Roofline Analysis with Cray Performance Analysis Tools (CrayPat) and Roofline-based Performance Projections for a Future Architecture

JaeHyuk Kwack, Galen Arnold, Celso Mendes, and Gregory H Bauer
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
e-mail: {jkwack2, gwarnold, cmendes, gbauer}@illinois.edu

Abstract— **The roofline analysis model is a visually intuitive performance model used to understand hardware performance limitations as well as potential benefits of optimizations for science and engineering applications. Intel Advisor has provided a useful roofline analysis feature since its version 2017 update 2, but it is not widely compatible with other compilers and chip-architectures. As an alternative, we have employed Cray Performance Analysis Tools (CrayPat) that are more flexible for multiple compilers and architectures. First, we present our procedure for measuring a reliable computational intensity for roofline analysis. We performed several numerical studies for validation via manually derived reference data as well as data from Intel Advisor. Second, we provide roofline analysis results on Blue Waters for several HPC benchmarks and sparse linear algebra libraries. In addition, we present an example of roofline-based performance projection for a future system.**

Keywords-roofline performance analysis; profiling; high performance computing; CrayPat; performance projection

I. Introduction

Measuring performance of High-Performance Computing (HPC) applications is exciting but challenging because their algorithms are often too complicated for HPC application developers to readily perceive their major performance bottlenecks. In addition, modern HPC systems are complex enough to mask choke points from the developers. As many experts in HPC communities have realized and raised [1-3], performance of modern HPC systems and their applications cannot be simply measured by an outdated kernel that is closely connected with the peak performance of processors. The performance of many HPC applications is actually bounded by other components of modern HPC systems: latency and bandwidth of cache, memory and network. For that reason, we have made continuous efforts to develop comprehensive and accurate benchmarks for modern HPC systems by employing prevailing kernels used by modern HPC codes, and popular HPC applications engaged by many domain-specific science groups [4-6].

The roofline analysis model [7] is a powerful tool for HPC code developers since it provides visually intuitive plots for the performance of their applications and kernels in terms of computational intensity (CI) and flop-rate. Intel® Advisor started providing a useful cache-aware roofline analysis model [8] in a handy way since its version 2017 update 2; however, it is not widely compatible with other compilers (e.g., PGI, CRAY, GNU) and other processor-architectures (e.g., AMD, ARM, GPUs). As an alternative, we have employed Cray Performance Analysis Tools (CrayPat) for a main platform of our roofline analysis model, making it flexible for multiple compilers and architectures. We have validated our CrayPat-based roofline analysis method via manually counted reference data from our codes as well as data from Intel Advisor. In addition, we have applied the proposed method for several HPC benchmarks and sparse linear algebra libraries in order to characterize them in term of computational intensity.

For numerical validation and development, we have used the Blue Waters system managed by National Center for Supercomputing Applications (NCSA) at University of Illinois at Urbana-Champaign. Blue Waters [9-10] is a Cray XE6/XK7 hybrid supercomputer with 22,640 CPU-based XE6 nodes and 4,228 GPU-enabled XK7 nodes. In this study, we have mainly used the XE6 dual-socket nodes that are populated with 2 AMD Interlagos model 6276 CPU processors with a nominal clock speed of at least 2.3GHz and 64 GB of DDR memory. Each XE node has 16 Floating-Point Units (FPUs) shared by 32 integer cores, 16 KB L1 data cache per core, 64 KB instruction cache per FPU, 2 MB L2 data cache per FPU, and 16 MB L3 data cache per socket.

This paper is organized as follows: Section II presents roofline measurements of Blue Waters XE nodes. In Section III, we discuss how to measure computational intensity based on hardware performance counters and validate it. Section IV covers our roofline performance analysis results for selected HPC benchmarks and several sparse direct/iterative linear equation solvers. Section V presents our roofline-based performance projection of a production-level science application for a new processor. In Section VI, we summarize our work in this study.

II. Roofline Measurements of a Blue Waters XE node

We measured the maximum bandwidth for the various levels of the memory and the maximum flop-rate on a Blue Waters XE node using the Empirical Roofline Tool (ERT) [11] version 1.1.0. Three types of compilers were used with the ERT as follows:

- GNU compiler: gcc/4.9.3 and gcc/6.3.0
- Cray compiler: cce/8.5.8
- PGI compiler: pgi/17.5.0

This paper has been accepted and will be published as an article in a special issue of Concurrency and Computation Practice and Experience on the Cray User Group 2018.

The ERT basically provides the maximum flop-rates for double precision (DP) calculations. In order to get the maximum flop-rates for single precision (SP) calculations, we changed “double” variables to “float” variables in driver1.c, kernel1.h, and kernel1.c of the ERT. For both DP and SP, we considered the following types of peak flop-rates:

- Vector FMA peak flop-rate: the maximum flop-rate with the SIMD vectorization and the FMA4 instructions
- Vector Add peak flop-rate: the maximum flop-rate with the SIMD vectorization and without the FMA4 instructions
- Scalar Add peak flop-rate: the maximum flop-rate without the SIMD vectorization and the FMA4 instructions

TABLE I. EMPIRICAL ROOFLINE TOOL (ERT) MEASUREMENTS ON A BLUE WATERS XE NODE

Type	**Compiler**	**Flop-rate (GFLOP/s)**	**Bandwidth (GB/s)**		
			L1	***L2***	***DRAM***
DP Vector FMA	gcc/4.9.3	223.7	833.1	383.0	62.0
	gcc/6.3.0	222.5	812.7	386.5	62.6
	cce/8.5.8	228.2	943.9	388.0	62.2
	pgi/17.5.0	211.4	980.0	398.1	62.2
	Max	228.2	980.0	398.1	62.6
SP Vector FMA	gcc/4.9.3	447.0	823.9	382.2	62.2
	gcc/6.3.0	450.8	815.4	385.0	61.7
	pgi/17.5.0	414.3	592.1	360.7	62.2
	Max	450.8	823.9	385.0	62.2
DP Vector Add	gcc/4.9.3	114.6	824.2	383.6	62.2
	gcc/6.3.0	115.8	715.0	389.2	62.2
	cce/8.5.8	117.8	940.5	382.1	62.2
	Max	117.8	940.5	389.2	62.2
SP Vector Add	gcc/4.9.3	225.5	816.8	385.9	61.8
	gcc/6.3.0	234.6	822.5	386.3	62.0
	cce/8.5.8	235.3	934.9	382.6	62.1
	Max	235.3	934.9	385.9	62.1
DP Scalar Add	gcc/4.9.3	60.1	495.0	357.5	62.1
	gcc/6.3.0	64.7	492.4	362.3	62.4
	cce/8.5.8	60.3	547.9	379.8	62.2
	Max	64.7	547.9	379.8	62.4
SP Scalar Add	gcc/4.9.3	60.8	266.6		62.3
	gcc/6.3.0	64.1	274.4		62.3
	cce/8.5.8	61.1	260.1		62.2
	Max	64.1	274.4		62.3

We measured three types of peak flop-rates with GNU and Cray compilers for DP and SP, while we only evaluated the Vector FMA peak flop-rates for DP and SP with PGI compiler. Table I presents our ERT measurements on a Blue Waters XE node. Since multiple executions for SP Vector FMA peak with Cray compiler generated unreasonable data, we didn’t add the data to the table. During all evaluations, ERT missed L3 cache bandwidth of AMD Interlagos processor, and ERT even missed L2 cache bandwidth during SP Scalar Add peak measurements. SP peak flop-rates are generally twice DP peak flop-rates with vectorization (i.e., Vector FMA and Vector Add peaks), while SP peaks are similar to DP peaks without vectorization (i.e., Scalar Add).

Figure 1 presents the maximum values of measured bandwidths and flop-rates from Table I, and corresponding rooflines for a Blue Waters XE node. The red lines represent rooflines for L1, L2 and DRAM bandwidths. Two blue lines describes Vector FMA peaks for SP and DP, while the magenta lines present rooflines of Vector Add peak flop-rates for SP and DP. The green line shows Scalar Add peak performance for SP and DP.

