Comparative Study of Profiling Tools on Fugaku Supercomputer

Samar A. Aseeri[1],Judit Gimenez[2], Sameer S. Shende[3], Benson K. Muite[4] and David E. Keyes[1]

1) KAUST Extreme Research Computing Center (ECRC) , Saudi Arabia

2) Barcelona Supercomputing Center, Technical University of Catalonia, Spain

3) University of Oregon, USA 4) Kichakato Kizito, Kenya

February 26, 2023

Introduction

This paper examines the overhead of performance tool measurements when profiling Fast Fourier Transform FFT based solvers for the Klein Gordon equation.

Motivation

- This study is evaluating the performance of FFT algorithms on the Fugaku supercomputer, an application with many scientific use cases.
- Performance tools are being used to measure the improvements that a specific computer architecture can bring. This allows scientists and engineers to better understand how their programs will run on the given computer.

Limitation of state-of-art approaches

- Engineers are working to create bigger, advanced computers with specialized hardware known as accelerators. These accelerators can process tasks faster than data can be sent between processes. Therefore, in many cases, simulations must be run for long periods of time to solve moderate-sized problems.
- Scientific programs such as computational chemistry and fluid mechanics use libraries of basic components from the community. New research suggests that Dragonfly networks, a type of computer technology, could have a negative effect on FFT codes, which require a lot of synchronous communication between various components.

Key insights and contributions

- Testing on the Fugaku supercomputer showed that performance of programs using FFT libraries was improved by minimizing interference to communication.
- Profiling and tracing techniques, such as binary instrumentation, were used to measure and analyze program performance. This requires knowledge of the program's structure and probing of memory set aside by the Linux kernel.
- Profiling using tracing has a higher overhead than traditional sampling, but the temporal information it reveals can be extremely helpful when analyzing programs without access to their source code.

Limitations of the proposed approach

The current study focuses on performance measurement using four tools and two FFT libraries, but there are a number of other tools and FFT libraries that can also be used. The profiled codes could be tuned to perform more efficiently on the given architecture. Performance tool overheads are minimized by selecting only a few measurements that may not be the most informative for other applications.

Outline

Outline

The Test Program

The Klein Gordon solver is used as a mini application benchmark for a parallel program that solves a partial differential equation. The FFT algorithm is one of the "Top 10 algorithms of the 20th century" and is simpler to use than alternative algorithms for solving linear equations. To ensure scalability, a two-dimensional pencil decomposition is used by the 2Decomp&FFT and FFTE libraries for the three-dimensional FFT algorithm. The grid size of 768^3 was chosen to allow examination of strong scaling behavior and does not utilize a huge amount of computational time.

The Test Platform

Fugaku is an ARM architecture-based computer at the Riken Center for Computational Science in Kobe, Japan. It is one of the most powerful computers in the world and has been in operation since 2020. The Fugaku supercomputer consists of 158,976 homogeneous nodes consisting of Fujitsu A64FX CPU (48+4 cores) per node and has a performance of 442 PFLOPS measured by the LINPACK benchmark.

The Profiling Tools

- TAU is an open-source set of tools that supports FORTRAN, C++, C, Java, Python, and UPC applications for performance measurement.
- Extrae is an open source tracing tool that can profile MPI, OpenMP, OMPss, pthreads, CUDA, OpenCL, OpenACC, GASPI and Python multiprocessing programs.
- FIPP and FAPP are Fujitsu vendor-provided tools used in this study to profile overall performance based on sampling.

The Parallel FFT Libraries

- 2Decomp&FFT is a FORTRAN software library for parallel applications that utilize the Fast Fourier Transform or need to do neighbor data exchange communications on structured grids. It uses two MPI_ALLTOALLV data exchanges to perform either a forward or backward parallel FFT. The memory requirements of the Klein Gordon solver using this library are dominated by 9 double complex arrays, 4 double complex arrays of combined size $Nx \times Ny \times Nz$ and a temporary double complex array.
- FFTE is a FORTRAN only parallel Fast Fourier Transform library that uses Spiral to generate FFT kernels in C. It uses one MPI_ALLTOALL exchange for a forward or backward parallel FFT. Memory requirements are dominated by 11 double complex arrays of size $Nx \times Ny \times Nz/Np$ allocated in the main program.

Outline

Related Work

- Leu, et al. measured the overhead of performance tools Craypat, FPMPI, mpiP, Scalasca, and TAU on a Cray XC40 platform. The study found that FFTE had better strong scaling behavior than 2Decomp&FFT because it used fewer collective all to all communications.
- Cankur, et al. presents a comparative evaluation of several profiling tools Caliper, HPCToolkit, TAU and Score-P. Its results shows which tool produces the lowest overheads and most meaningful gathered information.
- Hunold et al. conducted a study comparing the overheads of six performance analysis tools on a single node and 32-node Intel Xeon Hydra cluster.
- A PRACE report compared the tools HPCToolkit, Extrae, SCALASCA and the Intel Trace Analyzer and Collector (ITAC).
- This work differs from other works in several ways, such as experimenting on an ARM architecture machine with a torus network.

Outline

Profiling Results

- The FFTE library with Spiral generated C kernels with SVE instructions was the most performant in the current study.
- The Fugaku supercomputer offers a communication mode, the Small-Torus mode, which increases job queuing time and reduces job completion variation time. This mode is used for further experiments reported here.

