Toward Automated Application Profiling on Cray Systems

Charlene Yang, Brian Friesen, Thorsten Kurth, Brandon Cook
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, US
Email: {cjyang, bfriesen, tkurth, bgcook}@lbl.gov

Samuel Williams
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720, US
Email: swwilliams@lbl.gov

***Abstract*—Application performance data can be used by HPC users to optimize their code and prioritize their development efforts, and by HPC facilities to better understand their user base and guide their future procurements. This paper presents an exploration of six commonly used profiling tools in order to assess their ability to enable automated passive performance data collection on Cray systems. Each tool is benchmarked with three applications with distinct performance characteristics, to collect five pre-selected metrics such as the total number of floating-point operations and memory bandwidth. Results are then used to evaluate the tools' usability, overhead to run, amount of actionable information that they can provide, and accuracy of the information provided.**

***Keywords*-automated profiling; performance tools; Cray; HPC;**

I. Introduction

Profiling tools such as CrayPat [1] and LIKWID [2] provide additional insights to the code that developers can not obtain by simply inspecting the code. Thus it is important for HPC developers to collect performance data to identify the bottleneck and prioritize their efforts on more profitable optimizations. It is equally important for HPC centers to collect mass performance data on all users and applications in order to better understand their user base and make more informed decisions in their next procurement. However, these tools vary, in terms of their overhead, how easy to run, the amount of information they can provide and the accuracy of the information provided. This paper presents a survey of a set of commonly used profiling tools in the HPC area, to assess their ability to enable low-overhead, high-accuracy and easy-to-automate performance data collection on Cray systems. Each tool is benchmarked on three applications with distinctive performance characteristics, to collect five pre-selected performance metrics.

II. Tools, Metrics and Applications

A. Tools

The tools being assessed are, CrayPat (`perftools-lite`) [1], LIKWID from Regional Computing Center Erlangen in Germany [2], IPM from Lawrence Berkeley National Laboratory [3], Intel VTune Amplifier [4], Intel SDE Emulator [5], and the `perf` tool from the Linux Kernel [6].

The mechanisms behind these tools are very different. Some are based on sampling, some on performance counters; some require code instrumentation and some can run by simply appending a wrapper to the native run command.

B. Metrics and Tool Qualities

To evaluate these tools, five metrics are selected for these tools to collect.

1) Runtime, to evaluate the tools' overhead relative to codes' 'native' runtime; it is equally important to HPC users as well as HPC facilities.
2) GFLOP's and GFLOP/s, with GFLOP's representing the count of giga floating-point operations executed and GFLOP/s representing the rate of GFLOP's executed per second. This metric is to evaluate tools' ability to capture codes' floating-point operations; it could be a more important metric to HPC facilities than to HPC users as it shows how well their supercomputer is being utilized.
3) Bandwidth on different levels such as DRAM, LLC, L2 and L1. This metric is to assess whether tools are able to accurately report the data movement within the memory hierarchy; it is an indicator to whether a code is bandwidth-bound or not; also a measurement of how much a system's memory bandwidth is utilized by users, on a particular memory/cache level.
4) Memory high watermark. This metric is to evaluate tools' ability to capture the memory footprint of an application, helping users to avoid OOM (Out Of Memory) error, and helping HPC centers to decide on their next machine's memory sizes.
5) Vectorization efficiency. This is to assess whether tools can accurately report on how well a code is vectorized, helping code developers decide if efforts need to be spent on vectoring the code, and also helping HPC centers decide if architectures with more VPU's (Vector Processing Units) are beneficial to their user base.

Ideally, profiling tools should strive for the following four qualities and we assess the six abovementioned tools against them in this paper.

1) Usability: little to no user intervention to activate;

2) Overhead: negligible runtime overhead to the application's 'native' performance;
3) Actionability: produces information that can direct developers toward their next-step optimization or guide the center with their procurement;
4) Accuracy: produces information as close to the real-time events as possible.