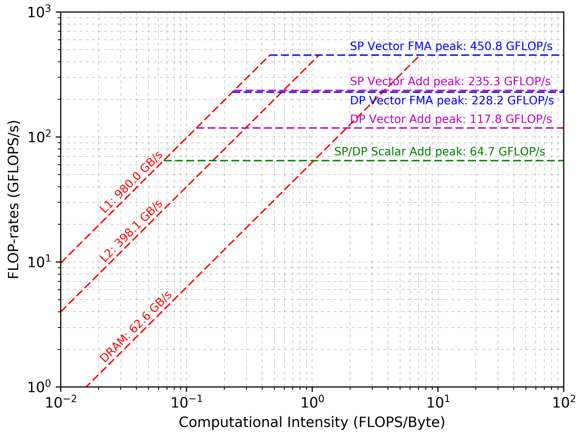


Figure 1. ERT measurements-based rooflines of a Blue Waters XE node

III. COMPUTATIONAL INTENSITIES BASED ON HARDWARE PERFORMANCE COUNTERS

Computational intensity (i.e., arithmetic intensity, operational intensity) is the ratio of floating-point operations (FLOPS) to data movements (bytes) among processor registers, L1/L2/L3 data caches and DRAM. One of the most challenging aspects of performing roofline analysis with CrayPat is how to validate if the measured computational intensity (CI) with selected hardware performance counters via CrayPat is reliable or not. For that, we have investigated many types of hardware performance counters and validated the derived CIs via manually-counted data from our own codes as well as arithmetic intensity data from Intel Advisor.

A. Validation via Kernels for Computing Geometric Series

In order to find an optimal set of hardware performance counters for a reliable CI measurement, we implemented kernels for computing a geometric series. The nth order of geometric series in (1) requires only 2 memory references (i.e., 1 read and 1 store) as presented in (2); therefore, data movements for a geometric series with SP and DP are 8 bytes and 16 bytes, respectively. Even though FLOPS depend on the implementation and its compiler optimization, the nth order geometric series includes at least n additions and n-1 multiplications; accordingly, the minimum FLOPS is 2n-1 as given in (3). As a result, kernels for computing geometric series can cover a wide range of CIs simply by employing various orders of series, as defined in (4). For DP, the CI is greater than or equal to 0.0625 FLOPS/byte, while it is greater than or equal to 0.125 FLOPS/byte for SP.

$$OM(n) = 1+M+M^2+M^3+ \dots +M^n \quad (1)$$
$$\text{Data movement} = 2 \text{ variables} \quad (2)$$
$$\text{Minimum FLOPs} = 2n - 1 \quad (3)$$
$$CI = (2n-1) / (2 \text{ variables}) \quad (4)$$

We implemented the geometric series computation in three ways (i.e., inline-implementation, recursive loops, and flat loops) as presented in Table II. All kernels start with a m×m array whose components are initialized by different random numbers. We tested the kernels for the order from 1 to 29; therefore, the ranges of theoretical CI in FLOPS/byte are from 0.0625 to 3.5625 for DP and from 0.125 to 7.125 for SP. Using 32 MPI ranks, we fully activated all cores in a single XE node during these tests.

TABLE II. THREE TYPES OF KERNELS IMPLEMENTED FOR COMPUTING N-TH GEOMETRIC SERIES

Inline-implementation: Loops for i and j from 1 to m: OM(i,j) = 1.0 + M(i,j) + M(i,j)^2 + M(i,j)^3 + ... + M(i,j)^n
Recursive loops: Loops for i and j from 1 to m: OM(i,j) = 1.0   A loop for k from 1 to n: OM(i,j) = OM(i,j)*M(i,j) + 1.0
Flat loops: Loops for i and j from 1 to m: OM(i,j) = 1.0   A loop for k from 1 to n: OM(i,j) = OM(i,j) + M(i,j)^k

We have tested a large set of hardware performance counters to measure data movements for roofline performance analysis, and then selected one of the most reliable counters, DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE: GOOD that represents the number of data cache refills satisfied from the L2 cache and/or the system with valid final status [12]. Each increment of this counter reflects a 64-byte transfer, so we convert the raw data to bytes with the ratio. In this paper we use DCR_GOOD as an abbreviation for DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE: GOOD.

As a default, CrayPat uses L1_DCA (i.e., level 1 data cache accesses) data for data movements in computing the default CI. L1_DCA reports the number of load and store references, and we consider a single reference as a single vector length (i.e., 16 bytes = 128 bits for Interlagos processors). We compare our choice (i.e., DCR_GOOD) of hardware counter data with the CrayPat-default data (i.e., L1_DCA) in this section.

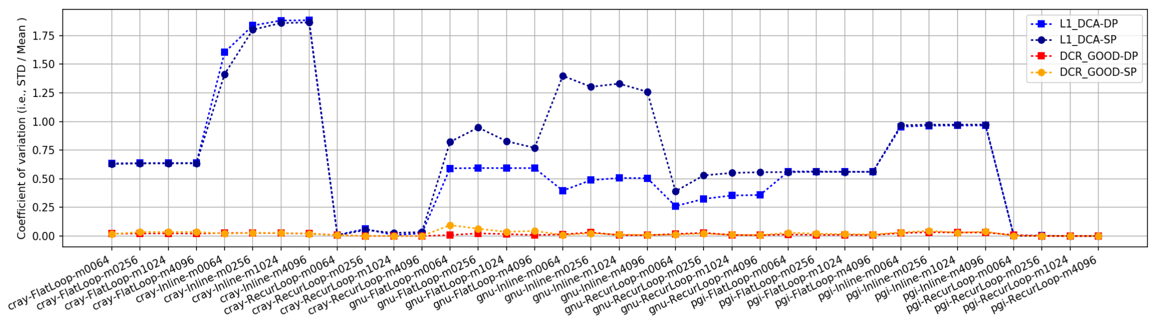
We performed the validation tests with an extensive set of runtime parameters (i.e., 4176 cases in total) as follows:

- The order of the series: 1 to 29 (i.e., 29 cases)
- The size of array per MPI rank: 64^2, 256^2, 1024^2, and 4096^2 (i.e., 4 cases)
- Variable type: 4-byte SP and 8-byte DP (i.e., 2 cases)
- Compiler type: gnu, cray, and pgi (i.e., 3 cases)
- Optimization level: O0 and O3 (i.e., 2 cases)
- Implementation type: inline, recursive loop, and flat loop (i.e., 3 cases)

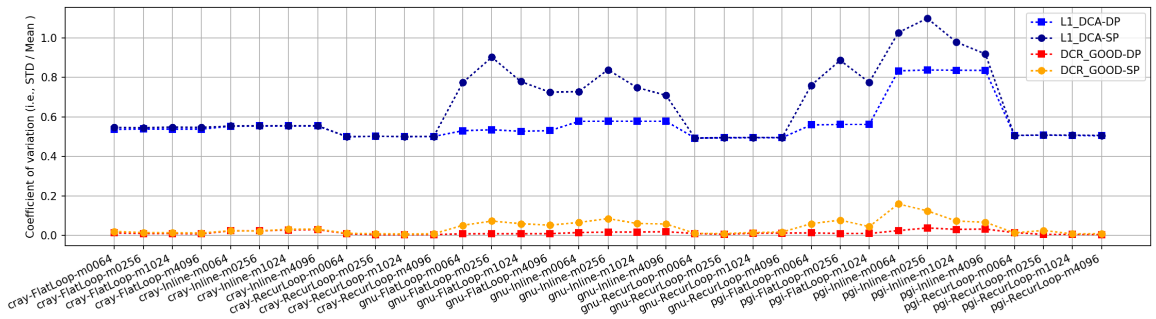
Due to the limited space in this paper, we cannot provide data for all 4176 cases. Instead, we present statistical data and several selected data for discussion.

Figures 2(a) and 2(b) present the coefficients of variation (i.e., ratios of the standard deviation over the mean value) of hardware performance counter data for byte movements with the O3 and O0 optimizations, respectively. Figure 3(a) and 3(b) show the range of the minimum and maximum values of byte measurements with the O3 and O0 optimization. Each figure has four plots for L1_DCA with DP, L1_DCA with SP, DCR_GOOD with DP, and DCR_GOOD with SP. Each point of the plots represents 29 cases that use the same size of array, variable type, compiler type, optimization level and implementation type, but compute different orders of the geometric series. The x-axis represents the characteristics of the case in the format of {compiler type}-{implementation type}-{row size of the tested array}. In general, DCR_GOOD-based measurements show very little variation compared to L1_DCA-based byte measurements. Depending on case conditions, L1_DCA with the O3 optimization provides relatively small variation for some cases (e.g., cray_RecurLoop and pgi_RecurLoop cases), but L1_DCA with the O0 optimization provides very large variation for all cases. From this comparison, we realize that L1_DCA-based measurements are not precise enough for measuring byte movements for roofline analysis, and DCR_GOOD provides very reliable data for byte measurements.

Based on the variation plots in Figures 2 and 3, we selected several cases to see raw data and corresponding exact values (based on (2)) as presented in Figure 4. Figure 4(a) shows a case, cray-RecurLoop-m4096-O3 that uses cray compiler with the O3 optimization and computes the geometric series for a 4096^2 array with recursive loops. As presented in the variation plots, DCR_GOOD and L1_DCA data of Figure 4(a) show little variation, and both of them provide very accurate data compared to the exact values. In Figures 4(b), 4(c), 4(d), and 4(e), L1_DCA plots show large variation as well as inaccurate data, while DCR_GOOD plots show very precise and accurate data. In Figure 4(f), L1_DCA plots show very little variation as presented in Figures 2 and 3; however, the measured data overestimate the byte movements, so it turns out the accuracy of L1_DCA is poor on the case.