Profiling Results

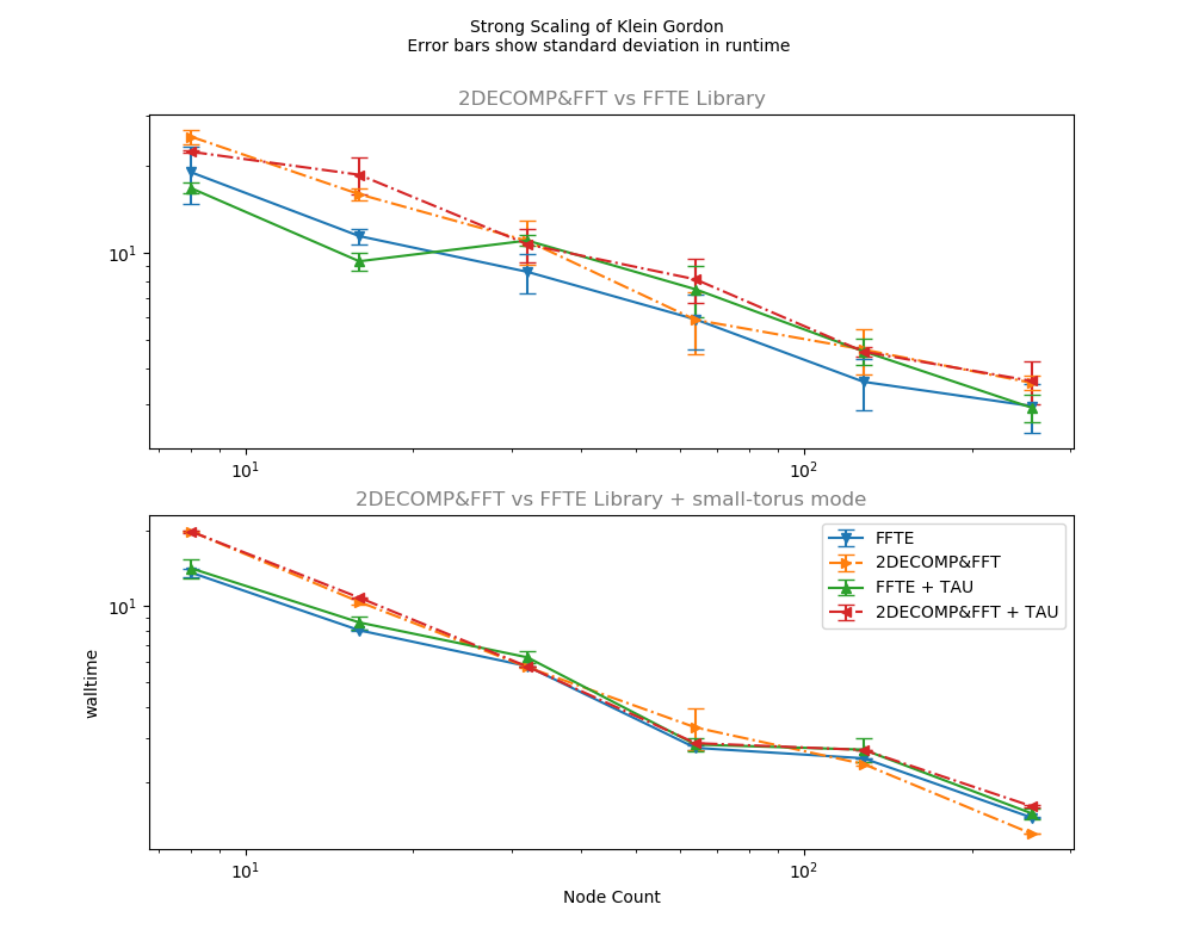
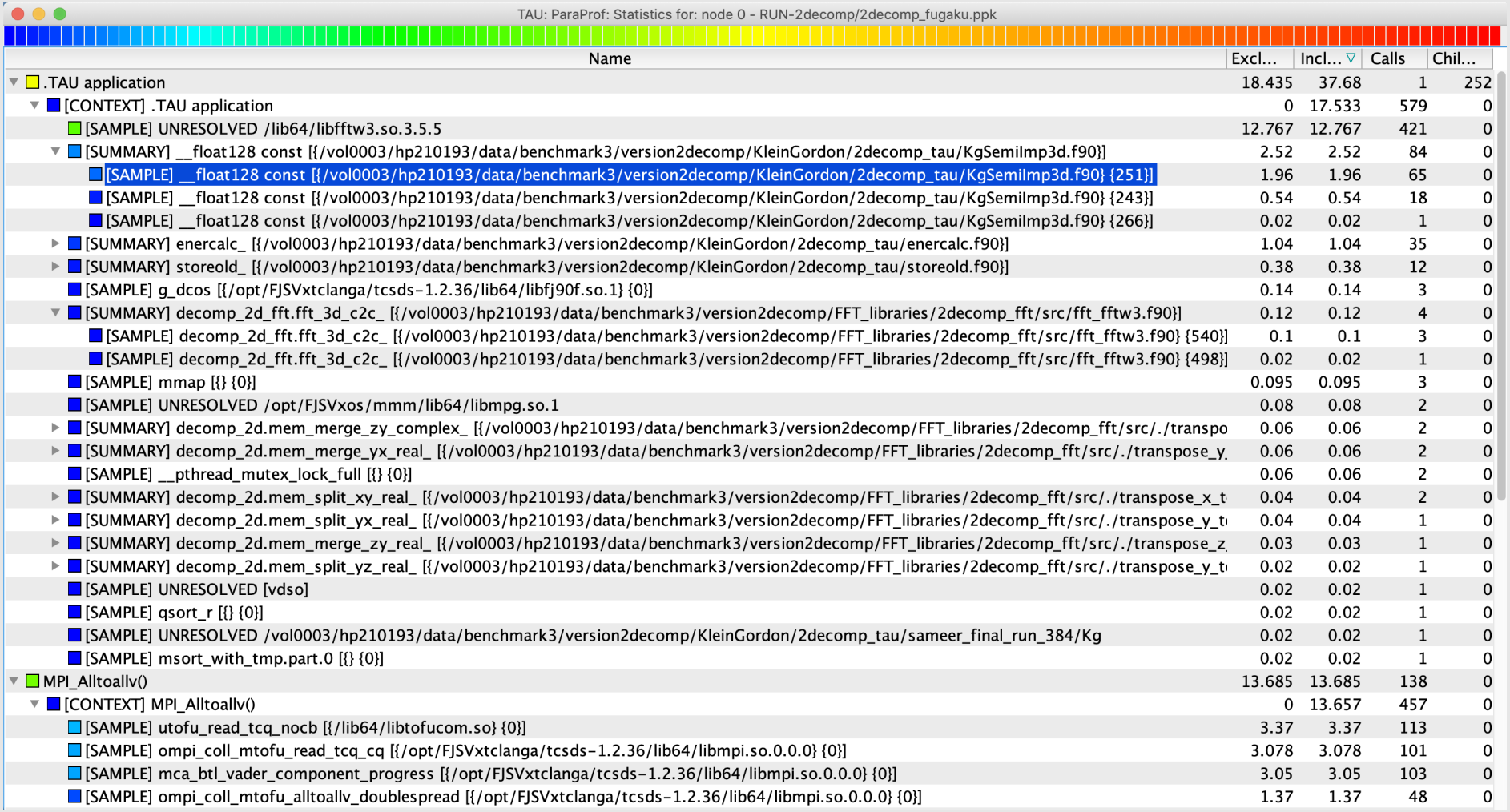


Figure: Test Case Compared with TAU in the bottom plot in Small-Torus mode and without Small-Torus mode in the top plot

TAU Profiling

- TAU is used for sample based profiling of the compiled executable.
- TAU allows a user to observe events inside system libraries and we can see that the routine utofu_read_tcq_nocb in the libtofucom.so library accounted for 3.37 seconds in this run.
- In this manner, TAU can show the precise time spent in MPI calls for each rank, the time spent in external libraries (like FFTW), the time spent inside MPI calls in proprietary libraries - all without modifying the application binary on Fugaku, by simply invoking the application using tau_exec -ebs.