C. Applications

Each tool is run on three applications, HPGMG [7], Nyx [8] and Tiramisu[9], to collect the aforementioned five metrics. These applications are selected so that they can highlight the difference between tools when reacting to different performance characteristics. As shown in Table I, these applications are distinct in scale, parallelization framework, and domain problems solved. A cumulative analysis of all three applications should provide a comprehensive view of each tool's merits and drawbacks.

HPGMG: HPGMG [7], [10] is a geometric multigrid benchmark designed to proxy the multigrid solves found in block structured AMR applications. HPGMG is implemented in C with MPI and OpenMP parallelization and has shown scalability to 8.5 million cores. HPGMG uses a 4^{th} order, variable-coefficient Laplacian. As such, it is both compute intensive and demanding of vectorization and cache locality.

Nyx: Nyx [8], [11] is a cosmological simulation code which models the evolution of the universe, in order to identify the initial conditions that give rise to the large-scale structures observed today. Nyx uses an N-body, particle-mesh method for tracking dark matter, and an Eulerian scheme on structured grids (implemented in AMReX) to track intergalactic gas (ionized states of hydrogen and helium). The dark matter and gas interact gravitationally, and so their interaction is determined by the solution to the Poisson equation at each time step. Nyx solves this equation using a geometric multigrid gravity solver (with a constant coefficient Laplacian operator) similar to HPGMG. However, unlike HPGMG, this solver is 2^{nd} order and constant coefficient (less compute intensive). Ionization of hydrogen and helium is effected through an ODE solver.

Tiramisu: Tiramisu [9], [12] is a deep learning model based on Convolutional Neural Network (CNN), implemented using the Google TensorFlow framework [13], and used for detection of extreme weathers in climate science. The code is a mix of Python and C. The Python part of code specifies dataflow graph, manages workloads and spawns appropriate number of processes as needed, while the C part of the code calls highly optimized libraries such as MKL-DNN [14] and cuDNN [15] to do the heavy-lifting work. MKL-DNN contains dynamically assembled code by the Xbyak JIT assembler [16] and also spawns variable number of threads based on the available resources. So whether the profiler can capture the activity from the different parts of the code, or the 'secretly' spawned threads, is going to be a challenge.

Table I
PERFORMANCE CHARACTERISTICS OF THREE APPLICATIONS

	HPGMG	Nyx	Tiramisu
Scale	Kernel	Full Application	Full Application
Lines of Code	∼20K	∼2M	∼3M
Language	C	C/C++, Fortran	Python, C/C++
Parallelism	MPI, OMP, CUDA	MPI, OMP	Python, MKL
Domain	HPC	HPC	Deep Learning
	PDE solvers	PDE/ODE solvers	TensorFlow
	Geometric Multigrid	Mesh Refinement	Image Processing

III. BENCHMARKING AND RESULTS

A. Configuration

All benchmarking is done on Cori at NERSC (National Energy Research Scientific Center), LBNL (Lawrence Berkeley National Laboratory). Applications are compiled with Cray wrappers for Intel 18.0.1.163 compilers and linked dynamically, except for Nyx when it's profiled with IPM. All codes are ran on the KNL partition. Both HPGMG and Nyx are run with 8 MPI ranks and 8 OpenMP threads per rank; Tiramisu is run with 2 Python processes and 33 threads per process. A fixed nominal CPU frequency 1.401 GHz is used for all benchmarks in order to avoid timing variation caused by frequency difference across cores.

B. Results

How Results Are Presented: Since we have three dimensions to the benchmark results (benchmarks, applications and tools), figures in this subsection are arranged so that each benchmark is presented in a different figure, each application is represented by a different color, and each tool is by a different marker. For example, in Fig. 1, red datapoints represent results from HPGMG for all tools and blue represent those from Nyx; diamond markers represent results from IPM for all applications while circles represent those from LIKWID. Within the same figure, there are multiple metric groups and plots are clustered based on them, for example, the two metric groups in Fig. 1, 'Overall Runtime' and 'Runtime Per Solve'.