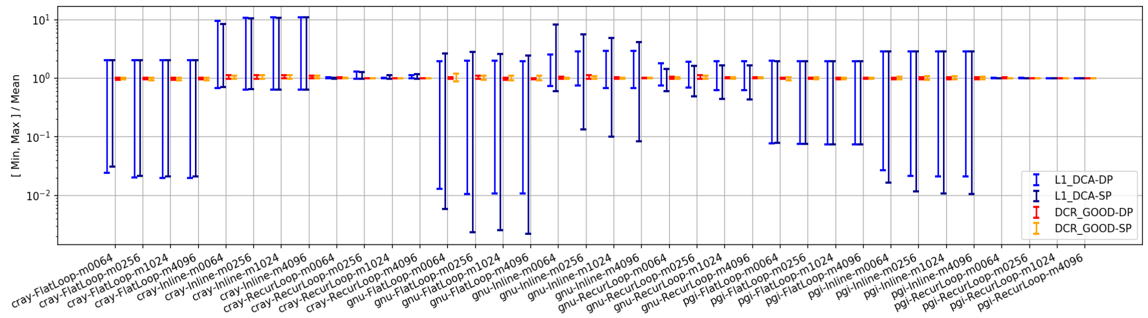


(a) The O3 optimization

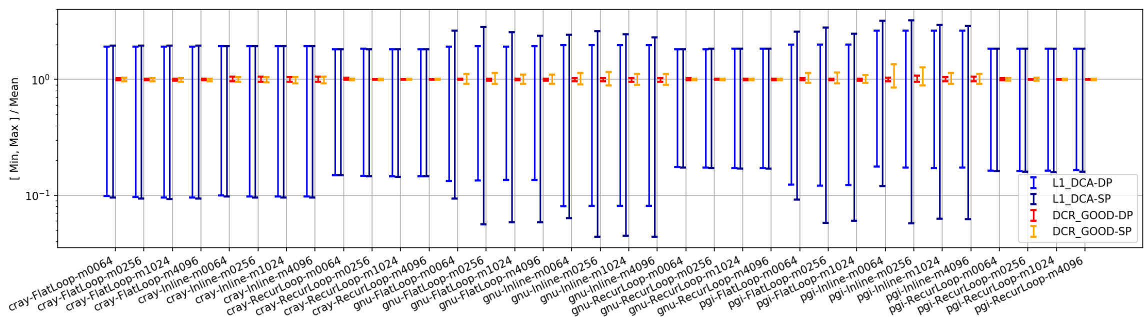


(b) The O0 optimization

Figure 2. The coefficients of variation of hardware counter data for byte movements with the O3 and O0 optimizations

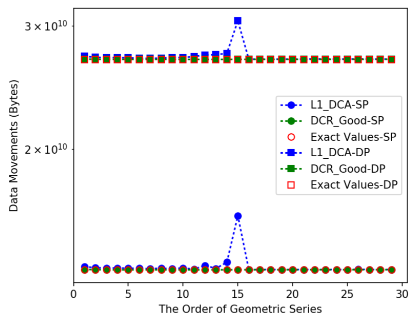


(a) The O3 optimization

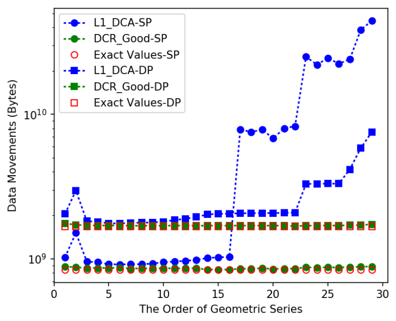


(b) The O0 optimization

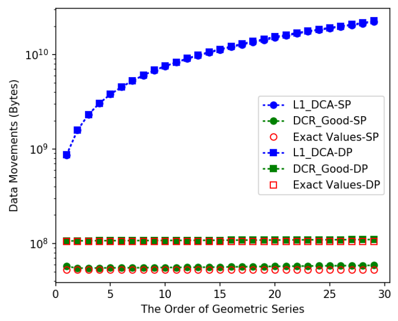
Figure 3. The range of the minimum and maximum values of byte measurements with the O3 and O0 optimizations



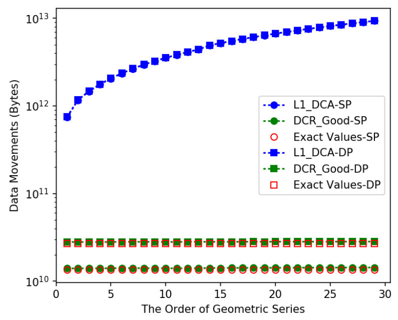
(a) cray-RecurLoop-m4096-O3



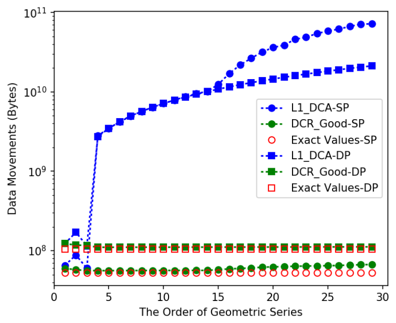
(b) gnu-Inline-m1024-O3



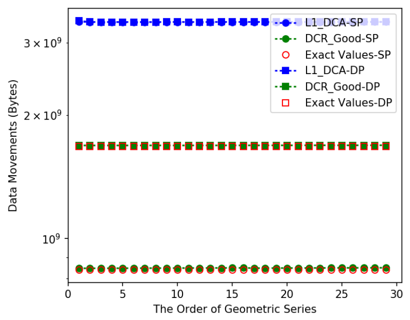
(c) pgi-FlatLoop-m0256-O3



(d) cray-RecurLoop-m4096-O0

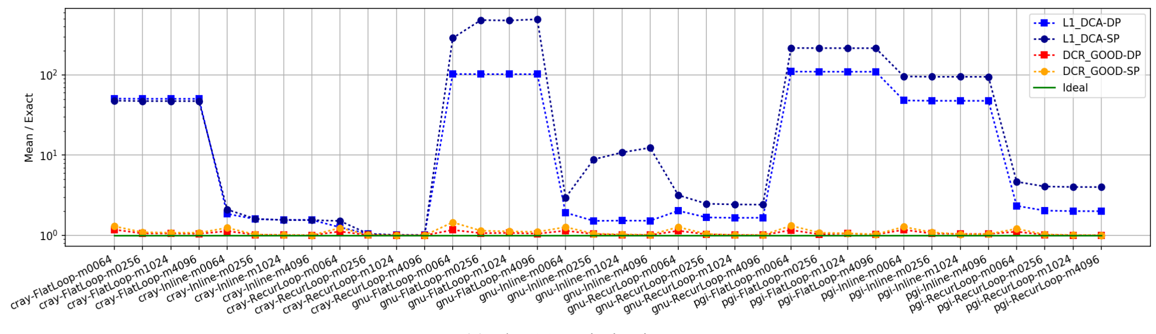


(e) gnu-FlatLoop-m0256-O3

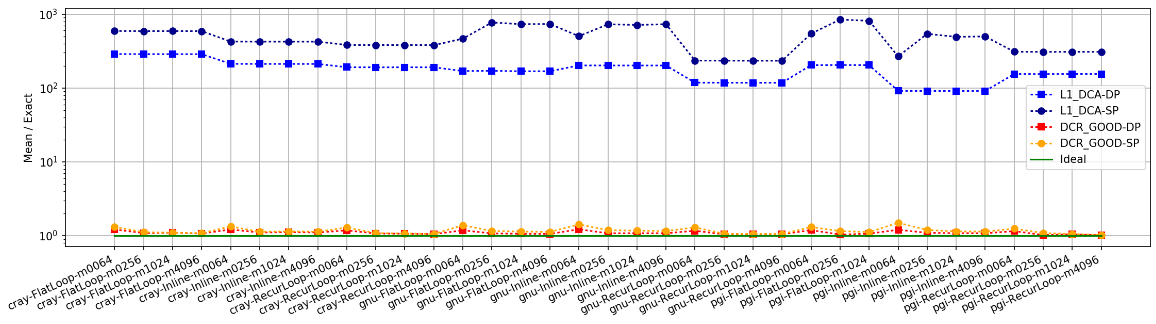


(f) pgi-RecurLoop-m1024-O3

Figure 4. Raw data of byte movements for selected cases (i.e., {compiler type}-{implementation type}-{row size of array}-{optimization level})



(a) The O3 optimization



(b) The O0 optimization

Figure 5. The ratio of the mean value of hardware counter data over the exact byte movements with the O3 and O0 optimizations

Figure 5 presents plots for the ratio of the mean value of hardware counter data over the exact byte movements for all cases that represents overall accuracy of L1_DCA and DCR_GOOD measurements. DCR_GOOD generates very consistent and accurate data for byte movements for all cases, while L1_DCA data are fluctuating and unreliable. Accordingly, it turns out that the CrayPat-default CI is not reliable to estimate the ratio of FLOPS over bytes. Instead, we recommend using DCR_GOOD-based CI for roofline performance analysis.

