional Cubic Nonlinear Klein Gordon Equation on a Single Core. In *Benchmarking, Measuring, and Optimizing*, Wanling Gao, Jianfeng Zhan, Geoffrey Fox, Xiaoyi Lu, and Dan Stanzione (Eds.). Springer International Publishing, Cham, 172–184.

[35] Dmitry Nikitenko, Alexander Antonov, Pavel Shvets, Sergey Sobolev, Konstantin Stefanov, Vadim Voevodin, Vladimir Voevodin, and Sergey Zhumatiy. 2017. JobDigest – Detailed System Monitoring-Based Supercomputer Application Behavior Analysis. In *Supercomputing*, Vladimir Voevodin and Sergey Sobolev (Eds.). Springer International Publishing, Cham, 516–529.

[36] D. Pekurovsky. 2012. P3DFFT: a framework for parallel computations of Fourier transforms in three dimensions. *SIAM Journal on Scientific Computing* 34, 4 (08 2012), C192–C209. https://doi.org/10.1137/11082748X

[37] M. Pippig. 2013. PFFT - An Extension of FFTW to Massively Parallel Architectures. *SIAM J. Sci. Comput.* 35 (2013), C213 – C236. https://doi.org/10.1137/120885887

[38] S. Plimpton. 2017. Parallel FFT. http://www.sandia.gov/~sjplimp/docs/fft/README.html

[39] S. Plimpton, A. Kohlmemer, P. Blood, and P. Coffman. 2018. fftmpi. https://fftmpi.sandia.gov/

[40] A. Pope, D. Daniel, and N. Frontiere. 2017. SWFFT. https://xgitlab.cels.anl.gov/hacc/SWFFT

[41] M. Rajan. 2007. Experiences with the use of CrayPat in Performance Analysis. In *Proceedings Cray User Group*. https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/rajan_paper_cug07.pdf

[42] Sameer S. Shende and Allen D. Malony. 2006. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311. https://doi.org/10.1177/1094342006064482

[43] D. Takahashi. 2003. Efficient implementation of parallel three-dimensional FFT on clusters of PCs. *Computer Physics Communications* 152, 2 (2003), 144 – 150. https://doi.org/10.1016/S0010-4655(02)00818-4

[44] D. Takahashi. 2010. *An Implementation of Parallel 3-D FFT with 2-D Decomposition on a Massively Parallel Cluster of Multi-core Processors*. Springer Berlin Heidelberg, Berlin, Heidelberg, 606–614. https://doi.org/10.1007/978-3-642-14390-8_63

[45] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. 2010. Collecting Performance Data with PAPI-C. In *Tools for High Performance Computing 2009*, Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 157–173.

[46] Top 500. 2020. Shaheen. https://www.top500.org/system/176502/

[47] Jeffrey S. Vetter and Michael O. McCracken. 2001. Statistical Scalability Analysis of Communication Operations in Distributed Applications. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming* (Snowbird, Utah, USA) *(PPoPP '01)*. ACM, New York, NY, USA, 123–132. https://doi.org/10.1145/379539.379590

[48] N. J. Wright, K. Füerlinger, and D. Skinner. 2010. Effective Performance Measurement at Petascale Using IPM. In *2010 IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)(ICPADS)*, Vol. 00. 373–380. https://doi.org/10.1109/ICPADS.2010.16

A PROFILERS AND TYPICAL RESULTS

This section contains typical output obtained with the profiling tools.

A.1 IPM

Example report generated using IPM generated on Vedur when running the Klein Gordon solver using 64 cores with the 2decomp&FFT library.

```
##IPMv2.0.6############################################################
#
# command   : ./Kg
# start     : Tue Sep 18 04:34:07 2018   host      : klots6
# stop      : Tue Sep 18 04:37:46 2018   wallclock : 219.70
# mpi_tasks : 64 on 2 nodes              %comm     : 0.00
# mem [GB]  : 33.50                      gflop/sec : 0.00
#
#           :       [total]        <avg>          min          max
# wallclock :      14040.03       219.38       219.30       219.70
# MPI       :          0.00         0.00         0.00         0.00
# %wall     :
#   MPI     :                       0.00         0.00         0.00
# #calls    :
#   MPI     :           128            2            2            2
# mem [GB]  :         33.50         0.52         0.51         0.53
#
##########################################################################
```