The disparity of markers along each vertical line in these figures shows the difference between tools for the same metric and the same application. For example, in Fig. 2, LIKWID and SDE produce similar results for `Scalar GFLOP's` for all applications, but different results for `Vector GFLOP's`. Also, placing results for different applications together in the same metric group highlights how differently tools react to different performance characteristics of applications. For example, for `Vector GFLOP's`, LIKWID and SDE produce similar results for HPGMG but not for Nyx. The reason is that HPGMG is well optimized for vectorization and cache locality while Nyx is not and it has many register-to-register instructions. LIKWID counts

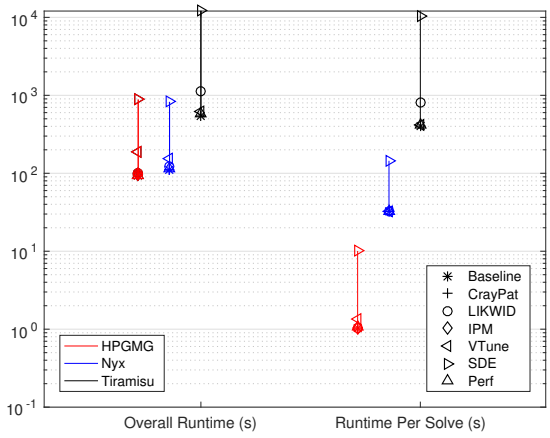

Figure 1. Comparison of tools on overall runtime and runtime per solve. Timing for each solve is taken by codes' internal timers.

both arithmetic floating-point vector instructions, like SDE does, and some non-arithmetic vector instructions such as `vextract` and `vpbroadcast`, resulting in its overcounting in Nyx's case.

Due to unavailability of pre-defined performance groups in some tools, there may be datapoints missing. For example, in Fig. 3, there is only one set of markers for LIKWID in the L2 data movement metric group. Also, baseline datapoints may be absent too. For example, the baseline data for GFLOP's for HPGMG and Nyx is unavailable in Fig. 2 because it is very difficult to calculate the number of floating-point operations analytically.

Metric 1: Runtime*:* This benchmark evaluates tools' overhead both in overall runtime and runtime per internal step. The purpose of this benchmark is to assess whether tools only have a fixed amount of startup/finalization overhead, or it slows down the execution of the code in every step. The internal step or 'per solve' runtime as shown in Fig. 1 is defined differently for these three applications. For HPGMG, it is the solve of a Poisson problem for a multigrid with 64 boxes, each of size 128^3; for Nyx, it is a time step in a cosmological simulation of the Lyman-alpha forest in a Lambda-CDM model of the universe [17], [18]; for Tiramisu, it is the training of the neural network for 10 climate images, each of size $1152 * 768 * 16$, in order to separate pixels that are related to hurricane, atmospheric river and background noise, from each image.

Fig. 1 shows that all tools closely gather around the baseline datapoint except for SDE and VTune. More details are shown in Table II, where CrayPat, LIKWID, IPM and Perf exhibit a very low overhead (less than 10%) for all applications but SDE and VTune present a much higher cost. With the complex workflow Tiramisu, SDE becomes prohibitively expensive with more than 20× overhead.

Table II
OVERHEAD OF TOOLS (NORMALIZED TO BASELINE RUNTIME)

		CrayPat	LIKWID	IPM	VTune	SDE	Perf
HPGMG	Overall	1.093	1.091	1.043	2.057	9.560	1.001
	Per Solve	1.023	1.011	1.014	1.306	9.84	1.054
Nyx	Overall	1.004	1.109	0.975	1.403	7.499	1.043
	Per Solve	1.012	1.001	0.915	1.016	4.464	1.015
Tiramisu	Overall	-	2.117	-	1.135	22.281	1.078
	Per Solve	-	2	-	1.012	25.683	1.027

Tiramisu is not benchmarked with CrayPat or IPM because both tools require recompilation of the code for instrumentation purpose and it is a great challenge to add custom linking flags or environment variables to the Bazel module in order to compile TensorFlow. Since the paper is focused on easy-to-use profilers, these two tools are omitted from this benchmark.