B. Validation via 2D/3D stencil codes

In addition to the geometric series code, we implemented our own stencil codes in 2D and 3D in order to validate measured byte movements and FLOPs via CrayPat, and corresponding CIs. Table III shows main kernels of 2D and 3D stencil codes instrumented by CrayPat API. As presented in Table IV, the 5-point 2D stencil kernel includes 4 floating-point operations and 5 data movements (i.e., 4 reads and 1 stores) per point, so CIs for DP and SP are 0.1 and 0.2 FLOPS/byte. The 7-point 3D stencil kernel includes 6 floating-point operations and 7 data movements (i.e., 6 reads and 1 stores); therefore, CIs for DP and SP are 0.1071 and 0.2143 in FLOPS/byte, respectively.

TABLE III. THE MAIN KERNELS OF THE 2D/3D STENCIL CODES

5-point 2D stencil code:

```
Loops for iter from 1 to niter:
    PAT_region_begin()
    Loops for i and j from 2 to n+1:
        OM(i,j) = 0.25* ( M(i+1,j) + M(i-1,j) + M(i,j+1) + M(i,j-1)  )
    PAT_region_end()
    Loops for i and j from 2 to n+1:
        M(i,j) = OM(i,j)
```

7-point 3D stencil code:

```
Loops for iter from 1 to niter:
    PAT_region_begin()
    Loops for i, j and k from 2 to n+1:
        OM(i,j,k) = 0.1666666666666666667 * ( M(i+1,j,k) + M(i-1,j,k)
                  + M(i,j+1,k) + M(i,j-1,k) + M(i,j,k+1) + M(i,j,k-1)  )
    PAT_region_end()
    Loops for i and j from 2 to n+1:
        M(i,j,k) = OM(i,j,k)
```

TABLE IV. FLOPS, DATA MOVEMENTS AND CI OF THE MAIN KERNELS OF THE 2D/3D STENCIL CODES

5-point 2D stencil code:

FLOPs = 4*n*n*niter
Data Movements = 5*n*n*niter
CI for DP = 4./5/8 = 0.1 FLOPS/byte
CI for SP = 4./5/4 = 0.2 FLOPS/byte

7-point 3D stencil code:

FLOPs = 6*n*n*n*niter
Data Movements = 7*n*n*n*niter
CI for DP = 6./7/8 = 0.1071 FLOPS/byte
CI for SP = 6./7/4 = 0.2143 FLOPS/byte

During validation tests, we fully activated all cores in a single XE node by launching kernels on 32 MPI ranks. With CrayPat version 6.4.6, we collected hardware performance counter data only for the main kernels enclosed by CrayPat API. We performed the kernels with a large set of parameters (i.e., 72 cases in total) as follows:

- Stencil size per MPI rank (i.e., $(n+2)^2$ for 2D and $(n+2)^3$ for 3D): $(2048+2)^2$, $(4096+2)^2$, $(8192+2)^2$ for 2D, and $(128+2)^2$, $(256+2)^2$, $(416+2)^2$ for 3D (i.e., 6 cases in total)
- Variable type: 4-byte SP and 8-byte DP (i.e., 2 cases)
- Compiler type: gnu 4.9.3, cray 8.5.8, and pgi 17.5.0 (i.e., 3 cases)
- Optimization level: O0 and O3 (i.e., 2 cases)

Figure 6 presents the ratio of the measured data movements via CrayPat over the exact data movements. L1_DCA measurements show big fluctuations between the O0 and O3 optimizations as well as between DP and SP. DCR_GOOD measurements are relatively consistent for all compiler types, stencil sizes, optimization levels and variable types. The ratio of DCR_GOOD measurements over exact byte movements is from 0.569 to 1.281, while the ratio of L1_DCA measurements over the exact byte transfers is between 1.798 and 38.3 as presented in Table V. Accordingly, the geometric mean of L1_DCA-based ratios is 5.87, while the geometric mean of DCR_GOOD-based ratios is 0.812. Therefore, it turns out DCR_GOOD measurements are more precise and accurate than L1_DCA measurements for data movements.

TABLE V. STATISTICS OF THE RATIO OF MEASURED BYTE OVER EXACT BYTE MOVEMENTS

Counters	Min	Geometric mean	Max
L1_DCA	1.798	5.87	38.3
DCR_GOOD	0.569	0.812	1.281

Figure 7 shows the measured FLOPs via CrayPat as well as the exact FLOPs. For all compiler types, variables types, stencil sizes and optimization levels, the FLOP measurements are extremely accurate. Figure 8 presents the ratio of the computed CIs over the exact CIs. DCR_GOOD-based CIs are much more accurate than L1_DCA-based CIs. As given in Table VI, the geometric mean of DCR_GOOD-based CIs is 23.1% higher than the exact CIs, while the geometric mean of L1_DCA-based CIs is around one sixth of the exact CIs. The ratio of the maximum CI over the minimum CI of L1_DCA-based CIs is 21.3, while the ratio of DCR_GOOD-based CIs is only 2.25. In summary, DCR_GOOD-based CIs are much more reliable than L1_DCA-based CIs.

TABLE VI. STATISTICS OF THE RATIO OF DERIVED CI OVER EXACT CI

Counters	Min	Geometric mean	Max	Max/min
L1_DCA	0.0261	0.1703	0.556	21.3
DCR_GOOD	0.781	1.231	1.757	2.25

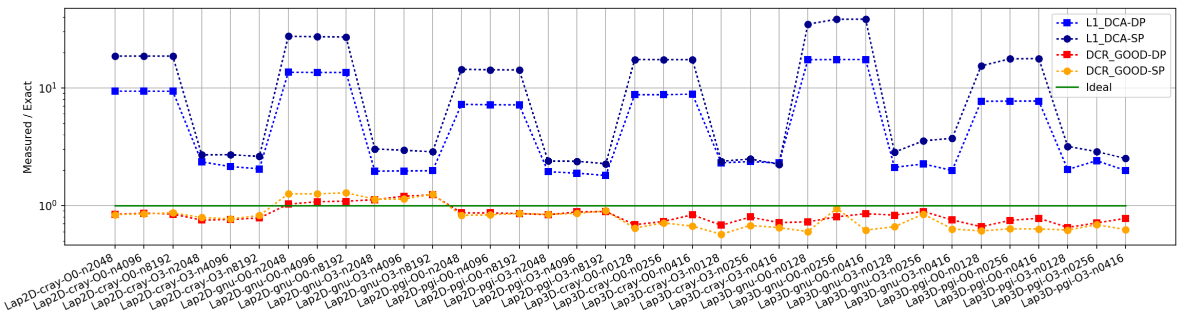


Figure 6. The ratio of the measured data movements via CrayPat over the exact data movements of 2D/3D stencil codes