A.2 Scalasca

Example report generated using Scalasca from a run on Shaheen II when running the Klein Gordon solver using 1024 cores. The first report is for the 2decomp&FFT library.

```
Estimated aggregate size of event trace:                   15MB
Estimated requirements for largest trace buffer (max_buf): 15kB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       4097kB
(hint: When tracing set SCOREP_TOTAL_MEMORY=4097kB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B]  visits time[s] time[%]  time/visit[us]  region
         ALL     15,107 289,793 5055.35   100.0        17444.68  ALL
         MPI     14,130 249,856 1989.65    39.4         7963.18  MPI
         USR        792  32,769  171.31     3.4         5227.87  USR
         COM        144   6,144 2894.37    57.3       471088.17  COM
      SCOREP         41   1,024    0.02     0.0           20.65  SCOREP

         MPI     11,616 180,224 1786.14    35.3         9910.64  MPI_Alltoall
         MPI        792  12,288    3.04     0.1          247.41  MPI_Reduce
         USR        720  30,720  161.56     3.2         5259.24  storeold
         MPI        462   7,168   10.77     0.2         1502.72  MPI_Allreduce
         MPI        384  16,384   36.63     0.7         2235.54  MPI_Cart_sub
         MPI        240  10,240    3.32     0.1          324.04  MPI_Cart_create
         MPI        192   8,192    0.00     0.0            0.46  MPI_Cart_coords
         MPI        144   6,144    0.01     0.0            1.97  MPI_Cart_shift
         MPI        132   2,048    2.36     0.0         1154.77  MPI_Bcast
         COM         96   4,096  545.30    10.8       133129.63  enercalc
         MPI         48   2,048    0.00     0.0            0.34  MPI_Comm_rank
         MPI         48   2,048    0.00     0.0            0.72  MPI_Comm_size
      SCOREP         41   1,024    0.02     0.0           20.65  KgScalasca
         COM         24   1,024    0.01     0.0            7.29  readinputfile
         USR         24       1    0.09     0.0        91238.32  saveresults
         MPI         24   1,024    0.00     0.0            1.21  MPI_Cart_get
         USR         24   1,024    0.14     0.0          137.21  getgrid
         USR         24   1,024    9.52     0.2         9293.37  initialdata
         MPI         24   1,024   55.51     1.1        54204.75  MPI_Init
         COM         24   1,024 2349.06    46.5      2294003.24  kg
         MPI         24   1,024   91.87     1.8        89712.60  MPI_Finalize
```

The second report is for the FFTE library.

```
Estimated aggregate size of event trace:                   10GB
Estimated requirements for largest trace buffer (max_buf): 10MB
Estimated memory requirements (SCOREP_TOTAL_MEMORY):       12MB
(hint: When tracing set SCOREP_TOTAL_MEMORY=12MB to avoid intermediate flushes
 or reduce requirements using USR regions filters.)

flt     type max_buf[B]      visits time[s] time[%] time/visit[us]  region
         ALL 10,366,309 407,990,273 2774.85   100.0           6.80  ALL
         USR 10,350,470 407,648,257  522.46    18.8           1.28  USR
         MPI     11,768     182,272 1523.49    54.9        8358.36  MPI
         COM      4,030     158,720  728.88    26.3        4592.22  COM
      SCOREP         41       1,024    0.02     0.0          21.29  SCOREP

         USR  4,432,896 174,587,904   20.09     0.7           0.12  fft8
         USR  2,955,264 116,391,936  174.71     6.3           1.50  fft8b
         USR  1,477,632  58,195,968  122.84     4.4           2.11  fft8a
         USR  1,477,632  58,195,968   19.87     0.7           0.34  fft235
         MPI     10,064     151,552 1434.73    51.7        9466.93  MPI_Alltoall
         USR      5,876     231,424    0.08     0.0           0.33  factor
         COM      1,950      76,800   24.64     0.9         320.87  pzfft3dv
         MPI      1,360      20,480    0.79     0.0          38.43  MPI_Reduce
         COM      1,092      43,008  213.34     7.7        4960.39  pzfft3dvb
         COM        832      32,768  186.60     6.7        5694.53  pzfft3dvf
         USR        780      30,720  177.05     6.4        5763.49  storeold
         USR        234       9,216    0.09     0.0           9.83  settbl0
         MPI        136       2,048    2.58     0.1        1260.51  MPI_Bcast
         COM        104       4,096   81.66     2.9       19935.89  enercalc
         USR         78       3,072    0.00     0.0           1.63  settbl
         MPI         52       2,048    0.13     0.0          63.06  MPI_Comm_free
         MPI         52       2,048   19.49     0.7        9515.21  MPI_Comm_split
      SCOREP         41       1,024    0.02     0.0          21.29  KgScalasca
         COM         26       1,024    0.01     0.0           9.56  readinputfile
         COM         26       1,024  222.63     8.0      217414.70  kg
         MPI         26       1,024   15.31     0.6       14947.74  MPI_Finalize
         MPI         26       1,024    0.00     0.0           1.33  MPI_Comm_size
         MPI         26       1,024    0.00     0.0           0.40  MPI_Comm_rank
         USR         26           1    0.02     0.0       16701.02  saveresults
         MPI         26       1,024   50.47     1.8       49286.24  MPI_Init
         USR         26       1,024    0.04     0.0          39.71  getgrid
         USR         26       1,024    7.65     0.3        7475.06  initialdata
```