Metric 2: GFLOP's and GFLOP/s*:* GFLOP's is defined as the number of giga floating-point operations executed in the code and GFLOP/s as the number of GFLOP's executed per second. Results in Fig. 2 are only for LIKWID, SDE and Perf because VTune does not have a pre-defined performance group for FLOP's measurement, and CrayPat and IPM only report FLOP's performance when the architecture provides hardware counters that directly measure the number of FLOP's executed and/or retired, which is not the case on KNL. Despite the missing direct performance counter, LIKWID and Perf make some assumptions about the instruction mix, and use two related counters, `UOPS_RETIRED.SCALAR_SIMD` and `UOPS_RETIRED.PACKED_SIMD`, to estimate the FLOP's performance of the code. Different from the abovementioned tools, SDE works by counting the number of floating-point instructions in the assembly code, thus not affected by the missing status of direct FLOP's counters on KNL.

It is very difficult to obtain the canonical FLOP's from the code, therefore no baseline data is presented in Fig. 2. However, Tiramisu is an exception, because its computation is concentrated in the training of the convolutional and deconvolutional layers of the network and the algorithm is tractable enough for us to derive a FLOP's estimate for reference.

As shown in Fig. 2, LIKWID, SDE and Perf produced similar results for `GFLOP's`, `Scalar GFLOP's` and `Vector GFLOP's`, for HPGMG. Due to timing differences, i.e. overhead differences, the rate, `GFLOP/s`, is different across the tools. For Nyx and Tiramisu, close results are reported for `Scalar GFLOP's` but there is some disparity in `Vector GFLOP's`. The reason for this is that Nyx is not well optimized and executes a lot of memory instructions such as `extract` and `broadcast` to rearrange data before computation. These memory instructions can be in vector form, which will trigger the performance counter, `UOPS_RETIRED.PACKED_SIMD`, which

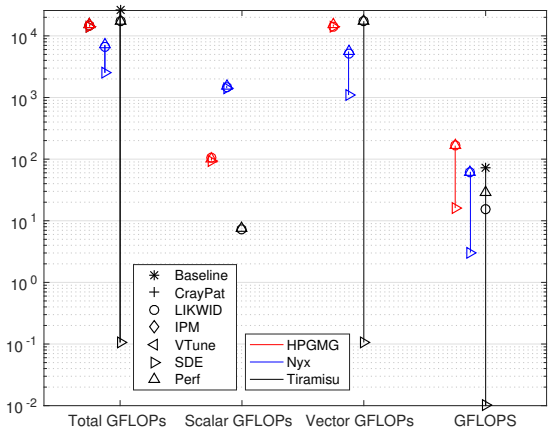

Figure 2. Comparison of tools on GFLOP's and GFLOP/s measurement. `Total GFLOP's` is the sum of `Scalar GFLOP's` and `Vector GFLOP's` and the rate `GFLOP/s` is the `Total GFLOP's` over each tool's respective runtime. Note, SDE's estimate of Tiramisu's `Total GFLOP's` and `GFLOP/s` is heavily skewed by SDE's inability to introspect into precompiled MKL routines.

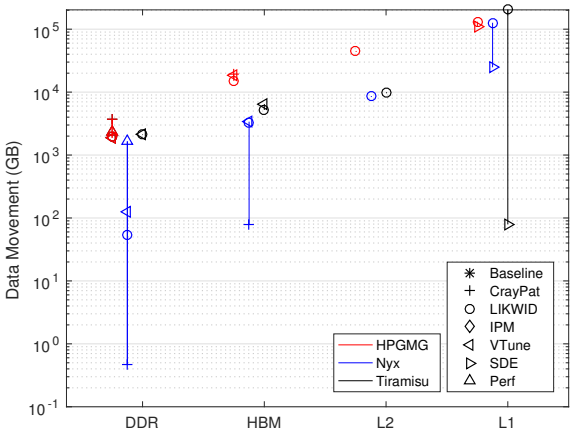

Figure 3. Comparison of tools on data movement measurement.