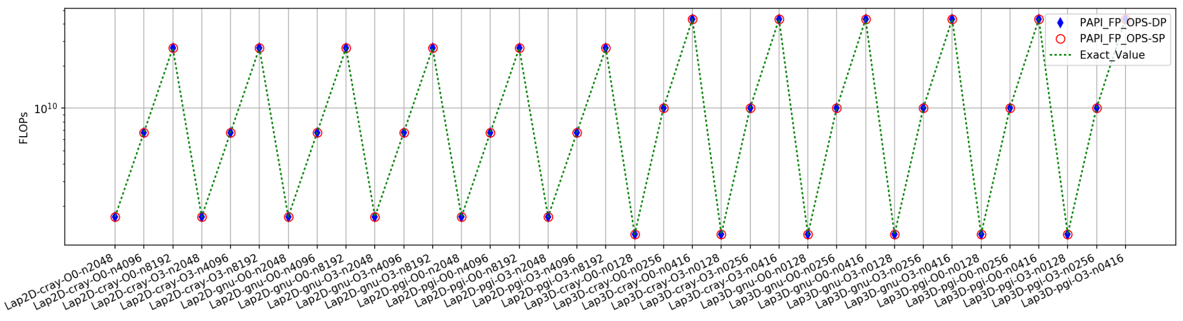


Figure 7. FLOP measurements via CrayPat of 2D/3D stencil codes

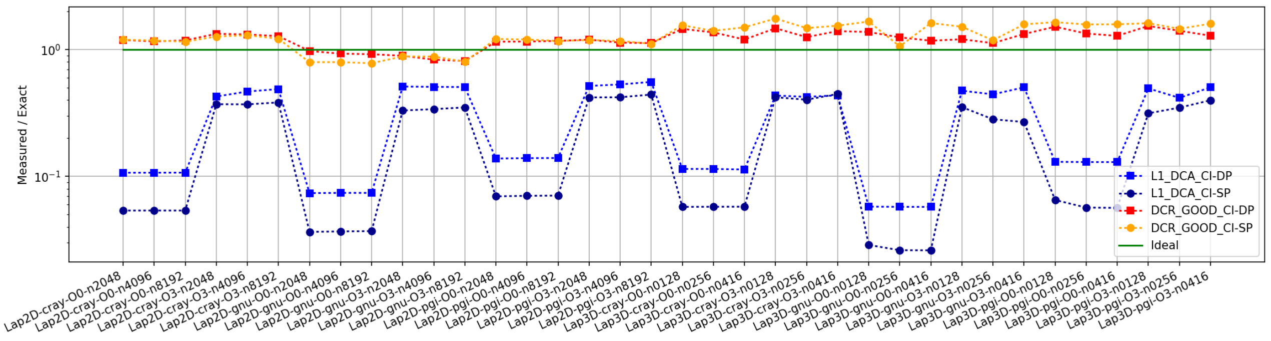
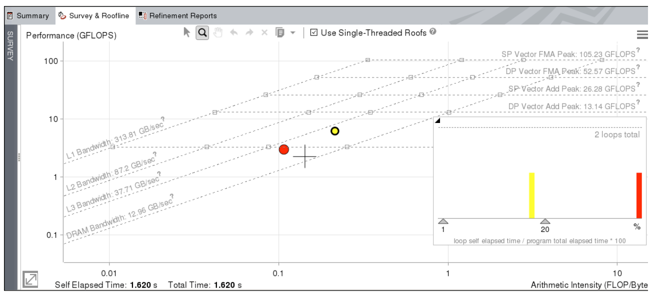


Figure 8. The ratio of the computed CIs over the exact CIs of 2D/3D stencil codes



Summary
Survey & Roofline
Refinement Reports
SURVEY
Performance (GFLOPS)
Use Single-Threaded Roofs
SP Vector FMA Peak: 105.23 GFLOPS
DP Vector FMA Peak: 52.57 GFLOPS
SP Vector Add Peak: 26.28 GFLOPS
DP Vector Add Peak: 13.14 GFLOPS
L1 Bandwidth: 313.81 GB/sec
L2 Bandwidth: 87.2 GB/sec
L3 Bandwidth: 37.71 GB/sec
DRAM Bandwidth: 12.96 GB/sec
2 loops total
loop self elapsed time / program total elapsed time * 100
100
10
1
0.1
0.01
0.1
1
10
Self Elapsed Time: 1.620 s
Total Time: 1.620 s
Arithmetic Intensity (FLOP/Byte)

Figure 9. A screenshot of Intel Advisor Roofline features; 3D kernel with $(256+2)^3$ stencil (red dot for DP, yellow dot for SP)

TABLE VII. CI FROM INTEL ADVISOR (INTEL/18.0 WITH -O3 -XHOST)

Stencil size	Source	CI with DP	CI with SP
2D: $(2048+2)^2$	Intel Advisor	0.10002	0.20008
3D: $(256+2)^3$	Intel Advisor	0.10726	0.21477

As a reference, 2D/3D stencil kernels were performed with Intel Advisor on Intel E5-2680V4 (Broadwell) processors. Figure 9 shows a screenshot of Intel Advisor roofline features and Table VII shows computational intensities of DP and SP loops. The CI values from Intel Advisor are very close to corresponding exact CI values. However, Intel Advisor may have the following limitations compared to CrayPat-based roofline analysis:

- Intel Advisor is not compatible with other vendor's processors such as AMD, ARM, and NVIDIA.
- Intel Advisor is not compatible with other compilers (e.g., cray, pgi, gcc, and arm).
- Intel Advisor requires at least two executions (i.e., Survey analysis and Trip Counts Analysis with FLOP) for roofline features analysis, thus it may not be suitable for large-scale simulations.

IV. CrayPat-based Roofline Analysis for HPC Benchmarks and Popular Kernels

Using the DCR_GOOD-based CI, we performed CrayPat-based roofline analysis for multiple HPC benchmarks on Blue Waters (See Appendix A for L1_DCA-based roofline analysis plots). We first provide roofline analysis plots for 2D/3D stencil kernels discussed in the previous section. Second, we report CrayPat-based roofline analysis results for High-Performance Conjugate Gradients (HPCG) and High-Performance Geometric Multigrid (HPGMG) benchmarks that represent multiple sets of kernels employed by many science and engineering applications [6, 13, 14]. Last, we introduce roofline analysis plots for sparse linear algebra libraries with multiple non-symmetric matrices from a computational fluid dynamics (CFD) code [4]. We employed PETSc [15] for parallel sparse iterative solvers, and MUMPS [16] and SuperLU [17] for parallel sparse direct solvers.

A. *2D/3D stencil kernels*

We performed CrayPat-based roofline analysis with the 2D/3D stencil kernels presented in Table III. The executable binaries were built with cce/8.5.8 and the O3 optimization, and they were instrumented by CrayPat 6.4.6.

Figure 10 presents CrayPat-based roofline analysis results for the 5-point 2D stencil with various stencil sizes (e.g., $(128+2)^2$ to $(8192+2)^2$). For all cases, the performance of the kernel is bounded by memory bandwidth (e.g., L2 or DRAM bandwidth). For small sizes of stencils (i.e., $(128+2)^2$ for DP and SP, $(256+2)^2$ for DP and SP), the size of required arrays (i.e., M and OM in Table III) for the kernel is less than or equal to 1 MB (i.e., L2 data cache size per a single integer core); therefore, the performance is bounded by the L2 data cache bandwidth. In other cases, the kernel performance is bounded by the DRAM bandwidth.

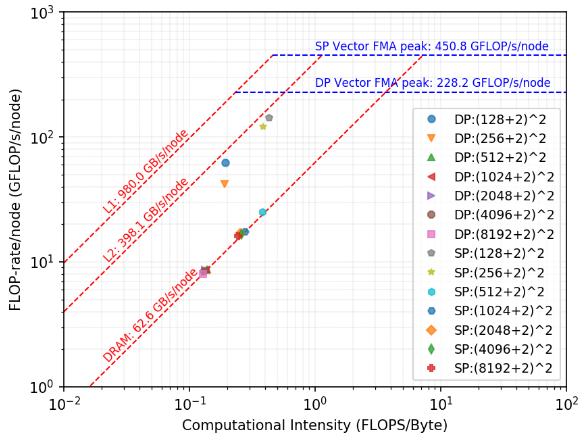


Figure 10. CrayPat-based roofline analysis plots for 5-point 2D stencil kernels

Figure 11 provides CrayPat-based roofline analysis plots for the 7-point 3D stencil kernel with multiple stencil sizes (e.g., $(16+2)^3$ to $(416+2)^3$). Performance with small stencils (e.g., $(16+2)^3$ and $(32+2)^3$ for DP and SP) is bounded by the L2 data cache bandwidth, since the required array size for the kernel is less than the L2 data cache size for a single integer core. For large stencils (e.g., $(64+2)^3$ to $(416+2)^3$), the performance is bounded by the DRAM bandwidth.

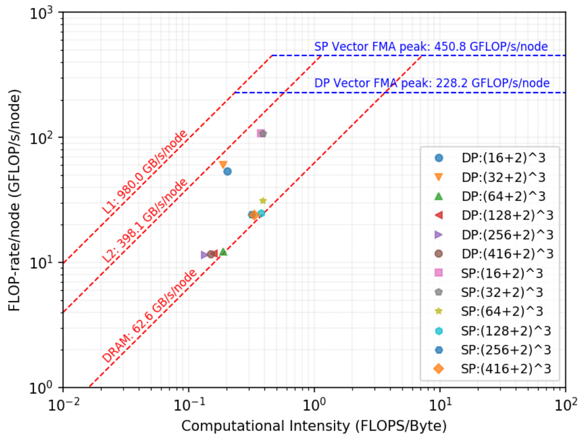


Figure 11. CrayPat-based roofline analysis plots for 7-point 3D stencil kernels

B. *HPCG and HPGMG*

The HPCG [13] solves a synthetically discretized elliptic partial differential equation in 3 dimensions with zero Dirichlet boundary conditions and a synthetic right-hand-size vector with a 27-point stencil operator. It includes dense and sparse computations, dense and sparse collectives, multi-scale execution of kernels through truncated multi-grid V cycles, and data-driven parallelism for unstructured sparse triangular solves. We used gcc/4.9.3 to build the executable binary with the following flags: -O3 -ffast-math -ftree-vectorize -fopenmp.