TAU Profiling



Name	Excl...	Incl... ▽	Calls	Chil...
.TAU application	18.435	37.68	1	252
[CONTEXT] .TAU application	0	17.533	579	0
[SAMPLE] UNRESOLVED /lib64/libfftw3.so.3.5.5	12.767	12.767	421	0
[SUMMARY] __float128 const [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/KgSemiImp3d.f90}]	2.52	2.52	84	0
[SAMPLE] __float128 const [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/KgSemiImp3d.f90} {251}]	1.96	1.96	65	0
[SAMPLE] __float128 const [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/KgSemiImp3d.f90} {243}]	0.54	0.54	18	0
[SAMPLE] __float128 const [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/KgSemiImp3d.f90} {266}]	0.02	0.02	1	0
[SUMMARY] enercalc_ [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/enercalc.f90}]	1.04	1.04	35	0
[SUMMARY] storeold_ [{/vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/storeold.f90}]	0.38	0.38	12	0
[SAMPLE] g_dcos [{/opt/FJSVxtclanga/tcsds-1.2.36/lib64/libfj90f.so.1} {0}]	0.14	0.14	3	0
[SUMMARY] decomp_2d_fft.fft_3d_c2c_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/fft_fftw3.f90}]	0.12	0.12	4	0
[SAMPLE] decomp_2d_fft.fft_3d_c2c_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/fft_fftw3.f90} {540}]	0.1	0.1	3	0
[SAMPLE] decomp_2d_fft.fft_3d_c2c_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/fft_fftw3.f90} {498}]	0.02	0.02	1	0
[SAMPLE] mmap [{} {0}]	0.095	0.095	3	0
[SAMPLE] UNRESOLVED /opt/FJSVxos/mmm/lib64/libmpg.so.1	0.08	0.08	2	0
[SUMMARY] decomp_2d.mem_merge_zy_complex_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpo	0.06	0.06	2	0
[SUMMARY] decomp_2d.mem_merge_yx_real_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpose_y	0.06	0.06	2	0
[SAMPLE] __pthread_mutex_lock_full [{} {0}]	0.06	0.06	2	0
[SUMMARY] decomp_2d.mem_split_xy_real_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpose_x_t	0.04	0.04	2	0
[SUMMARY] decomp_2d.mem_split_yx_real_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpose_y_t	0.04	0.04	1	0
[SUMMARY] decomp_2d.mem_merge_zy_real_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpose_z	0.03	0.03	1	0
[SUMMARY] decomp_2d.mem_split_yz_real_ [{/vol0003/hp210193/data/benchmark3/version2decomp/FFT_libraries/2decomp_fft/src/./transpose_y_t	0.02	0.02	1	0
[SAMPLE] UNRESOLVED [vdso]	0.02	0.02	1	0
[SAMPLE] qsort_r [{} {0}]	0.02	0.02	1	0
[SAMPLE] UNRESOLVED /vol0003/hp210193/data/benchmark3/version2decomp/KleinGordon/2decomp_tau/sameer_final_run_384/Kg	0.02	0.02	1	0
[SAMPLE] msort_with_tmp.part.0 [{} {0}]	0.02	0.02	1	0
MPI_Alltoallv()	13.685	13.685	138	0
[CONTEXT] MPI_Alltoallv()	0	13.657	457	0
[SAMPLE] utofu_read_tcq_nocb [{/lib64/libtofucom.so} {0}]	3.37	3.37	113	0
[SAMPLE] ompi_coll_mtofu_read_tcq_cq [{/opt/FJSVxtclanga/tcsds-1.2.36/lib64/libmpi.so.0.0.0} {0}]	3.078	3.078	101	0
[SAMPLE] mca_btl_vader_component_progress [{/opt/FJSVxtclanga/tcsds-1.2.36/lib64/libmpi.so.0.0.0} {0}]	3.05	3.05	103	0
[SAMPLE] ompi_coll_mtofu_alltoallv_doublespread [{/opt/FJSVxtclanga/tcsds-1.2.36/lib64/libmpi.so.0.0.0} {0}]	1.37	1.37	48	0

Figure: TAU's ParaProf shows the thread statistics table for MPI rank 0 for 2decomp&fft run on 384 ranks on Fugaku