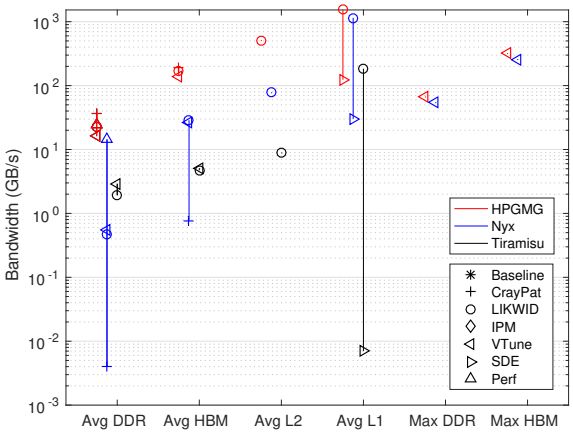

Figure 4. Comparison of tools on bandwidth measurement. Average bandwidth is calculated using the data movement results in Fig. 3 and the overall runtime in Fig. 1. Maximum bandwidth is reported by tools as the peak instantaneous bandwidth.

both LIKWID and Perf use, resulting in them overcounting the arithmetic vector instructions. HPGMG does not have this problem because it is highly optimized for cache locality and does not have many such memory instructions. Tiramisu has very different `Vector GFLOP's` from LIKWID, SDE and Perf is because SDE is not able to capture the C/C++ code executed by its Python wrappers (hence missing those FLOP's generated) while the others can.

***Metric 3: Memory Bandwidth**:* This benchmark assesses tools' ability to capture average and maximum memory bandwidth drawn by an application. It requires tools to measure both overall data movement (in order to produce the average bandwidth) and instantaneous bandwidth (in order to obtain the maximum bandwidth), on all levels of the memory/cache hierarchy, namely on KNL, DDR, HBM, L2 and L1. Fig. 3 shows the data movement measured by all possible tools for all four levels, and Fig. 4 shows the average bandwidth measured for all four levels, and maximum bandwidth measured only on DDR and HBM level.

CrayPat and VTune can report data movement on both DDR and HBM levels, with VTune also being able to report maximum bandwidth on these two levels. LIKWID measures data movement on all four levels but does not report maximum bandwidth on any level. SDE produces L1 data movement, and Perf is only experimented with on the DDR level (Perf does not have a pre-defined performance group for data movement measurement but we passed on the hexadecimal codes for certain performance counters to take this measurement and so far it's only the DDR level that we have experimented with).

For HPGMG, CrayPat, LIKWID and VTune produce very similar results for DDR and HBM data movement in Fig. 3 (hence their respective average bandwidth in Fig. 4). LIKWID and SDE produce L2 and L1 data movement, which is on an increasing trajectory (see Fig. 3) as the memory/cache level gets closer to the CPU. The maximum bandwidth reported by VTune in Fig. 4 on both DDR and HBM levels also falls within a reasonable range between the average bandwidth and hardware limit, on each respective level. For Nyx, CrayPat's reporting is much lower than LIKWID's, VTune's or Perf's, on DDR and HBM data movement in Fig. 3. The reason for this is unclear as the formula behind CrayPat's report is not published. For Tiramisu, LIKWID and VTune are able to produce consistent measurement on DDR and HBM data movement (hence average bandwidth),

Table III
MEMORY HIGH WATERMARK (GB)

	CrayPat	LIKWID	IPM	VTune	SDE	Perf
HPGMG	0.255	-	11.93	-	-	-
Nyx	2.76	-	13.52	-	-	-
Tiramisu	-	-	-	-	-	-

Table IV
VECTORIZATION EFFICIENCY (RANGE OF 0 TO 1)

	CrayPat	LIKWID	IPM	VTune	SDE	Perf
HPGMG	-	0.945	-	-	0.949	0.949
Nyx	-	0.309	-	-	0.088	0.311
Tiramisu	-	0.993	-	-	0.999	0.993

and reasonable L2 and L1 data movement. However, SDE is not able to estimate Tiramisu's L1 data movement due to its inability to introspect precompiled binaries and count loads and stores.