A.3 TAU

Example report generated using TAU when running the Klein Gordon solver using 1024 cores of Shaheen II. Due to space constraints, node level information and a summary of average information at a per node level are not presented. The first report is for the 2decomp&FFT library.

```
FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0       26,862  1:25:09.260        1024        1024    4989512 .TAU application
 99.5          155  1:24:42.397        1024        1024    4963279 main
 99.5    38:54.667  1:24:42.241        1024      249857    4963127 MAIN__
 34.3    29:13.799    29:13.799      180224           0       9731 MPI_Alltoall()
 14.0     8:11.688    11:55.319        4096       36864     174638 enercalc_
  3.7     3:06.982     3:06.982        1024           0     182600 MPI_Finalize()
  2.9     2:27.637     2:27.637       30720           0       4806 storeold_
  1.1       55,457       55,457        1024           0      54157 MPI_Init()
  0.8           60       40,969        1024        2048      40009 readinputfile_
  0.8       40,908       40,908        2048           0      19975 MPI_Bcast()
  0.8       39,554       39,554       16384           0       2414 MPI_Cart_sub()
  0.3       14,072       14,072        7168           0       1963 MPI_Allreduce()
  0.2        9,968        9,968        1024           0       9735 initialdata_
  0.1        6,151        6,151       12288           0        501 MPI_Reduce()
  0.0        1,209        1,209       10240           0        118 MPI_Cart_create()
  0.0           46           46        1024           0         45 getgrid_
  0.0           18           18           1           0      18054 saveresults_
  0.0            9            9        8192           0          1 MPI_Cart_coords()
  0.0            6            6        6144           0          1 MPI_Cart_shift()
  0.0            1            1        1024           0          2 MPI_Cart_get()
  0.0        0.894        0.894        2048           0          0 MPI_Comm_size()
  0.0        0.599        0.599        2048           0          0 MPI_Comm_rank()
  0.0        0.384        0.384        1024           0          0 MPI_Type_size()
```

The second report is for the FFTE library.

```
FUNCTION SUMMARY (total):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0           7    49:20.598        1024        1024    2891210 .TAU application
100.0    3:56.906    49:20.591        1024      111617    2891203 KG
 72.9      24,649    35:58.678       76800      306176      28108 PZFFT3DV
 46.6    22:59.941    22:59.941      151552           0       9105 MPI_Alltoall()
 38.5     3:27.367    18:59.314       43008 3.31162E+07      26491 PZFFT3DVB
 33.6     3:04.985    16:34.543       32768 2.52314E+07      30351 PZFFT3DVF
 14.5     1:13.790     7:09.595        4096       32768     104882 ENERCALC
 12.2       49,956     6:01.563 5.8196E+07 1.60597E+08          6 FFT235
  7.0       25,215     3:26.386 1.02401E+08 1.02401E+08          2 FFT8 [THROTTLED]
  6.8     3:20.838     3:20.838        1024           0     196132 MPI_Finalize()
  6.0     2:58.012     2:58.012       30720           0       5795 STOREOLD
  5.5     2:41.939     2:41.939 1.02401E+08           0          2 FFT8B [THROTTLED]
  4.2     2:04.451     2:04.451  5.8196E+07           0          2 FFT8A
  1.8       52,073       52,073        1024           0      50853 MPI_Init()
  1.2           36       35,458        1024        2048      34628 READINPUTFILE
  1.2       35,422       35,422        2048           0      17296 MPI_Bcast()
  0.5       16,185       16,185        2048           0       7903 MPI_Comm_split()
  0.3        7,671        7,671        1024           0       7491 INITIALDATA
  0.0          859          859       20480           0         42 MPI_Reduce()
  0.0            8          114        3072       12288         37 SETTBL
  0.0          104          104        9216           0         11 SETTBL0
  0.0           74           74        1024           0         73 GETGRID
  0.0           57           57      231424           0          0 FACTOR
  0.0           22           22           1           0      22157 SAVERESULTS
  0.0           18           18        2048           0          9 MPI_Comm_free()
  0.0        0.642        0.642        1024           0          1 MPI_Comm_size()
  0.0        0.268        0.268        1024           0          0 MPI_Comm_rank()
```