We performed CrayPat-based roofline analyses for HPCG with 128 MPI ranks on 4 XE nodes for multiple sizes of stencils (e.g., 16^3 to 128^3 per MPI rank), as presented in Figure 12. In all cases, the DCR_GOOD-based CIs are similar to each other, and the performance is bounded by DRAM bandwidth. We cannot observe cache effect for small stencils.

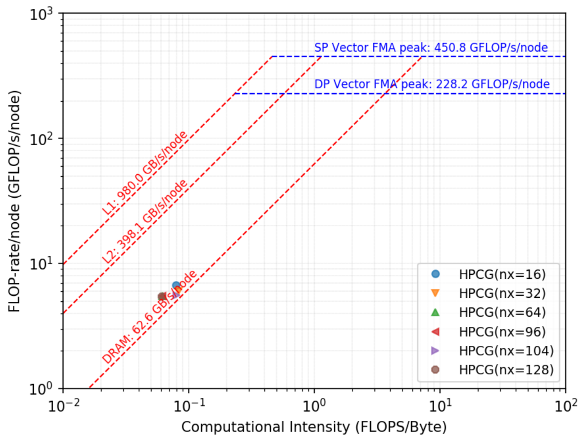


Figure 12. CrayPat-based roofline analysis plots for HPCG on 4 XE nodes

HPGMG-FV [14] employs a geometric multi-grid method to solve an elliptic problem on isotropic Cartesian grids. Using 4th-order accuracy finite volumes, it calculates a flux term on each of the 6 faces on every cell in the entire domain. For the solution process, it employs the full multi-grid (FMG) F-cycle that is a series of progressively deeper geometric multi-grid V-cycles. We built HPGMG-FV with gcc/4.9.3 with the following flags: -O3 -ftree-vectorize -ffast-math -fopenmp -m64. For roofline analyses, we employed 5 different resolutions (i.e., 16^3, 32^3, 64^3, 128^3 and 256^3 cubes per MPI ranks) on 16 XE nodes (i.e., 512 MPI ranks). In Figure 13, the performance of HPGMG-FV gradually increases as the local cube size increases; it is because the ratio of local computations over communications becomes higher and the load imbalance from the FMG F-cycle is getting negligible. The overall performance of HPGMG-FV is bounded by the L2 data cache bandwidth.

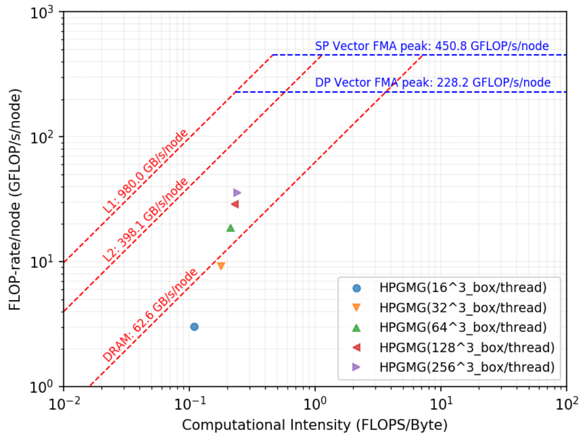


Figure 13. CrayPat-based roofline analysis plots for HPGMG-FV on 16 XE nodes

C. *Sparse Linear Solvers*

PETSc [15] provides a variety of Krylov subspace methods and scalable parallel preconditioners. In this section, we performed roofline analysis with the transpose free QMR (i.e., tfqmr) solver and the classical additive Schwarz (i.e., asm_basic) preconditioner from cray-petsc/3.6.3.0. For the performance analyses, we employed various compressed sparse row (CSR) matrices (i.e., 10K, 83K, 684K and 2Mdofs) derived from a computational fluid dynamics (CFD) code [4]. Figure 14 presents CrayPat-based roofline analysis plots for the selected PETSc solver and preconditioner on 1 XE and 32 XE nodes. Since the PETSc solver mainly uses FPUs, we assigned one MPI rank per one FPU; as a result, we used 16 MPI ranks on 1 XE node and 512 MPI ranks on 32 XE nodes. All cases show pretty similar CI values to each other. Due to cache effects, small inputs yield higher performance per node. For the same CSR matrices, per-node performance on a single node is better than per-node performance on multiple nodes, since single-node computation doesn't require an additional networking cost. However, the networking overhead is not very critical in Figure 15, so we can expect that PETSc users easily accumulate the per-node performance by employing more compute nodes to solve their linear equations.

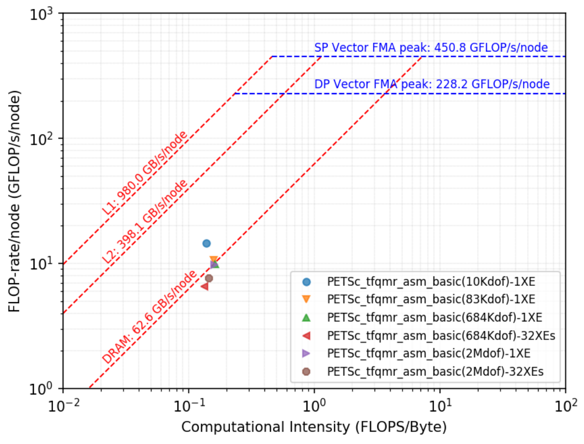


Figure 14. CrayPat-based roofline analysis plots for PETSc tfqmr solver with asm_basic preconditioner on 1 XE or 32 XE nodes

Figures 15 and 16 shows roofline analysis results for MUMPS and SuperLU from cray-tpsl/16.03.1. As the CSR matrix size increases on 1 XE node, the corresponding computational intensity value also increases, and the per-node performance increases accordingly. For the same size of CSR matrix (i.e., 684Kdofs), the computational intensity on 32 XE nodes is lower than on 1 XE node, since the local block size per MPI rank decreases. The per-node performance of sparse direct solvers is highly dependent on their local block size; therefore, their strong scalability may not be competitive with other sparse iterative solvers.

Note that the per-node performance here means GFLOPS/s; therefore, it cannot represent the actual performance of the sparse solvers. Since sparse iterative or sparse direct solvers require totally different numbers of FLOPS to get a solution with a certain accuracy, the time-to-solutions of sparse iterative and direct solvers can be very different from each other. Our former study [4] provides more details.

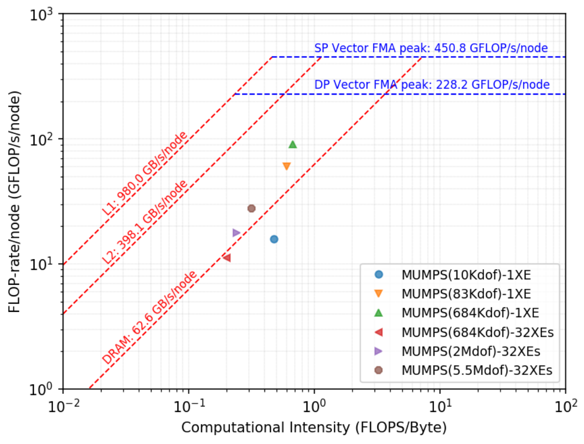


Figure 15. CrayPat-based roofline analysis plots for MUMPS on 1 XE or 32 XE nodes

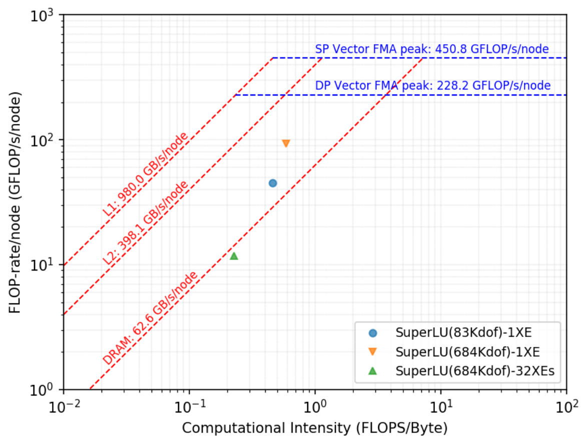


Figure 16. CrayPat-based roofline analysis plots for SuperLU on 1 XE or 32 XE nodes

V. Roofline-based Performance Projection for a New Processor

As an example of roofline-based performance projection for a new processor, we employed SETSM (Surface Extraction with TIN-based Search-space Minimization) code [18] from the ArcticDEM project [19] on Blue Waters.

We first performed the CrayPat-based roofline analysis, as presented in Figure 17. For the analysis, we selected four critical groups (i.e., U/VLL, U/main, U/OT, and U/VLL_B) that used 84.6% of the entire wall time, as presented in Table VIII. According to the CrayPat report, U/main was a serial function, while other three groups were multi-threaded functions. Based on the roofline analysis results, we assumed flop-rates of all groups are bounded by the DRAM bandwidth.