TAU Profiling

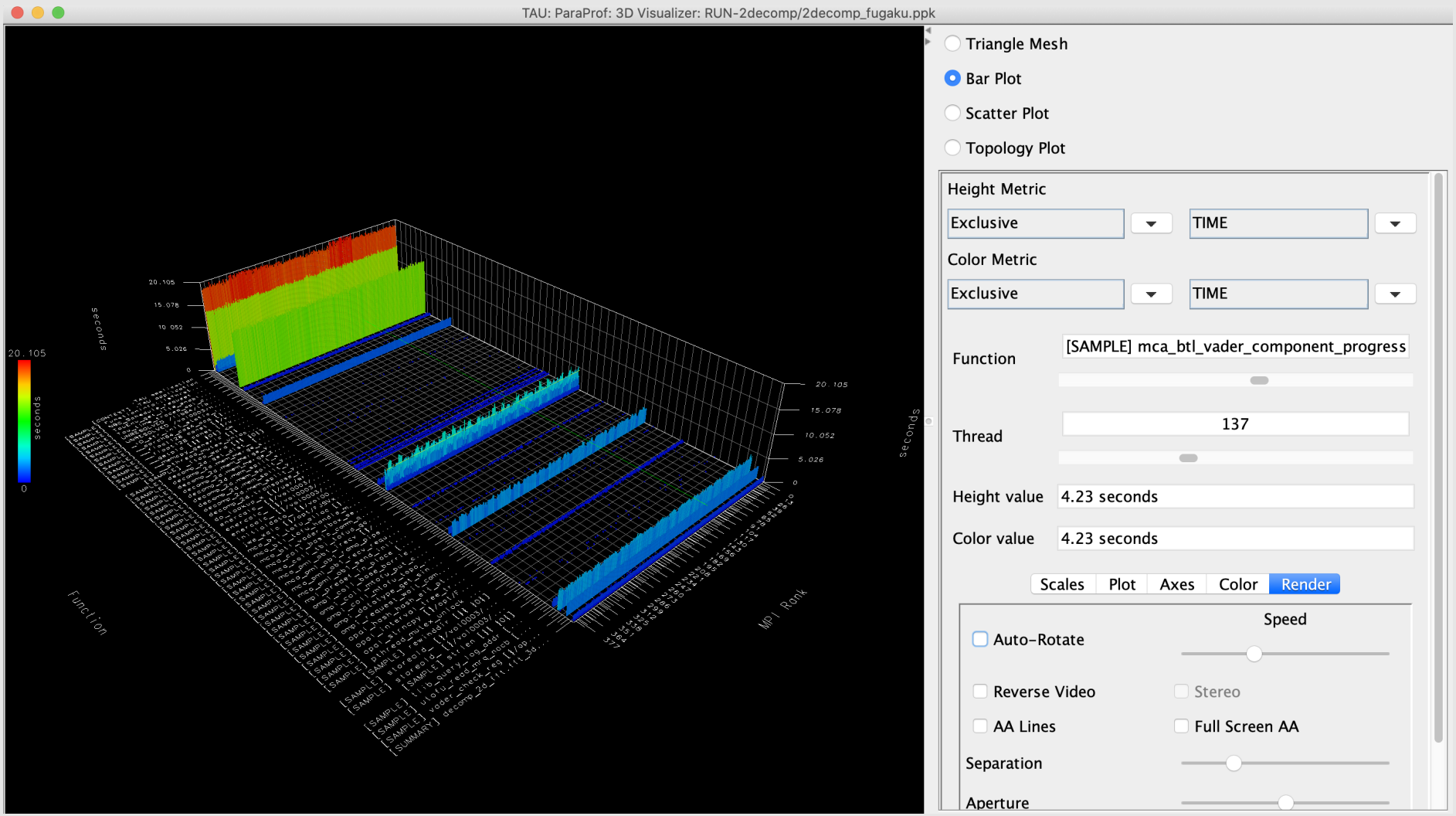
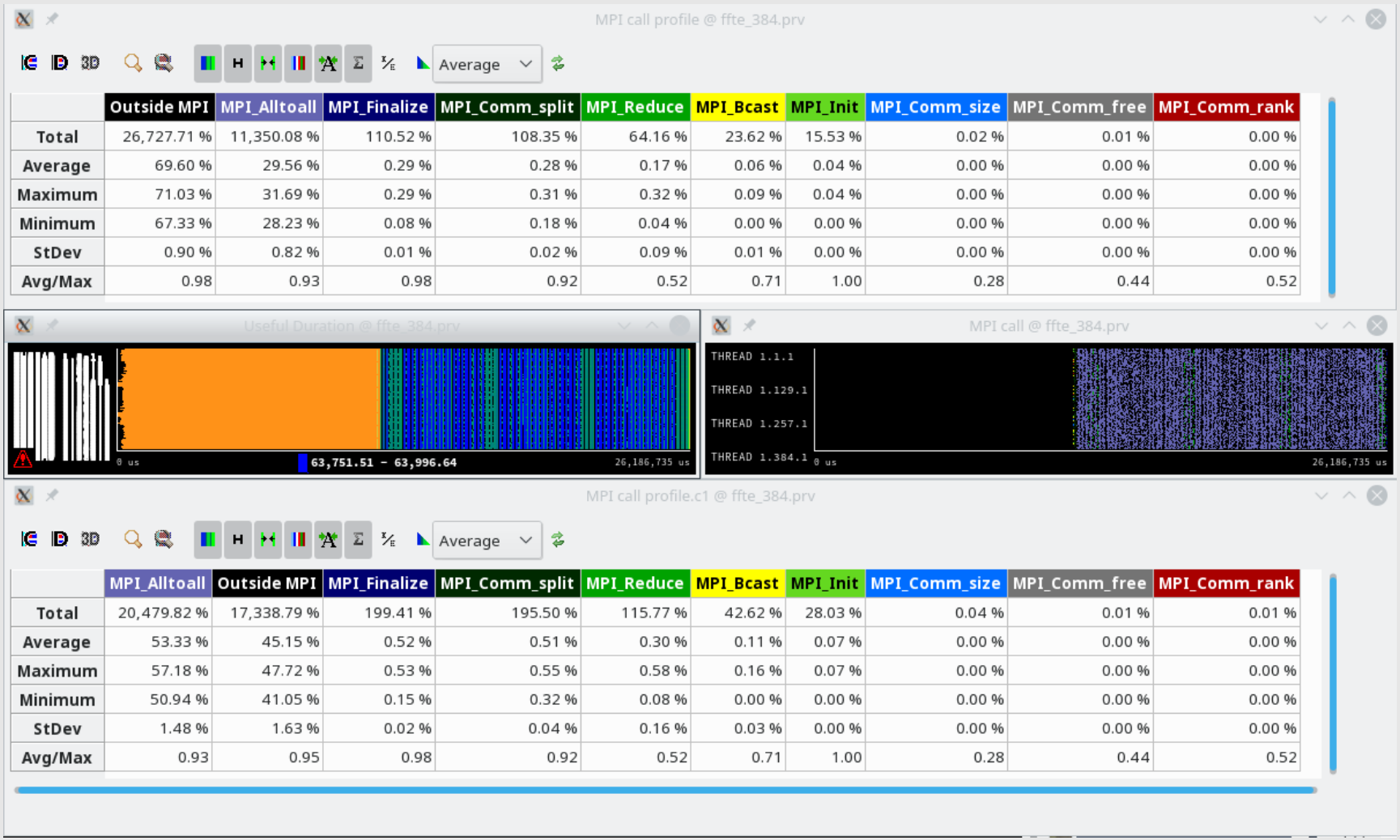


Figure: TAU's ParaProf 3D visualization shows the shape of the overall profile of 2decomp&fft run on 384 ranks on Fugaku.

Extrae Tracing

- Extrae-3.8.3 also collects performance information from binaries that are compiled with debug information.
- The instrumentation tool is configured to capture calls to the MPI library.
- The top MPI profile focus in the full execution reporting that the computation represents almost 70% of the execution time (Average value of Outside MPI) that seems a good percentage.
- With the MPI profile we can also measure that the balance of work between processes that in this case it is quite good (Avg/Max for Outside MPI is 93%) pointing that the problem is related with the communications (an analysis with Dimemas shows it can be related with data transfer).