A.4 Craypat-lite

Example report generated using Craypat-lite when running the Klein Gordon solver using 1024 cores of Shaheen II. Information on file I/O is removed since it is not relevant for this work, similarly much of the information in the first table is replicated in the second table, so the first table is removed. The first report is for the 2decomp&FFT library.

```
################################################################
#                                                              #
#            CrayPat-lite Performance Statistics               #
#                                                              #
################################################################

CrayPat/X:  Version 7.1.3 Revision 09f1a4886  11/29/19 13:50:15
Experiment:                   lite  lite-samples
Number of PEs (MPI ranks):   1,024
Numbers of PEs per Node:        32  PEs on each of  32  Nodes
Numbers of Threads per PE:       1
```

```
Number of Cores per Socket:     16
Execution start time:  Sat Sep 12 13:29:17 2020
System name and speed:  nid03852  2.301 GHz (nominal)
Intel Haswell    CPU  Family:  6  Model: 63  Stepping:  2
DRAM: 128 GiB DDR4-2133 on 2.3 GHz nodes


Avg Process Time:              5.33 secs
High Memory:               63,301.1 MiBytes     61.8 MiBytes per PE
Observed CPU clock boost:     116.1 %
Percent cycles stalled:        28.7 %
Instr per Cycle:               1.84
I/O Read Rate:             82.093269 MiBytes/sec
I/O Write Rate:             0.936558 MiBytes/sec


Notes for table 2:

  This table shows functions, and line numbers within functions, that
    have significant exclusive sample hits, averaged across ranks.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O samp_profile+src ...

Table 2:  Profile by Group, Function, and Line

  Samp% |  Samp | Imb. |  Imb. | Group
        |       | Samp | Samp% |  Function=[MAX10]
        |       |      |       |   Source
        |       |      |       |    Line
        |       |      |       |     PE=HIDE

 100.0% | 524.6 |   -- |    -- | Total
|-------------------------------------------------------------------------
|  95.7% | 502.1 |   -- |    -- | ETC
||------------------------------------------------------------------------
||  28.8% | 151.2 | 17.8 | 10.6% | main
||  13.3% |  69.9 | 11.1 | 13.7% | __decomp_2d_MOD_transpose_x_to_y_complex
||  12.3% |  64.8 | 11.2 | 14.8% | __decomp_2d_MOD_transpose_y_to_x_complex
||   5.7% |  29.8 | 21.2 | 41.7% | apply
||   4.0% |  20.9 | 16.1 | 43.5% | __decomp_2d_MOD_transpose_y_to_z_complex
||   3.5% |  18.3 |  9.7 | 34.6% | __memcpy_avx_unaligned_erms
||   2.8% |  14.7 | 13.3 | 47.5% | __hypot_finite
||   1.9% |  10.2 |  6.8 | 40.1% | __decomp_2d_MOD_transpose_z_to_y_complex
||   1.8% |   9.5 |  1.5 | 13.6% | __decomp_2d_MOD_transpose_x_to_y_real
||   1.6% |   8.5 |  6.5 | 43.2% | __decomp_2d_fft_MOD_fft_3d_c2c
||=========================================================================
|   4.2% |  22.2 |   -- |    -- | FFTW
|=========================================================================

Observation:  MPI utilization

    No suggestions were made because MPI data was not traced (use
    pat_build -g mpi).


Observation:  Program sensitivity to memory latency

    The relatively low memory bandwidth utilization but significant rate
    of stalls in the program suggest that its performance is limited by
    memory latency.
    It could be beneficial to improve prefetching in loops in functions
    high in the profile, by modifying compiler-generated prefetches or
    inserting directives into the source code.


Notes for table 3:

  This table shows memory traffic to local and remote memory for numa
    nodes, taking for each numa node the maximum value across nodes.
    It also shows the balance in memory traffic by showing the top 3
    and bottom 3 node values.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O mem_bw ...

Table 3:  Memory Bandwidth by Numanode

  Memory |   Local |  Remote |   Thread |  Memory |  Memory | Numanode
 Traffic |  Memory |  Memory |     Time | Traffic | Traffic |  Node Id=[max3,min3]
  GBytes | Traffic | Traffic |          |  GBytes |       / |   PE=HIDE
         |  GBytes |  GBytes |          |   / Sec | Nominal |
         |         |         |          |         |    Peak |
|-------------------------------------------------------------------------
|   28.90 |   28.15 |    0.75 | 7.898865 |    3.66 |    5.4% | numanode.0
||------------------------------------------------------------------------
||   28.88 |   28.14 |    0.74 | 5.639828 |    5.12 |    7.5% | nid.3864
||   28.88 |   28.15 |    0.72 | 5.639401 |    5.12 |    7.5% | nid.3857
||   28.87 |   28.15 |    0.72 | 5.642093 |    5.12 |    7.5% | nid.3865
||   28.67 |   27.94 |    0.73 | 5.629921 |    5.09 |    7.5% | nid.3875
||   28.65 |   27.92 |    0.73 | 5.631063 |    5.09 |    7.5% | nid.3882
||   28.64 |   27.91 |    0.72 | 5.643203 |    5.07 |    7.4% | nid.3883
||=========================================================================
|   28.84 |   27.98 |    0.86 | 5.737451 |    5.03 |    7.4% | numanode.1
||------------------------------------------------------------------------
||   28.82 |   27.98 |    0.84 | 5.595931 |    5.15 |    7.5% | nid.3856
||   28.82 |   27.97 |    0.84 | 5.595890 |    5.15 |    7.5% | nid.3872
||   28.80 |   27.95 |    0.85 | 5.598326 |    5.14 |    7.5% | nid.3873
||   28.59 |   27.74 |    0.85 | 5.596116 |    5.11 |    7.5% | nid.3875
||   28.56 |   27.71 |    0.85 | 5.596610 |    5.10 |    7.5% | nid.3878
||   28.55 |   27.72 |    0.82 | 5.602052 |    5.10 |    7.5% | nid.3853
|=========================================================================

Notes for table 4:

  This table shows energy and power usage for the nodes with the
    maximum, mean, and minimum usage, as well as the sum of usage over
    all nodes.
    Energy and power for accelerators is also shown, if applicable.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O program_energy ...

Table 4:  Program energy and power usage (from Cray PM)

   Node | Node Power |  Process | Node Id=[mmm]
 Energy |        (W) |     Time |  PE=HIDE
    (J) |            |          |

 56,092 | 10,524.474 | 5.329682 | Total
|-------------------------------------------------
|  1,828 |    343.036 | 5.328890 | nid.3854
|  1,756 |    329.623 | 5.327291 | nid.3876
|  1,687 |    316.664 | 5.327410 | nid.3875
|=================================================

Program invocation:
  /2decomp/run_craypat_1024/./KgCraypat

==============  End of CrayPat-lite output  ==========================
```