***Metric 4: Memory High Watermark**:* This benchmark is to assess whether tools are able to capture the maximum memory footprint of an application. As shown in Table III, only CrayPat and IPM are and their results are very different. The reason is that CrayPat reports using the information in $/proc/self/numa\_maps$ files which are captured near the end of the program, while IPM reports using the information in $/proc/self/status$. Take Nyx as an example. IPM has proved to be more accurate than CrayPat since Nyx tracks its memory allocation for grid variable (but not temporary arrays) and its reporting of 12.3GB can be used as a lower bound.

***Metric 5: Vectorization Efficiency**:* There are several definitions of vectorization efficiency and this paper defines it as the ratio of packed arithmetic floating-point instructions to the total number of arithmetic floating-point instructions. SDE works on the instruction level while LIKWID and Perf work on the micro-op level. Since not all instructions are implemented with one micro-op (most are), there could be some discrepancy between SDE, and LIKWID and Perf. Also, the performance counter `UOPS_RETIRED.SCALAR_SIMD` that both LIKWID and Perf read, can be triggered by non-arithmetic operations, as long as they are vector instructions. This can result in some inaccuracy in the reporting of arithmetic vectorization efficiency from LIKWID and Perf.

As shown in Table IV, HPGMG has a very high vectorization efficiency reported by all tools and it agrees with the general understanding of the code that it is well vectorized. Nyx is not. All three tools report very low efficiency ratio, with LIKWID and Perf possibly overreporting due to their inclusion of the non-arithmetic vector micro-ops. Tiramisu is highly vectorized due to its utilization of libraries such as MKL-DNN and cuDNN, and all three tools report a near-1 vectorization efficiency.

C. Analysis

In this section, the performance tools are examined along four qualitative axes: usability, overhead, actionability, and accuracy.

*1) **Usability**:* Three major aspects affect usability. These include recompilation, instrumentation, and special(privileged) access. Any tool requiring manual instrumentation by definition mandates recompilation.

Where as IPM requires no code modification for profiling and can be run without recompilation by appending the 'native' execution command, CrayPat requires recompilation for both sampling and tracing and is thus less attractive when profiling workflows like Tiramisu.

For basic, time-integrated statistics, LIKWID requires neither recompilation nor instrumentation. However, when one desires regional profiling, manual instrumentation with LIKWID's Marker API is required. In all cases, LIKWID requires elevated admin access such as the paranoid level defined in file $/proc/sys/kernel/perf\_event\_paranoid$ being reduced to 0 (default is 1). Like LIKWID, VTune requires additional kernel modules and privileged access. Although it requires dynamic linking, no manual instrumentation is required. Similarly, for time-integrated statistics, SDE does not require manual instrumentation. However, like LIKWID, instrumentation with its API facilitates region based profiling. Unlike LIKWID or VTune where their designers have done the hard work and encoded microarchitecture-specific counter values, Perf requires users to explicitly specify the hexadecimal counter identifiers of the counters they wish to monitor. Nevertheless, no code instrumentation is required. Perf does not support region-based profiling.

*2) **Overhead**:* Perhaps one of the biggest impediments to the deployment and acceptance of a performance tool is the overhead its use incurs. $10\times$ overheads can turn 30 minute debug jobs into runs lasting most of a work day. Higher overheads or attempting to profile longer runs can become prohibitive as one may be limited by the maximum job time.

CrayPat(sampling), LIKWID, Perf, and IPM all have very low nominal overheads allowing data collection for long-running applications. That being said, CrayPat overhead can increase quickly when tracing is being used, to the point where it can become intractable when many function groups are being traced. Similarly, LIKWID can incur substantial overheads when aggregating performance data at the end of a run at high concurrency creating an impediment to its use at scale (n.b. this can be mitigated by collecting data on a subset of the processes).