We selected three types of nodes for this practice: dual Ivy Bridge EP processors (i.e., E5-2670V2), dual Haswell processors (i.e., E5-2680V3), and dual Broadwell processors (i.e., E5-2680V4). Table IX shows the measured DRAM bandwidths of an XE node and three target nodes using the stream benchmark [20]. We used DRAM bandwidth per NUMA domain for the maximum bandwidth of the serial processes in the U/main group. While an XE node has four NUMA domains, other nodes with Intel processors have two NUMA domains.

We used (5) for the performance projections of multi-threaded groups (i.e., U/VLL, U/OT and U/VLL_B), and (6) was used for the serial group (i.e., U/main). After the performance projection of four groups, we computed the overall wall time using (7). The constant, 0.846 is the ratio of the sum of four groups' wall times over the total wall time on an XE node. As a result, we computed the roofline-based performance projections as presented in Table X.

$$WT_{target}^{threaded} = WT_{XE} * (BW^{NODE})_{XE} / (BW^{NODE})_{target} \quad (5)$$
$$WT_{target}^{serial} = WT_{XE} * (BW^{NUMA})_{XE} / (BW^{NUMA})_{target} \quad (6)$$
$$WT_{target}^{overall} = (\text{sum of } WT_{target}) / 0.846 \quad (7)$$

Table XI shows actually measured wall times of SETSM on the target nodes. The error range of the projected performance over the measured performance is from 7% to 10%.

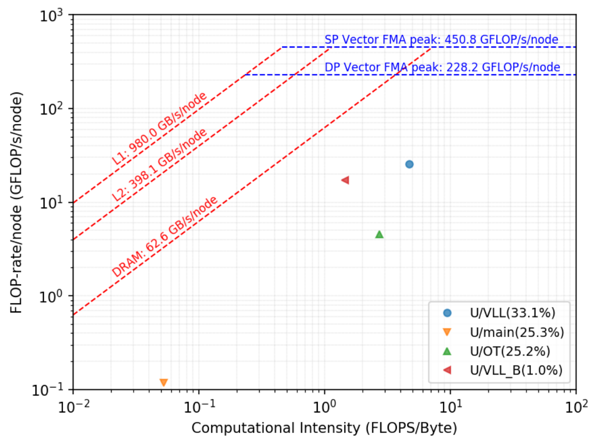


Figure 17. CrayPat-based roofline analysis plots for SETSM (32 threads on 1 XE node) with various optimizations

TABLE VIII. CrayPat performance data of SETSM on 1 XE node

Groups	Wall time (s)	Percentiles (%)	Note
U/VLL	997.6	33.1	multi-threaded
U/main	761.3	25.3	serial
U/OT	757.8	25.2	multi-threaded
U/VLL_B	31.1	1.0	multi-threaded
Others	464.1	15.4	
Total	3011.9	100	

TABLE IX. Measured DRAM bandwidths of an XE node and target nodes

Processor type	DRAM bandwidth per node (GB/s) [(a)]	DRAM bandwidth per NUMA (GB/s)
Dual Interlagos	59.6	14.9[(b)]
Dual Ivy Bridge EP (E5-2670V2)	93.5	46.7[(c)]
Dual Haswell (E5-2680V3)	112.3	56.2[(c)]
Dual Broadwell (E5-2680V4)	125.1	62.5[(c)]

(a) Measured via the stream benchmark, (b) Four NUMA domains per node, (c) Two NUMA domains per node

TABLE X. Roofline-based performace projections

Processor type	Projected wall time (s)	Speed up over an XE
Dual Ivy Bridge EP (E5-2670V2)	1715.3	1.76
Dual Haswell (E5-2680V3)	1427.7	2.11
Dual Broadwell (E5-2680V4)	1282.2	2.35

TABLE XI. Measured wall times on the target nodes

Processor type	Measured wall time (s)	Error of projection (%)
Dual Ivy Bridge EP (E5-2670V2)	1603.0	7.00
Dual Haswell (E5-2680V3)	1293.0	10.42
Dual Broadwell (E5-2680V4)	1168.0	9.78

VI. Concluding remarks

We propose a reliable and practical method for the roofline analysis model with CrayPat. It is widely compatible with many types of compilers and various processor architectures. We first employed CrayPat-default computational intensity data for roofline analysis; however, it turned out the default configuration was not accurate, since the employed hardware performance counter for data movements was not reliable for different optimization levels, multiple compiler types and various problem sizes. We have performed an extensive set of tests to find one of the most optimal hardware counters for data movements; as a result, we recommend using the number of data cache refills satisfied from the L2 cache and/or the system (e.g., L3 cache or DRAM) with valid final status to measure data movements for the roofline analysis. During the investigation processes, we have used our homegrown test kernels (e.g., multiple kernels for computing geometric series, a 5-point 2D stencil kernel, and a 7-point 3D stencil kernel) to verify our procedure.

After intensive verification and validation processes, we have performed the proposed CrayPat-based roofline analysis for multiple kernels, HPC benchmarks and popular linear algebra libraries for HPC applications. Based on the "visually-intuitive" roofline performance plots, we present our discussion about performance characteristics of the employed applications. In addition, we present an example of the roofline analysis model for a performance projection on a new processor. Using CrayPat and the proposed method, we first analyzed the performance characteristics of several important groups of a production-level code on Blue Waters; then, we projected the groups' performance for target processors based on the characteristics and accumulated the projected performance to represent the overall performance of the code on the new processor. After having the projected overall performance, we performed the simulation on the target processor, and validated our performance projection approach. It turns out our performance projection method is a reasonable way to estimate overall performance of HPC applications on new processors.

We plan to perform similar analyses with GPU nodes and other processors (e.g., ARM), and we will share our developed python scripts for general roofline performance model, General Roofline Evaluation Gadget (GREG) via our public GitHub repository [21]. We hope our study will help other Cray users characterize and optimize their science and engineering applications via the proposed CrayPat-based roofline analysis at scale on their HPC systems. In addition, we hope the introduced methodology will be used to determine an optimal method for roofline analyses on other Cray systems.

Acknowledgment

This study is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

References

[1] W. Kramer, Keynote talk: Top500 versus sustained performance—or the top problems with the TOP500 list—and what to do about them. The 21st International Conference on Parallel Architectures and Compilation Techniques—PACT, Minneapolis, MN; 2012.

[2] J. Dongarra, M. A. Heroux, Toward a new metric for ranking high performance computing systems. Sandia National Laboratories Technical Report. SAND2013-4744: 2013;312.

[3] M. Adams, J. Brown, J. Shalf, B. Straalen, E. Strohmaier, S. Williams , HPGMG 1.0: A benchmark for ranking high performance computing systems. LBNL Technical Report, LBNL 6630E, 2014.

[4] J. Kwack, G. Bauer and S. Koric, "Performance Test of Parallel Linear Equation Solvers on Blue Waters – Cray XE6/XK7 system," Preceedings of the Cray Users Group Meeting (CUG2016), London, England, May 2016.

[5] G. Bauer, V. Anisimov, G. Arnold, B. Bode, R. Brunner, T. Cortese, R. Haas, A. Kot, W. Kramer, J. Kwack, J. Li, C. Mendes, R. Mokos, C. Steffen, "Updating the SPP Benchmark Suite for Extreme-Scale Systems," Proceedings of the Cray Users Group Meeting (CUG2017), Redmond, WA, May 2017.

[6] J. Kwack and G. Bauer, "HPCG and HPGMG benchmark tests on Multiple Program, Multiple Data (MPMD) mode on Blue Waters - a Cray XE6/XK7 hybrid system," Concurrency and Computation, Practice and Experience journal 30(1), 10.1002/cpe.4298, 2017.

[7] S. Williams, A. Waterman and D. Patterson, "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures," Communications of the ACM, 52(4), 65-76, 2009.

[8] A. Ilic, F. Pratas and L. Sousa, "Cache-aware Roofline model: Upgrading the loft," IEEE Computer Architecture Letters, 13(1), 21-24, 2014.

[9] B. Bode, M. Butler, T. Dunning, W. Gropp, T. Hoe-fler, W. Hwu, and W. Kramer (alphabetical), "The Blue Waters Super-System for Super-Science," Contemporary HPC Architectures, Jeffery Vetter editor. Sitka Publications, November 2012.Edited by Jeffrey S . Vetter, Chapman and Hall/CRC 2013, Print ISBN: 978-1-4665-6834-1, eBook ISBN: 978-1-4665-6835-8.

[10] W. Kramer, M. Butler, G. Bauer, K. Chadalavada and C. Mendes, "Blue Waters Parallel I/O Storage Sub-system," High Performance Parallel I/O, Prabhat and Quincey Koziol editors, CRC Publications, Taylor and Francis Group, Boca Raton FL, 2015, Hardback Print ISBN 13:978-1-4665-8234-7.