The second report is for the FFTE library.

```
############################################################
#                                                          #
#            CrayPat-lite Performance Statistics           #
#                                                          #
############################################################

CrayPat/X:  Version 7.1.3 Revision 09f1a4886  11/29/19 13:50:15
Experiment:                  lite  lite-samples
Number of PEs (MPI ranks):   1,024
Numbers of PEs per Node:        32  PEs on each of  32  Nodes
Numbers of Threads per PE:       1
Number of Cores per Socket:     16
Execution start time:  Thu Sep 10 17:43:34 2020
System name and speed:  nid07102  2.301 GHz (nominal)
Intel Haswell    CPU  Family:  6  Model: 63  Stepping:  2
DRAM: 128 GiB DDR4-2133 on 2.3 GHz nodes


Avg Process Time:              3.15 secs
High Memory:               47,041.3 MiBytes     45.9 MiBytes per PE
Observed CPU clock boost:     120.7 %
Percent cycles stalled:        46.9 %
Instr per Cycle:               1.86
I/O Read Rate:             89.733103 MiBytes/sec
I/O Write Rate:             0.066739 MiBytes/sec

Notes for table 2:

  This table shows functions, and line numbers within functions, that
    have significant exclusive sample hits, averaged across ranks.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O samp_profile+src ...

Table 2:  Profile by Group, Function, and Line

  Samp% |  Samp | Imb. |  Imb. | Group
        |       | Samp | Samp% |  Function=[MAX10]
        |       |      |       |   Source
        |       |      |       |    Line
        |       |      |       |     PE=HIDE

 100.0% | 306.9 |   -- |    -- | Total
|-------------------------------------------------------------------------
|  62.2% | 191.0 |   -- |    -- | MPI
||------------------------------------------------------------------------
||  61.9% | 189.9 | 11.1 |  5.5% | MPI_ALLTOALL
||========================================================================
|  26.5% |  81.4 |   -- |    -- | USER
||------------------------------------------------------------------------
||   5.9% |  18.2 |   -- |    -- | fft8b_
3|       |       |      |       |  ../FFT_libraries/ffte-7.0/kernel.f
||   5.5% |  16.9 |   -- |    -- | pzfft3dvb_
3|       |       |      |       |  FFT_libraries/ffte-7.0/mpi/pzfft3dv.f
||||---------------------------------------------------------------------
4|||   2.3% |   7.0 |  6.0 | 46.0% | line.309
4|||   2.0% |   6.1 |  2.9 | 32.6% | line.363
||||=====================================================================
||   5.1% |  15.7 |   -- |    -- | MAIN__
3|       |       |      |       |  KleinGordon/ffte-7.0/mpi2/KgSemiImp3d.f90
||||---------------------------------------------------------------------
4|||   2.2% |   6.6 |  6.4 | 48.9% | line.278
4|||   1.6% |   5.0 |  5.0 | 50.4% | line.295
||||=====================================================================
||   4.6% |  14.1 |   -- |    -- | pzfft3dvf_
3|       |       |      |       |  FFT_libraries/ffte-7.0/mpi/pzfft3dv.f
||||---------------------------------------------------------------------
4|||   1.6% |   5.0 |  4.0 | 44.9% | line.144
4|||   1.5% |   4.7 |  2.3 | 33.5% | line.171
4|||   1.3% |   4.0 |  4.0 | 50.2% | line.198
||||=====================================================================
||   4.2% |  13.0 |   -- |    -- | fft8a_
3|       |       |      |       |  ../FFT_libraries/ffte-7.0/kernel.f
||========================================================================
|  11.1% |  34.0 |   -- |    -- | ETC
||------------------------------------------------------------------------
||   8.9% |  27.4 | 12.6 | 31.5% | __memcpy_avx_unaligned_erms
||   1.5% |   4.7 |  5.3 | 52.8% | __hypot_finite
```

```
|==========================================================================

Observation:  MPI utilization

    No suggestions were made because MPI data was not traced (use
    pat_build -g mpi).


Observation:  Program sensitivity to memory latency

    The relatively low memory bandwidth utilization but significant rate
    of stalls in the program suggest that its performance is limited by
    memory latency.
    It could be beneficial to improve prefetching in loops in functions
    high in the profile, by modifying compiler-generated prefetches or
    inserting directives into the source code.


Notes for table 3:

  This table shows memory traffic to local and remote memory for numa
    nodes, taking for each numa node the maximum value across nodes.
    It also shows the balance in memory traffic by showing the top 3
    and bottom 3 node values.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O mem_bw ...

Table 3:  Memory Bandwidth by Numanode

  Memory |   Local |  Remote |    Thread |  Memory |  Memory | Numanode
 Traffic |  Memory |  Memory |      Time | Traffic | Traffic |  Node Id=[max3,min3]
  GBytes | Traffic | Traffic |           |  GBytes |       / |   PE=HIDE
         |  GBytes |  GBytes |           |   / Sec | Nominal |
         |         |         |           |         |    Peak |
|-------------------------------------------------------------------------
|   29.14 |   28.25 |    0.89 | 12.124747 |    2.40 |    3.5% | numanode.0
||------------------------------------------------------------------------
||   29.12 |   28.25 |    0.87 | 12.124747 |    2.40 |    3.5% | nid.7102
||   29.01 |   28.15 |    0.87 |  3.977943 |    7.29 |   10.7% | nid.7124
||   28.91 |   28.05 |    0.87 |  3.899221 |    7.42 |   10.9% | nid.7116
||   28.66 |   27.81 |    0.86 |  3.899679 |    7.35 |   10.8% | nid.7134
||   28.66 |   27.78 |    0.88 |  3.953556 |    7.25 |   10.6% | nid.7141
||   28.63 |   27.76 |    0.87 |  3.992280 |    7.17 |   10.5% | nid.7115
||========================================================================
|   28.65 |   27.67 |    0.98 |  3.992400 |    7.18 |   10.5% | numanode.1
||------------------------------------------------------------------------
||   28.63 |   27.67 |    0.96 |  3.944062 |    7.26 |   10.6% | nid.7122
||   28.62 |   27.67 |    0.95 |  3.830104 |    7.47 |   10.9% | nid.7119
||   28.58 |   27.62 |    0.95 |  3.942247 |    7.25 |   10.6% | nid.7118
||   28.38 |   27.41 |    0.97 |  3.936805 |    7.21 |   10.6% | nid.7124
||   28.37 |   27.42 |    0.95 |  3.948462 |    7.18 |   10.5% | nid.7134
||   28.30 |   27.36 |    0.94 |  3.942950 |    7.18 |   10.5% | nid.7133
|==========================================================================

Notes for table 4:

  This table shows energy and power usage for the nodes with the
    maximum, mean, and minimum usage, as well as the sum of usage over
    all nodes.
    Energy and power for accelerators is also shown, if applicable.
  For further explanation, see the "General table notes" below,
    or use:  pat_report -v -O program_energy ...

Table 4:  Program energy and power usage (from Cray PM)

   Node | Node Power |  Process | Node Id=[mmm]
 Energy |        (W) |     Time |  PE=HIDE
    (J) |            |          |

 34,208 | 10,846.016 | 3.153992 | Total
|-----------------------------------------------
|  1,197 |    351.704 | 3.404141 | nid.7102
|  1,059 |    336.577 | 3.146384 | nid.7141
|  1,023 |    325.237 | 3.145398 | nid.7124
|===============================================

Program invocation:
  /ffte-7.0/mpi2/run_craypat_1024/./KgCraypat

================  End of CrayPat-lite output  ==========================
```