Extrae Profiling



	Outside MPI	MPI_Alltoall	MPI_Finalize	MPI_Comm_split	MPI_Reduce	MPI_Bcast	MPI_Init	MPI_Comm_size	MPI_Comm_free	MPI_Comm_rank
Total	26,727.71 %	11,350.08 %	110.52 %	108.35 %	64.16 %	23.62 %	15.53 %	0.02 %	0.01 %	0.00 %
Average	69.60 %	29.56 %	0.29 %	0.28 %	0.17 %	0.06 %	0.04 %	0.00 %	0.00 %	0.00 %
Maximum	71.03 %	31.69 %	0.29 %	0.31 %	0.32 %	0.09 %	0.04 %	0.00 %	0.00 %	0.00 %
Minimum	67.33 %	28.23 %	0.08 %	0.18 %	0.04 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	0.90 %	0.82 %	0.01 %	0.02 %	0.09 %	0.01 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.98	0.93	0.98	0.92	0.52	0.71	1.00	0.28	0.44	0.52

	MPI_Alltoall	Outside MPI	MPI_Finalize	MPI_Comm_split	MPI_Reduce	MPI_Bcast	MPI_Init	MPI_Comm_size	MPI_Comm_free	MPI_Comm_rank
Total	20,479.82 %	17,338.79 %	199.41 %	195.50 %	115.77 %	42.62 %	28.03 %	0.04 %	0.01 %	0.01 %
Average	53.33 %	45.15 %	0.52 %	0.51 %	0.30 %	0.11 %	0.07 %	0.00 %	0.00 %	0.00 %
Maximum	57.18 %	47.72 %	0.53 %	0.55 %	0.58 %	0.16 %	0.07 %	0.00 %	0.00 %	0.00 %
Minimum	50.94 %	41.05 %	0.15 %	0.32 %	0.08 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	1.48 %	1.63 %	0.02 %	0.04 %	0.16 %	0.03 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.93	0.95	0.98	0.92	0.52	0.71	1.00	0.28	0.44	0.52

Figure: Extrae MPI instrumentation of FFTE version on Fugaku

Analysis using Extrae to Compare Executions with 384 tasks

- Focusing on the goal to compare the executions with 384 cores, we report a quick analysis that can be done with Paraver for the performance data collected by Extrae.
- As the two versions execute a different number of iterations, we focused the analysis on metrics for the full run.
- Both timelines are on the same scale reflecting that FFTE is much faster than 2Decomp&FFT.

Analysis using Extrae to Compare Executions with 384 tasks

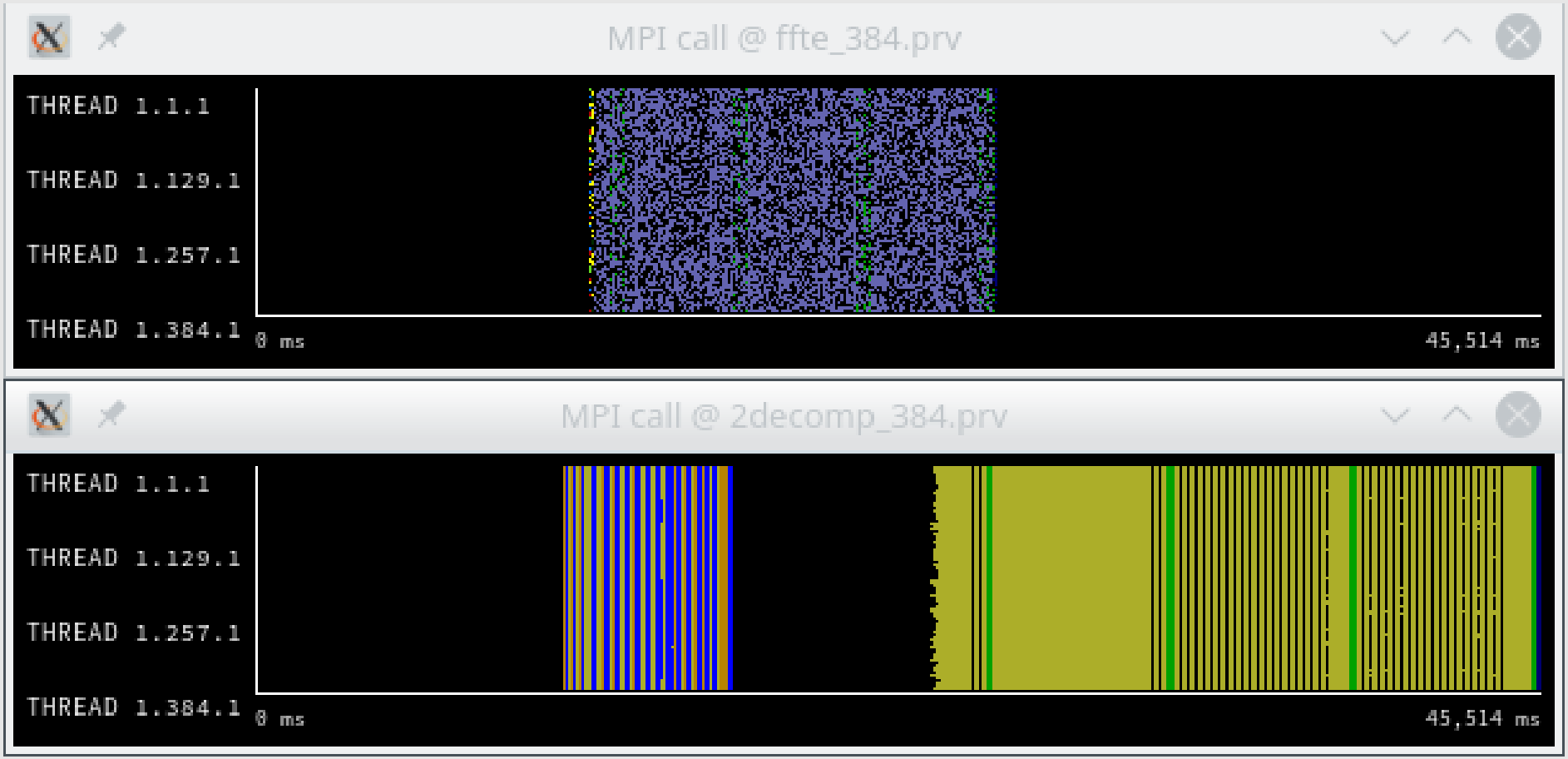


Figure: FFTE has shorter runtime than 2Decomp&FFT, but has less synchronized MPI calls.

FIPP and FAPP Profiling

- FIPP and FAPP have higher overhead.
- Unlike TAU and Extrae, FIPP and FAPP do not support dynamic instrumentation of a binary executable with debug information, the instrumentation for FIPP and FAPP must be done during compilation.
- The source codes need to be annotated with the commands fipp_start/fapp_start and ended with the fipp_stop/fapp_stop.
- There is no special tool to visualize FIPP data on Fugaku.
- FIPP produces a summary file that enables one to determine processor imbalance in the annotated region.

