Parallel Application Performance on Two Generations of Intel Xeon HPC Platforms

Christopher H. Chang, Hai Long, Scott Sides, Deepthi Vaidhynathan, and Wesley Jones
National Renewable Energy Laboratory

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Technical Report
NREL/TP-2C00-64268
October 2015

Contract No. DE-AC36-08GO28308

Parallel Application Performance on Two Generations of Intel Xeon HPC Platforms

Christopher H. Chang, Hai Long, Scott Sides, Deepthi Vaidhynathan, and Wesley Jones
National Renewable Energy Laboratory

Prepared under Task No. 2C00.2050

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

Technical Report
NREL/TP-2C00-64268
October 2015

Contract No. DE-AC36-08GO28308

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Available electronically at SciTech Connect http:/www.osti.gov/scitech

Available for a processing fee to U.S. Department of Energy and its contractors, in paper, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
OSTI http://www.osti.gov
Phone: 865.576.8401
Fax: 865.576.5728
Email: reports@osti.gov

Available for sale to the public, in paper, from:

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312
NTIS http://www.ntis.gov
Phone: 800.553.6847 or 703.605.6000
Fax: 703.605.6900
Email: orders@ntis.gov

Cover Photos by Dennis Schroeder: (left to right) NREL 26173, NREL 18302, NREL 19758, NREL 29642, NREL 19795.

NREL prints on paper that contains recycled content.

Acknowledgments

This work was supported by the U.S. Department of Energy under Contract No. DE-AC36-08GO28308 with the National Renewable Energy Laboratory.

List of Acronyms

AVX	Intel Advanced Vector Extensions
AVX2	Intel Advanced Vector Extensions, second generation
B3LYP	Becke three-parameter exchange with Lee-Yang-Parr electron correlation density functional
BLAS	Basic Linear Algebra Subroutines
CPU	central processing unit
DDR3	double data rate random-access memory, Type 3
DIMM	dual in-line memory module
DP	double precision (referring to floating-point representation; 64-bit)
eV	electron volt
F77	Fortran 77
FAST	Fatigue, Aerodynamics, Structures, and Turbulence
FFT	fast Fourier transform
FlOps	floating-point operations
FMA	fused multiply-add
FP	floating point
GB	gigabyte (1,073,741,824 bytes)
GFlOp	gigaFlOp (1,073,741,824 FlOps)
GHz	gigahertz (1,073,741,824 cycles $\cdot$ s^{-1})
HPC	high-performance computing
ID	identification number
I/O	input-output
JPD	jobs per day
kB	kilobyte (1,024 bytes)
L1	Level 1 (referring to CPU cache)
L2	Level 2 (referring to CPU cache)
L3	Level 3 (referring to CPU cache)
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
LAPACK	Linear Algebra Package
MB	megabyte (1,048,576 bytes)
MKL	Intel Math Kernel Library
MM	matrix multiplication
MPI	Message Passing Interface
NREL	National Renewable Energy Laboratory
PW91	Perdew-Wang 1991 exchange-correlation density functional
SDRAM	synchronous dynamic random-access memory
SMT	simultaneous multithreading
TLB	translation look-aside buffer
VASP	Vienna Ab Initio Simulation Package

Executive Summary

Two generations of Intel Xeon microarchitecture were tested with a suite of microbenchmarks and application examples and compared to a current Ivy Bridge production node on the National Renewable Energy Laboratory's (NREL's) Peregrine high-performance computing (HPC) cluster. A primary observation from this study is that the benchmarks achieved peak performance on a node prior to utilizing all the cores available on a socket/node. This is largely because of limitations in application parallelism or resource contention among concurrent independent tasks. Additionally, hyperthreads were found to have a positive impact on performance in the general case. However, detailed specification of thread placement and pinning continues to outperform the general case, and overloading a core with multiple hyperthreads did not positively impact performance.

The observations offer some guidance to the procurement of future HPC systems at NREL. First, raw core count must be balanced with available resources, particularly memory bandwidth. The balance-of-system will determine value more than processor capability alone. Second, active exploitation of hyperthreading via oversubscription of cores continues to be largely irrelevant to the workloads that are commonly seen, and were tested here, at NREL. Finally, perhaps the most impactful enhancement to productivity might occur through enabling multiple concurrent jobs per node. For workloads comprised of many separate jobs that cannot each productively use a full node's resources, more might be achieved by doing multiple slightly slower things concurrently than single slightly faster things sequentially.

Table of Contents

List of Figures

List of Tables

1 Introduction

In this technical report we compare the performance characteristics among three systems:

1. A dual-socket, 2×12-core instantiation of the Intel Ivy Bridge processor currently in production use on the National Renewable Energy Laboratory's (NREL's) Peregrine cluster, referred to herein as "Ivy Bridge"
2. A single-socket, 4-core instantiation of Intel's "Haswell" microarchitecture, which was made available as a low-power test server platform, referred to herein as "Haswell4"
3. A dual-socket, 2×18-core instantiation of Intel's Haswell processor, representing a large core-count server platform, referred to herein as "Haswell36."