A.5 FPMPI

Example report generated using FPMPI when running the Klein Gordon solver using 1024 cores of Shaheen II. Information on point to point communications has been removed since for this application all communication calls are collective. The first report is for the 2decomp&FFT library.

```
MPI Routine Statistics (FPMPI2 Version 2.4)
Options: FPMPI enabled, Collective sync,
Explanation of data:
Times are the time to perform the operation, e.g., the time for MPI_Send
Average times are the average over all processes, e.g., sum (time on each
process) / number of processes
Min and max values are over all processes
(Data is always average/min/max)
Amount of data is computed in bytes.  For point-to-point operations,
it is the data sent or received.  For collective operations, it is the
data contibuted to the operation.  E.g., for an MPI_Bcast, the amount of
data is the number of bytes provided by the root, counted only at the root.
For synchronizing collective operations, the average, min, and max time
spent synchronizing is shown next.
Calls by message size shows the fraction of calls that sent messages of a
particular size.  The bins are
0 bytes, 1-4 bytes, 5-8 bytes, 9-16, 17-32, 33-64, -128, -256, -512, -1024
 -4K, -8K, -16K, -32K, -64K, -128K, -256K, -512K, -1M, -4M, -8M, -16M,
 -32M, -64M, -128M, -256M, -512M, -1GB, >1GB.
Each bin is represented by a single digit, representing the 10's of percent
of messages within this bin.  A 0 represents precisely 0, a . (period)
represents more than 0 but less than 10%.  A * represents 100%.
Messages by message size shows similar information, but for the total
message size.

The experimental topology information shows the 1-norm distance that the
longest point-to-point message travelled, by process.

MPI_Pcontrol may be used to control the collection of data.  Use the values
defined in fpmpi.h, such as FPMPI_PROF_COLLSYNC, to control what data is
collected or reported by FPMPI2.

Date:           Sat Sep 12 15:10:38 2020
Processes:      1024
Execute time:   5.427
Timing Stats: [seconds] [min/max]       [min rank/max rank]
  wall-clock: 5.427 sec 5.420879 / 5.899858     659 / 0
        user: 5.565 sec 4.654960 / 5.751065     0 / 26
         sys: 0.1302 sec        0.064124 / 0.356651     771 / 320

Memory Usage Stats (RSS) [min/max KB]:  78992/81152

                Average of sums over all processes
Routine                 Calls       Time Msg Length    %Time by message length
                                                    0.........1.........1........
                                                                    K        M
MPI_Allreduce       :       9   0.000546         72 00*00000000000000000000000000
MPI_Alltoall        :     184       1.34   3.48e+08 00000000000000000002800000000
MPI_Bcast           :       2     0.0179     0.0586 0000.*00000000000000000000000
MPI_Reduce          :      12   0.000857         96 00*00000000000000000000000000
MPI_Cart_create     :      12    0.00143

Details for each MPI routine
                Average of sums over all processes
                                                % by message length
                                    (max over          0.........1.........1........
                                     processes [rank])           K        M
MPI_Allreduce:
        Calls     :          9            9 [   0] 00*00000000000000000000000000
        Time      :   0.000546      0.00059 [ 571] 00*00000000000000000000000000
        Data Sent :         72           72 [   0]
        SyncTime  :     0.0111        0.022 [ 935] 00*00000000000000000000000000
        By bin    : 5-8 [9,9]   [0.000496,0.00059] [0.0035,0.022]
MPI_Alltoall:
        Calls     :        184          184 [   0] 00000000000000000002800000000
        Time      :       1.34         1.44 [ 560] 00000000000000000002800000000
        Data Sent :   3.48e+08    348127232 [   0]
        SyncTime  :      0.856         1.01 [ 138] 00000000000000000000.*00000000
        By bin    : 524289-1048576      [36,36] [0.197,0.236] [0.0114,0.0401]
                  : 1048577-4194304     [148,148] [1.03,1.22] [0.078,0.981]
MPI_Bcast:
        Calls     :          2            2 [   0] 00005500000000000000000000000
        Time      :     0.0179       0.0339 [ 484] 0000.*00000000000000000000000
        Data Sent :     0.0586           60 [   0]
        By bin    : 17-32       [1,1]   [0,1.6e-05]
                  : 33-64       [1,1]   [4.98e-05,0.0339]
MPI_Reduce:
        Calls     :         12           12 [   0] 00*00000000000000000000000000
        Time      :   0.000857      0.00629 [1012] 00*00000000000000000000000000
        Data Sent :         96           96 [   0]
        By bin    : 5-8 [12,12] [4.55e-05,0.00629]
MPI_Cart_create:
        Calls     :         12           12 [   0]
        Time      :    0.00143      0.00158 [  68]
        SyncTime  :   2.46e-05     2.91e-05 [ 153]


PAPI Data:
PAPI_TOT_CYC: average = 15447837144.74, max = 15528701710, min = 12149808092
PAPI_L3_TCM: average = 8980659.27, max = 9567080, min = 8494555
PAPI_L2_DCA: average = 150172977.33, max = 206534513, min = 145189127
```