VTune (e.g. Memory Access) and SDE both incur very high overheads. For SDE, there is substantial slowdown during application execution as well as substantial overheads in startup and finalization. As such, it is not suitable for profiling long or high-concurrency production runs. Like LIKWID, for MPI and hybrid codes, some of VTune's

overhead can be mitigated by only collecting data on a single rank. Regardless, as 'finalization' is a serial operation, it is best to defer it to be a post-processing step that is run on a traditional Xeon rather than on a compute-optimized KNL node.

*3) **Actionability**:* Actionability is multi-faceted. First, users should be informed of hotspots (where time is going). This should be coupled with some efficiency analysis to motivate optimization. Finally, guidance on viable optimizations distills the nearly unbounded optimization space into something tractable.

CrayPat supports a wide range of performance measurements and can report on hotspots, load balance, MPI communication, and IO. The level of detail is selectable at runtime and presented through a variety of text and graphical reports. Similarly, VTune can also report on hotspots, load balance, concurrency and locks and waits, provide optimization suggestions, but can also add efficiency metrics. Where as Perf and LIKWID both provide access to performance counter data, LIKWID distills this down to GFLOP's, data movement, and bandwidths that can be used to infer overall efficiency. However, LIKWID lacks information on hotspots, load balance, or MPI communication pattern. IPM is nominally focused on MPI communication and thus can highlight communication hot spots, but lacks information on effective utilization of the network. Although SDE may provide the gold standard in detailed instruction characteristics, its ability to identify hot spots, memory usage, and efficiency is minimal.

*4) **Accuracy**:* Most tools produce accurate performance information for the metrics they can measure. LIKWID's and Perf's counts of total floating-point operations is inaccurate because they both rely on the `UOPS_RETIRED.SCALAR_SIMD` performance counter which counts both arithmetic floating-point operation related vector micro-ops as well as non-arithmetic floating-point operation related ones. For applications that do not have good data locality or vectorization, this could skew LIKWID and Perf's measurement by a large amount.

IV. Conclusion

Through this exploration of six profiling tools, it is found that CrayPat, LIKWID and Perf are able to collect an important set of actionable performance information with low overhead and high accuracy. Furthermore, without the requirement of code instrumentation, and with some predefined performance groups readily available, LIKWID provides an easy-to-use, easy-to-automate way to enable mass performance data collection on Cray systems.

Acknowledgment

This material is based upon work supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] L. DeRose, B. Homer, D. Johnson, S. Kaufmann, and H. Poxon, "Cray performance analysis tools," pp. 191–199, 2008.

[2] T. Röhl, J. Treibig, G. Hager, and G. Wellein, "Overhead analysis of performance counter measurements," pp. 176–185, 9 2014.

[3] D. Skinner, *Performance Monitoring of Parallel Scientific Applications*, 5 2005. [Online]. Available: http://www.osti.gov/scitech/servlets/purl/881368

[4] (2017) Intel VTune Amplifier. [Online]. Available: https://software.intel.com/en-us/intel-vtune-amplifier-xe

[5] (2017) Intel Software Development Emulator. [Online]. Available: https://software.intel.com/en-us/articles/intel-software-development-emulator

[6] (2017) Perf Wiki. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[7] https://bitbucket.org/hpgmg/hpgmg.

[8] https://github.com/AMReX-Astro/Nyx.

[9] https://github.com/azrael417/ClimDeepLearn.

[10] http://crd.lbl.gov/departments/computer-science/PAR/research/hpgmg.

[11] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Luki, and E. V. Andel, "Nyx: A massively parallel amr code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013. [Online]. Available: http://stacks.iop.org/0004-637X/765/i=1/a=39

[12] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, 2017, pp. 1175–1183.

[13] (2017) TensorFlow. [Online]. Available: https://www.tensorflow.org

[14] "Introducing DNN primitives in Intel® Math Kernel Library," https://software.intel.com/en-us/articles/introducing-dnn-primitives-in-intelr-mkl, 2017.

[15] cuDNN website. [Online]. Available: https://developer.nvidia.com/cudnn

[16] https://github.com/herumi/xbyak.

[17] http://iopscience.iop.org/article/10.1088/0004-637X/765/1/39/meta.

[18] https://academic.oup.com/mnras/article/446/4/3697/2892888.