[11] Y. Lo, S. Williams, B. Straalen, T. Ligocki, M. Cordery, N. Wright, M. Hall, L. Oliker, "Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis," Performance Modeling, Benchmarking, and Simulation (PMBS), November 2014, doi: 10.1007/978-3-319-17248-4_7.

[12] Advanced Micro Devides, "BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors", 42301 Rev 3.14, January 23, 2013.

[13] J. Dongarra, M. Heroux and P. Luszczek "HPCG Benchmark: a New Metric for Ranking High Performance Computing Systems," Technical Report, Electrical Engineering and Computer Sciente Department, Knoxville, Tennessee, UT-EECS-15-736, November, 2015.

[14] S. Williams, "4th Order HPGMG-FV Implementation," HPGMG BoF, Supercomputing, November 2015.

[15] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," https://www.mcs.anl.gov/petsc, 2018.

[16] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," Parallel Computing, 32(2),136-156, 2006.

[17] X. S. Li and J. W. Demmel, "A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems," ACM Trans. Mathematical Software, 29(2),110-140, 2003.

[18] M. J. Noh, I. M. Howat, C. C. Porter, M. J. Willis, P. J. Morin, "Arctic Digital Elevation Models (DEMs) generated by Surface Extraction

from TIN-Based Searchspace Minimization (SETSM) algorithm from RPCs-based Imagery," American Geophysical Union, Fall General Assembly 2016.

[19] P. Morin, J. Pundsack, "The Polar Geospacial Center web page," https://www.pgc.umn.edu, 2018.

[20] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

[21] NCSA, "NCSA GitHub repository for GREG: General Roofline Evaluation Gadget," https://github.com/ncsa/GREG, 2018.

APPENDIX A

Section IV includes roofline performance analysis plots based on DCR_GOOD measurements. As a reference, we provide roofline plots based on L1_DCA measurements in Appendix A. Since L1_DCA always overestimates the byte transfer, L1_DCA-based CI values are much smaller than DCR_GOOD-based CIs in Section IV.

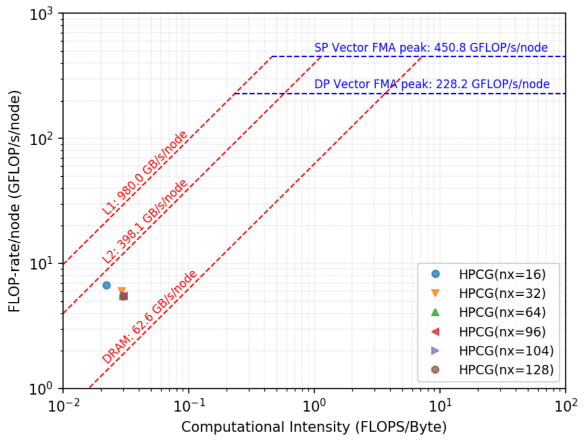


Figure 20. L1_DCA-based roofline analysis plots for HPCG on 4 XE nodes

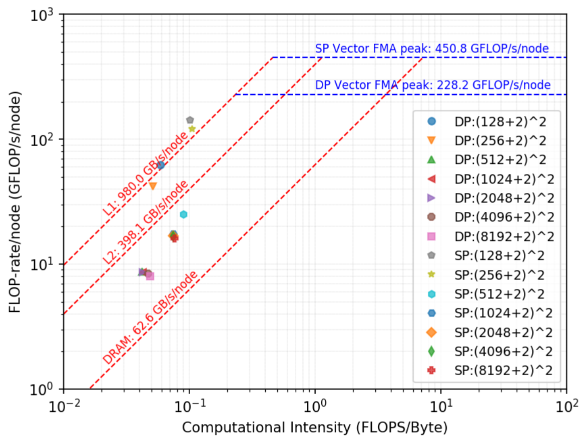


Figure 18. L1_DCA-based roofline analysis plots for 5-point 2D stencil kernels

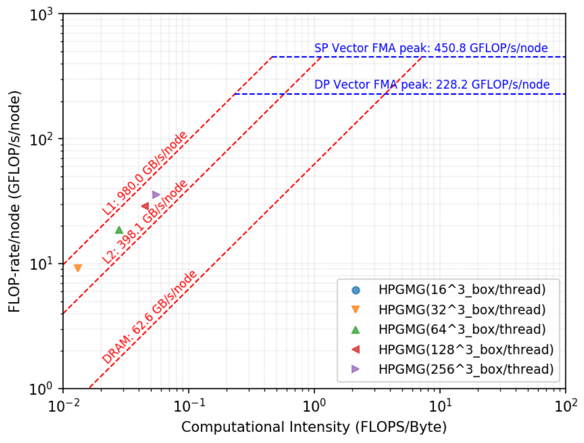


Figure 21. L1_DCA-based roofline analysis plots for HPGMG-FV on 16 XE nodes

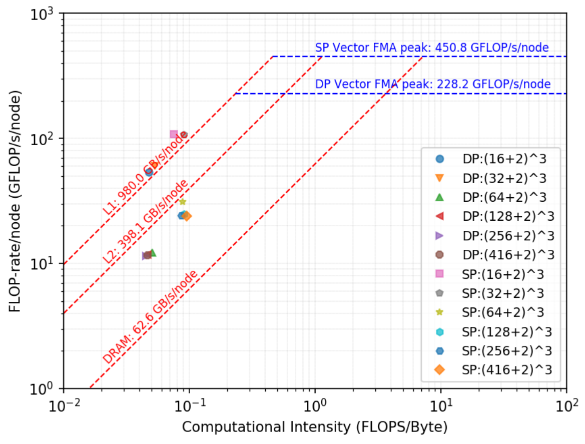


Figure 19. L1_DCA-based roofline analysis plots for 7-point 3D stencil kernels

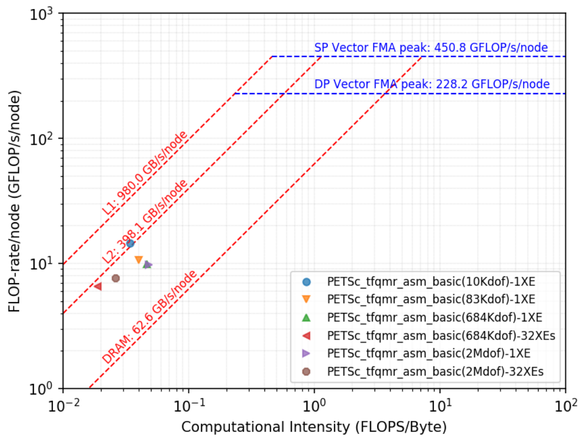


Figure 22. L1_DCA-based roofline analysis plots for PETSc tfqmr solver with asm_basic preconditioner on 1 XE or 32 XE nodes

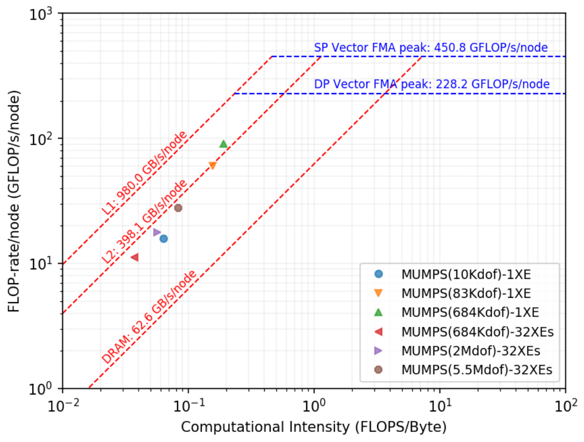


Figure 23. L1_DCA-based roofline analysis plots for MUMPS on 1 XE or 32 XE nodes

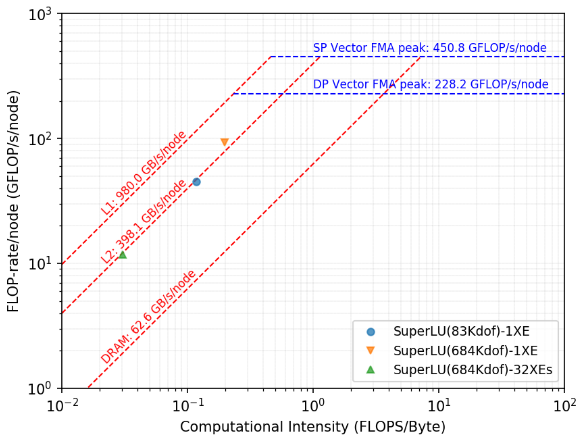


Figure 24. L1_DCA-based roofline analysis plots for SuperLU on 1 XE or 32 XE nodes