The second report is for the FFTE library.

```
MPI Routine Statistics (FPMPI2 Version 2.4)
Options: FPMPI enabled, Collective sync,
Explanation of data:
Times are the time to perform the operation, e.g., the time for MPI_Send
Average times are the average over all processes, e.g., sum (time on each
process) / number of processes
Min and max values are over all processes
(Data is always average/min/max)
Amount of data is computed in bytes.  For point-to-point operations,
it is the data sent or received.  For collective operations, it is the
data contibuted to the operation.  E.g., for an MPI_Bcast, the amount of
data is the number of bytes provided by the root, counted only at the root.
For synchronizing collective operations, the average, min, and max time
spent synchronizing is shown next.
Calls by message size shows the fraction of calls that sent messages of a
particular size.  The bins are
0 bytes, 1-4 bytes, 5-8 bytes, 9-16, 17-32, 33-64, -128, -256, -512, -1024
 -4K, -8K, -16K, -32K, -64K, -128K, -256K, -512K, -1M, -4M, -8M, -16M,
 -32M, -64M, -128M, -256M, -512M, -1GB, >1GB.
Each bin is represented by a single digit, representing the 10's of percent
```

```
of messages within this bin.  A 0 represents precisely 0, a . (period)
represents more than 0 but less than 10%.  A * represents 100%.
Messages by message size shows similar information, but for the total
message size.

The experimental topology information shows the 1-norm distance that the
longest point-to-point message travelled, by process.

MPI_Pcontrol may be used to control the collection of data.  Use the values
defined in fpmpi.h, such as FPMPI_PROF_COLLSYNC, to control what data is
collected or reported by FPMPI2.

Date:           Thu Sep 10 18:45:41 2020
Processes:      1024
Execute time:   2.817
Timing Stats: [seconds] [min/max]       [min rank/max rank]
  wall-clock: 2.817 sec 2.815902 / 2.861843     211 / 0
       user: 4.839 sec 2.698295 / 5.374994     0 / 31
        sys: 0.1155 sec        0.048117 / 1.125137     734 / 992

Memory Usage Stats (RSS) [min/max KB]:  56176/59924

                Average of sums over all processes
Routine                 Calls       Time Msg Length    %Time by message length
                                                          0.........1.........1.........
                                                                  K         M
MPI_Alltoall        :     148       1.24    3.1e+08 00000000000000000000*00000000
MPI_Bcast           :       2     0.0139     0.0664 0000.*00000000000000000000
MPI_Reduce          :      20   0.000593        160 00*00000000000000000000000000
MPI_Comm_split      :       2     0.0152

Details for each MPI routine
                Average of sums over all processes
                                                % by message length
                                (max over          0.........1.........1........
                                 processes [rank])           K         M
MPI_Alltoall:
        Calls     :        148          148 [   0] 00000000000000000000*00000000
        Time      :       1.24         1.35 [ 164] 00000000000000000000*00000000
        Data Sent :    3.1e+08    310378496 [   0]
        SyncTime  :      0.267        0.412 [ 625] 00000000000000000000*00000000
        By bin    : 1048577-4194304     [148,148] [1.12,1.35] [0.0749,0.412]
MPI_Bcast:
        Calls     :          2            2 [   0] 00005500000000000000000000000
        Time      :     0.0139       0.0339 [ 705] 0000.*00000000000000000000
        Data Sent :     0.0664           68 [   0]
        By bin    : 17-32       [1,1]   [0,1.72e-05]
                  : 33-64       [1,1]   [2.86e-06,0.0339]
MPI_Reduce:
        Calls     :         20           20 [   0] 00*00000000000000000000000000
        Time      :   0.000593      0.00439 [ 384] 00*00000000000000000000000000
        Data Sent :        160          160 [   0]
        By bin    : 5-8 [20,20] [9.54e-05,0.00439]
MPI_Comm_split:
        Calls     :          2            2 [   0]
        Time      :     0.0152       0.0153 [ 352]
        SyncTime  :    1.8e-05     2.41e-05 [ 248]

PAPI Data:
PAPI_TOT_CYC: average = 8738404072.31, max = 8802466864, min = 7089545694
PAPI_L3_TCM: average = 16766188.72, max = 18410863, min = 15327369
PAPI_L2_DCA: average = 60141562.31, max = 75212054, min = 58659609
```