FIPP Profiling

```
Time statistics

        Elapsed(s)        User(s)      System(s)
   ------------------------------------------
          38.5575             --             --   Application
   ------------------------------------------
          38.5575             --             --   Process      5
          38.5567             --             --   Process      7
          38.5563             --             --   Process      4
          38.5561             --             --   Process      3
          38.5557             --             --   Process      0
          38.5540             --             --   Process      2
          38.5540             --             --   Process      1
          38.5419             --             --   Process      6
          38.5378             --             --   Process      8
          38.5216             --             --   Process     12
          38.5183             --             --   Process     11
          38.5058             --             --   Process     13
          38.5055             --             --   Process      9
          38.4910             --             --   Process     10
          38.4784             --             --   Process     14
          38.4605             --             --   Process     16

Performance monitor event:statistics
```

Figure: Head of FIPP Profiling data for 2decomp&fft

FAPP Profiling

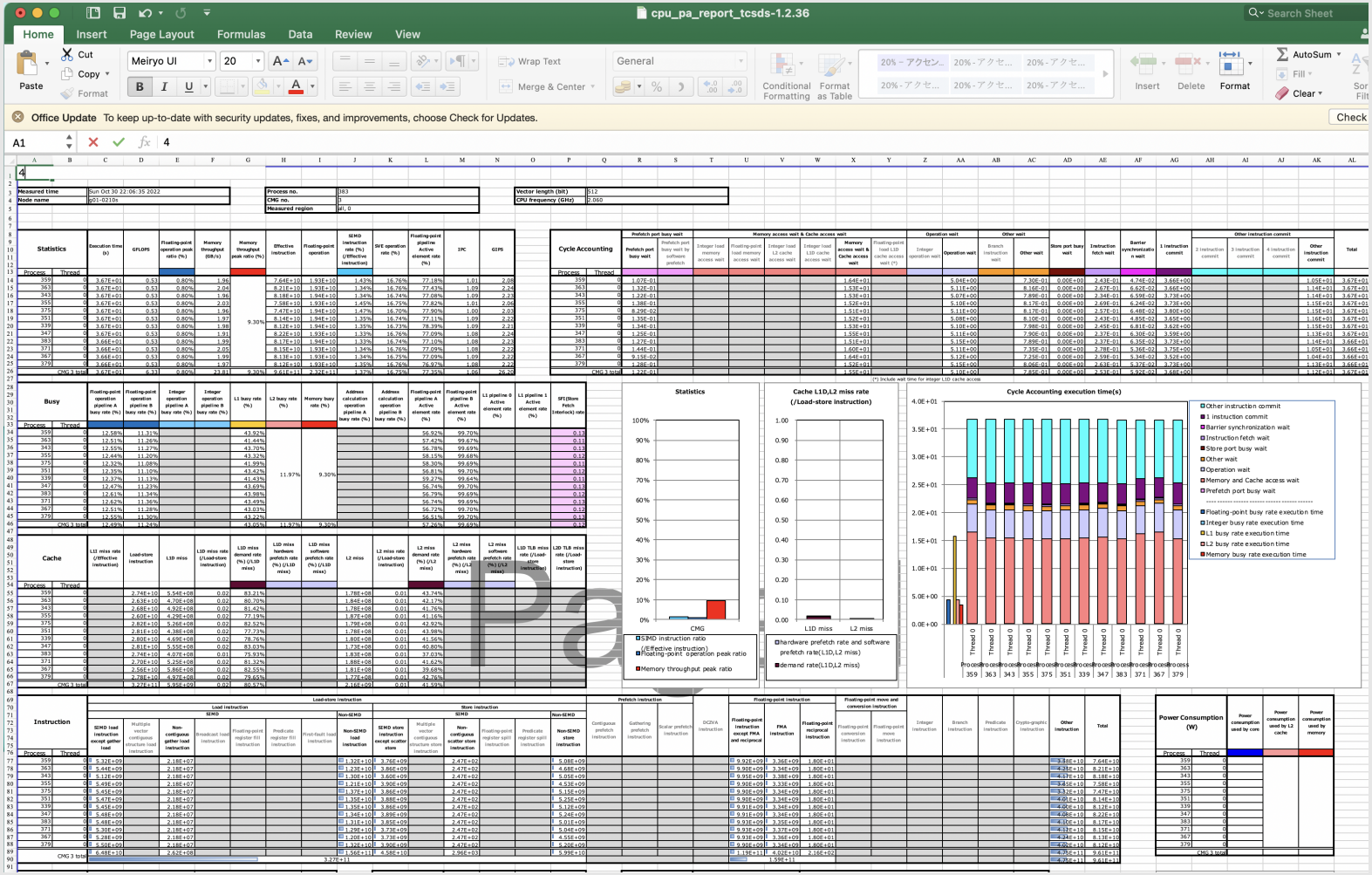


Figure: FAPP CPU Performance Analysis Report for the 2Decomp&FFT executed on 384 tasks for reading the Pa1,Pa2,Pa3,Pa4 and Pa5 reports to obtain the Brief Report

Outline

Results Evaluation

- We have run three test cases of 284^3 768^3 and 1536^3 grid sizes, with GNU and with Fujitsu compilers but Fujitsu was superior as the three run average gave 12.56851788998271 while the GNU average gave 38.865136866666641 for the 7683 case.
- Consequently, for our experiments we use default Fugaku environment "lang/tcsds- 1.2.36(default)" to compile our MPI code.
- For this, we sought to consider three metrics: run-time overhead, memory usage, size of generated data.