References to the package or chip rather than the system will use these same monikers, but should be clear from the context. Additional details about the three systems are provided below.

1.1 Constitution of the Benchmarks

Our chosen benchmark strategy relied on two classes of application. The first comprised microkernel benchmarks, distinguished by their relatively clear interpretability and focus on a single hardware performance feature (e.g., memory bandwidth, floating-point [FP] execution). We chose to focus on a small subset of benchmarks that have straightforward implications for performance. The STREAM Triad benchmark [1] is a standard test of memory bandwidth within the high-performance computing (HPC) community involving a simple calculation of $\widehat{\mathbf{A}} = q\widehat{\mathbf{B}} + \widehat{\mathbf{C}}$, where $\widehat{\mathbf{A}}$, $\widehat{\mathbf{B}}$, and $\widehat{\mathbf{C}}$ are arrays of adjustable size, and q is a scalar constant. Matrix multiplication (MM) served as our second microkernel benchmark; this focused on floating-point performance. MM is important as a central component of more complex linear algebra routines that are the bedrock (and usually rate-limiting factor) of scientific applications. It is a fundamental routine that has received extensive attention from hardware vendors and algorithm specialists to accelerate it via library calls. Thus, one can compare these nontrivial optimized algorithms to naive implementations that an application programmer might write to get a sense of how important the use of mathematics libraries is as well as the potential performance left "on the floor" by domain programmers not using them.

The second class of benchmarks included five scientific codes in production use at NREL paired with input problem sets representative of the typical application workload.

- The Vienna Ab Initio Simulation Package (VASP) is a plane-wave density functional theory code that is heavily used by the NREL materials science community.
- Gaussian is a molecular electronic structure program using nuclear-centered Gaussian function basis sets that is highly optimized for shared-memory parallelism and extensively used in high-throughput studies on Peregrine.
- Amber is a classical molecular dynamics package used for atomistic simulation.
- The Large-Scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is an atomistic and mesoscale molecular dynamics application used at NREL for polymer simulations.
- Fatigue, Aerodynamics, Structures, and Turbulence (FAST) is an NREL-written engineering code for simulating the coupled dynamic response of wind turbines.

These five applications represent sufficient utilization of the Peregrine cluster to draw some conclusions about the performance of these codes on currently available Intel central processing units (CPUs) for typical production workloads at NREL.

2 Processor and System Detail

2.1 Ivy Bridge

The E5-2695v2 Xeon processors that form the backbone of Peregrine are 12-core Intel chips running at 2.4 GHz. Each core has dedicated access to 32 kB each of 8-way associative data and instruction L1 cache, and 256 kB of 8-way associative L2 cache. L3 cache (30 MB, 20-way associative) is shared among the 12 cores. The nodes are dual-socket, with four memory channels (two per socket) able to sustain 60 GB/s peak total throughput. The Ivy Bridge architecture has Advanced Vector Extensions (AVX), which enable each core to carry out 4 double-precision (DP), FP operations in a single instruction. Parallel FP units permit 2 such operations per clock cycle per core, so nominal node-level DP throughput is 24 cores $\times$ (2 $\times$ 4 DP/(core·clock)) $\times$ (2.4 $\times$ 10^9 clocks/sec) = 460 GFlOps.

2.2 Haswell

Although the cache structure is largely the same between the two chip generations, the Haswell core differs from Ivy Bridge in several ways that might be expected to improve performance on scientific applications. First, the new AVX2 instruction set and supporting hardware include a fused multiply-add (FMA) instruction, which effectively doubles theoretical peak FP performance. Although vector width expansion between AVX (which Ivy Bridge has) and AVX2 is only for integer instructions, other features of AVX2 such as gather instructions for sparse memory accesses can improve FP throughput as well. L1 and L2 caches share sizes and associativities with Ivy Bridge. However, the L3 cache size on the 18-core E5-2699v3 chip is 45 MB compared to the 30 MB on Ivy Bridge, which maintains the MB:core ratio constant at 2.5 MB/core. The low-power 4-core Core i7-4700EQ has a more modest 6-MB 12-way set associative L3 cache for an MB:core value of 1.5 MB/core. Significantly, the Haswell core doubles the L1 and L2 cache load and store (32 bits/cycle) and read (256 bits/cycle) port widths relative to Ivy Bridge [2]. In addition to AVX2, an increase in Haswell's reordering buffer to 192 entries from 168 in Ivy Bridge offers an enhanced, out-of-order execution, which should have a generally beneficial effect on all workloads. Scheduling has also been enhanced to 60 entries from 54, and execution units can support 8 μops/cycle compared to 6 μops/cycle in Ivy Bridge. Specific to vectorizable FP workloads, whereas Ivy Bridge has one 256-bit multiplier on one port and one 256-bit adder on another, Haswell replaces these with two fully pipelined FMA units (which are capable of pure FMUL and FADD operations as well). Although theoretical throughput is not changed when FP operation types are not distinguished, the new units might offer better throughput on realistic workloads—for example, one might have differing numbers of scheduled multiplications and additions, so that an Ivy Bridge unit might be left idle for one or more cycles. A final distinction between Haswell and Ivy Bridge with significant potential performance implications is the addition in Haswell of an address generation unit for store operations. This change effectively doubles the memory storage bandwidth to 96 bytes/cycle, and permits overall 72 loads and 42 stores to be in flight simultaneously.