A.6 mpiP

Example report generated using mpiP when running the Klein Gordon solver using 1024 cores of Shaheen II. Due to space constraints, only aggregate time information is provided. Information at a node level is removed. The first report is for the 2decomp&FFT library.

```
@ mpiP
@ Command : /2decomp/summary_run_mpip_1024/./KgmpiP
@ Version                  : 3.4.2
@ MPIP Build date          : Aug 16 2020, 11:18:40
@ Start time               : 2020 09 15 05:29:36
@ Stop time                : 2020 09 15 05:29:41
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 10192
@ Final Output Dir         : .
@ Report generation        : Single collector task

--------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
--------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      Count    COV
Alltoall               11   6.76e+05   14.20   34.99      32768   0.08
Alltoall               10   6.31e+05   13.27   32.70      43008   0.05
Alltoall               20   2.05e+05    4.32   10.64      32768   0.32
Alltoall               28   9.88e+04    2.08    5.12      43008   0.10
Bcast                   7   7.81e+04    1.64    4.04       1024   0.03
Alltoall                6   6.55e+04    1.38    3.39       7168   0.09
Alltoall               29   5.67e+04    1.19    2.94       7168   0.06
Alltoall               15   3.42e+04    0.72    1.77       7168   0.19
Cart_sub                8   2.95e+04    0.62    1.53       7168   0.00
Alltoall                5    2.6e+04    0.55    1.35       7168   0.02
Allreduce              17   1.21e+04    0.25    0.63       7168   0.33
Cart_sub               22   7.86e+03    0.17    0.41       7168   0.01
Reduce                 19   4.05e+03    0.09    0.21       4096   1.09
Cart_sub                2   3.74e+03    0.08    0.19       1024   0.00
Cart_create            26        992    0.02    0.05       7168   0.05
Cart_sub               13        186    0.00    0.01       1024   0.01
Cart_create            21        127    0.00    0.01       1024   0.04
Cart_create            12        116    0.00    0.01       1024   0.05
Reduce                 27       81.6    0.00    0.00       4096   1.31
Cart_create             4       55.7    0.00    0.00       1024   0.10
```

The second report is for the FFTE library.

```
@ mpiP
@ Command : /ffte-7.0/mpi2/summary_run_mpip_1024/./
KgmpiP
@ Version                  : 3.4.2
@ MPIP Build date          : Aug 16 2020, 11:18:40
@ Start time               : 2020 09 15 06:04:51
@ Stop time                : 2020 09 15 06:04:54
@ Timer Used               : PMPI_Wtime
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 12018
@ Final Output Dir         : .
--------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
--------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      Count    COV
Alltoall                3   8.43e+05   26.84   43.15      32768   0.07
Alltoall               11   7.66e+05   24.40   39.23      43008   0.06
Alltoall                6   2.08e+05    6.61   10.63      43008   0.28
Alltoall                5   7.54e+04    2.40    3.86      32768   0.16
Bcast                   9   4.34e+04    1.38    2.22       1024   0.03
Comm_split             14    9.6e+03    0.31    0.49       1024   0.00
Comm_split              2   6.96e+03    0.22    0.36       1024   0.00
Reduce                 10        421    0.01    0.02       4096   3.04
Reduce                  1        288    0.01    0.01       4096   1.36
Reduce                 12        209    0.01    0.01       4096   1.25
Reduce                  8        128    0.00    0.01       4096   1.12
Reduce                 13       82.1    0.00    0.00       4096   0.91
Comm_free               7       26.3    0.00    0.00       1024   0.45
Bcast                   4       12.2    0.00    0.00       1024   1.20
Comm_free              15       5.32    0.00    0.00       1024   0.31
```