Tool Overhead Comparisons

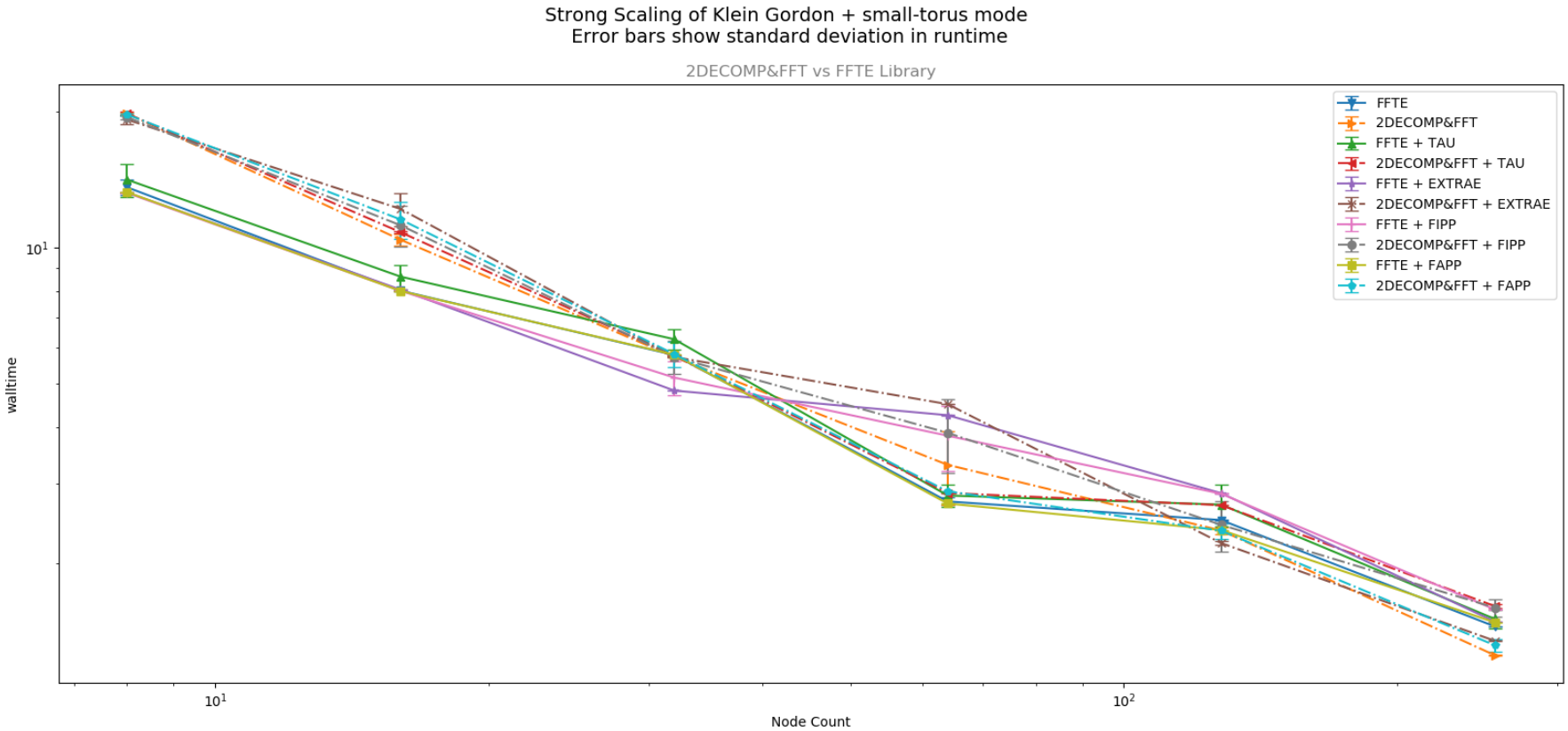


Figure: Performance Tools Comparison with Small-Torus mode

Overhead Memory Measurements

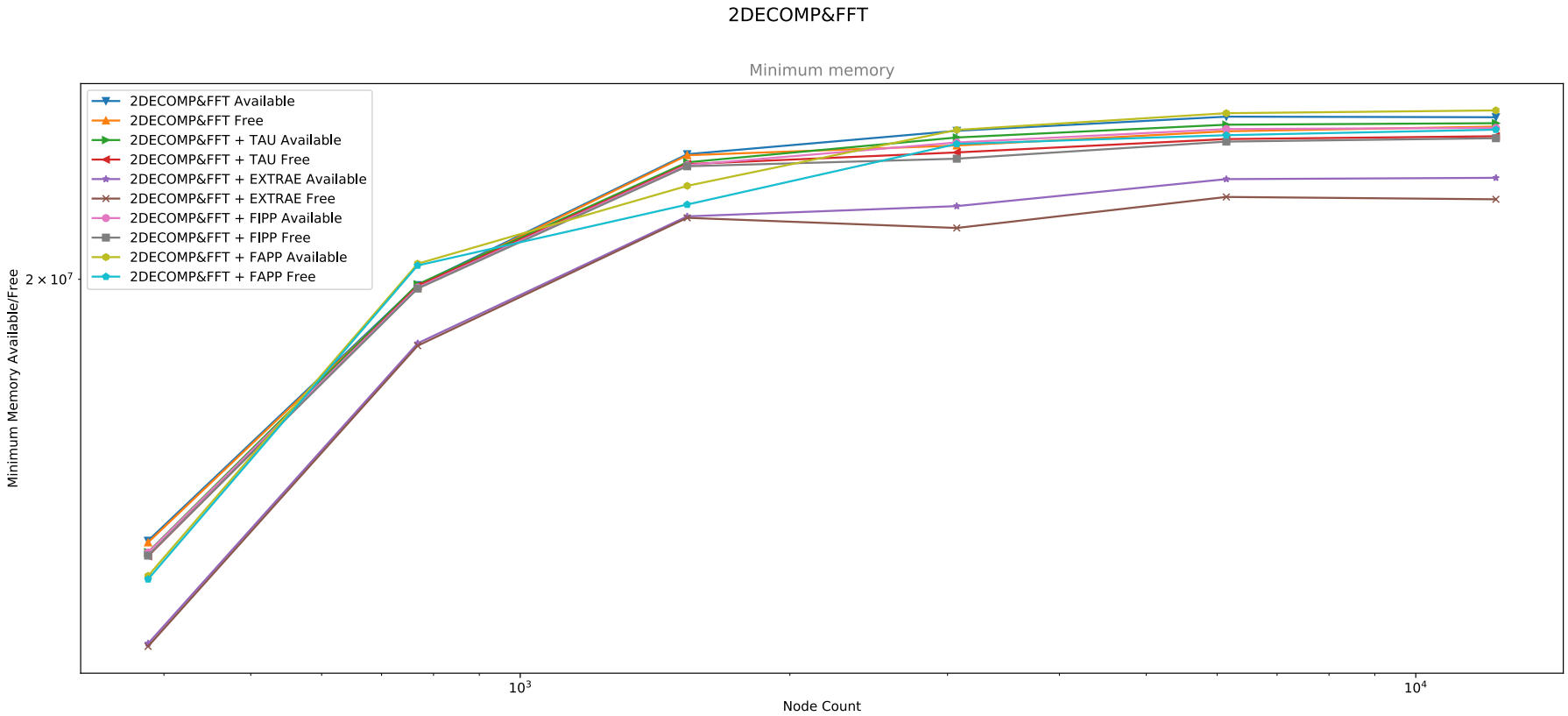


Figure: Performance Tools memory-check comparison of all cases for 2Decomp&FFT Library

Overhead Memory Measurements

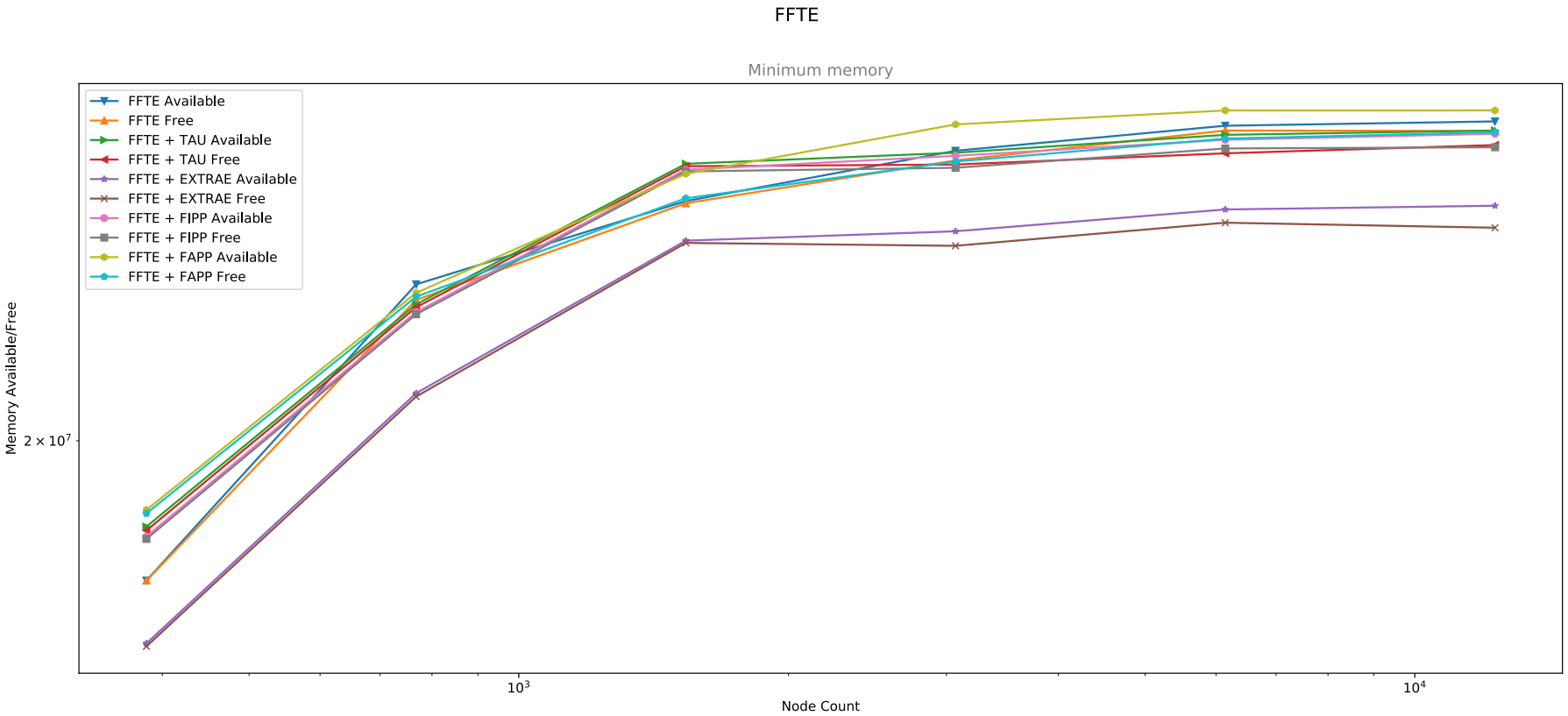


Figure: Performance Tools memory-check comparison of all cases for FFTE Library

Data Storage Comparisons

- Size of data generated from running the executable with the examined profiling tools can be found in Tables in the upcoming slides.
- TAU supports compression and normalization of profile files using the paraprof - -pack command to generate files in TAU's packed profile format.
- The ppk format is very efficient in reducing the size of the profile dataset and can result in several orders of magnitude reduction in data sizes.

Data Storage Comparisons

Table: Table comparing data size generated for parallel FFT for a 768^3 problem size (30 forward and backward FFTs) using 2Decomp&FFT library executed with the Extrae, TAU and FIPP tools on Fugaku.

Nodes	Cores	2Decomp&FFT	2Decomp&FFT Extrae	2Decomp&FFT TAU	2Decomp&FFT FIPP	2Decomp&FFT FAPP
8	384	623.23K	79.18M	11.29M	9.54M	23.00M
16	768	629.42K	281.12M	22.80M	17.42M	11.80M
32	1536	642.29K	730.95M	37.13M	28.94M	6.20M
64	3072	668.33K	1.42G	64.37M	55.17M	3.40M
128	6144	720.44K	2.82G	115.35M	99.73M	2.00M
256	12288	826.97K	5.59G	219.98M	197.52M	1.30M

Data Storage Comparisons

Table: Table comparing data size generated for parallel FFT for a 768^3 problem size (30 forward and backward FFTs) using FFTE library with executed with the Extrae, TAU and FIPP tools on Fugaku.

Nodes	Cores	FFTE	FFTE + Extrae	FFTE + TAU	FFTE + FIPP	FFTE + FAPP
8	384	385.13K	20.91M	20.31M	3.11M	17.15M
16	768	391.28K	42.11M	30.32M	5.12M	8.76M
32	1536	404.13K	91.72M	42.91M	8.54M	4.56M
64	3072	430.24K	188.33M	65.09M	15.20M	2.47M
128	6144	482.45K	439.29M	101.76M	28.28M	1.42M
256	12288	589.08K	920.25M	172.14M	57.52M	916.75K

Data Storage Comparisons

Table: Table comparing data sizes for TAU's packed profile format (ppk) for 2Decomp&FFT and FFTE library generated for parallel FFT for a 768^3 problem size (30 forward and backward FFTs) on Fugaku.

Nodes	Cores	2Decomp&FFT + TAU (ppk)	FFTE + TAU (ppk)
8	384	461K	574K
16	768	840K	862K
32	1536	1.3M	1.2M
64	3072	2.3M	1.8M
128	6144	3.9M	2.9M
256	12288	7.0M	4.5M

Outline

Summary & Conclusions

- This study examines the overhead of performance tool measurements when profiling Fast Fourier Transform (FFT) based solvers for the Klein Gordon equation. The tools examined are TAU and Extrae in addition to the Fujitsu provided Fugaku profiling tools FAPP and FIPP.
- Performance measurement is an important part of improving the efficiency of parallel programs for a specific architecture.
- Fugaku's small torus mode is shown to significantly improve performance of communication intensive codes by ensuring communication from other jobs is not intermingled with communication of the job.
- This paper examines the performance of programs that heavily utilize the FFT, a communication intensive algorithm, by minimizing communication interference from other running jobs.

Summary & Conclusions

- It shows that profiling and tracing using binary instrumentation can capture many aspects of a program, but knowledge of the program structure is still required to interpret the resulting performance measurements.
- The paper focuses on performance measurement using four tools and two FFT libraries, but there are other tools and FFT libraries that can also be used.
- The paper also presents memory overhead measurements and profiling result storage requirements.
- The Klein Gordon solver is used as a mini application benchmark for a parallel program that solves a partial differential equation.

Outline

Further Work

- When McKernel is available on Fugaku, it is important to measure the effect of OS noise reduction on performance of bulk synchronous programs.
- Further work involves profiling other parallel FFT libraries and additional profiling and tracing tools, examining other exascale hardware architectures, and examining other numerical methods to aid in performance prediction and optimal matching of algorithms to hardware architecture.
- Finally, GPU-based FFT libraries should be analyzed for their potential to improve performance on exascale systems.

Acknowledgments