Relative to Ivy Bridge, the Haswell core architecture has improved FP operations intrinsic to technical computing (AVX2 has FMA and gather, and 2 FMA instead of FMUL+FADD executing units) as well as applicable to general computing (out-of-order scheduling, memory bandwidth). One thus expects a boost in performance for technical applications, and the remainder of this document seeks to test and quantify that expectation.

Table 1 presents a comparison between the Ivy Bridge and Haswell test processors, including the key features noted above in addition to ones related to memory subsystem performance.

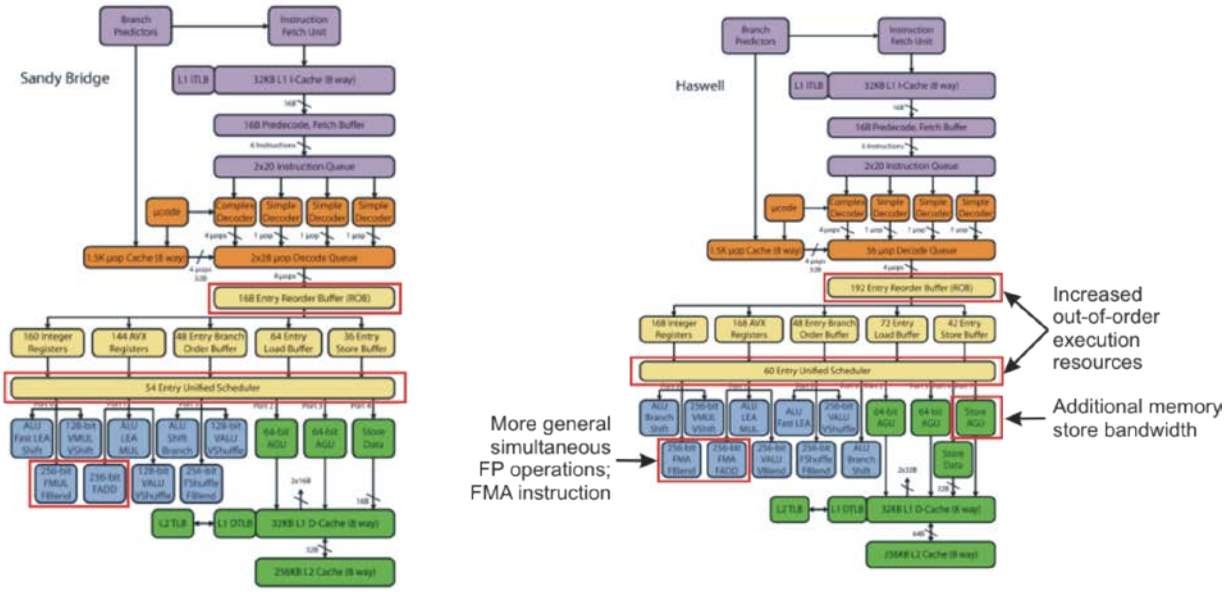

Figure 1. Significant performance-related differences between the Haswell and Ivy Bridge architectures. ***Image adapted from [3]***

Table 1. Comparison Among Test Processors of Key Architectural Features Impacting Performance

	E5-2695v2	Core i7-4700EQ	E5-2699v3
µops/clock	6	8	8
µop scheduler depth	54	60	60
Reorder buffer depth	168	192	192
AVX	ADD+MULT	2×FMA	2×FMA
Cores sharing cache (L1/L2/L3)	1/1/12	1/1/4	1/1/10+8[b]
Cache sizes (L1 data/instruction, L2, L3)	32 K/32 K,[a] 256 K, 30 M	32 K/32 K, 256 K, 6 M	32 K/32 K, 256 K, 25+20 M[b]
Cache associativities	8/8/20	8/8/12	8/8/20
L1 cache bandwidth (bytes/clock)	48	96	96
L2 cache bandwidth (bytes/clock)	32	64	64
L1 TLB (instruction) entries×size/associativity	128×4 K/4-way 8×2 M/full	128×4 K/4-way 8×2 M/full	128×4 K/4-way 8×2 M/full
L1 TLB (data) entries×size/associativity	64×4 K/4-way 32×2 M/4-way 4×1 G/4-way	64×4 K/4-way 32×2 M/4-way 4×1 G/4-way	64×4 K/4-way 32×2 M/4-way 4×1 G/4-way
L2 TLB entries×size/associativity	512×4K/4-way	1024×(4 K or 2 M)/ 8-way	1024×(4 K or 2 M)/ 8-way
Load/store buffer depth	64/36	72/42	72/42

[a] Cache and translation look-aside buffer sizes are presented as K = kilobytes, M = megabytes, and G = gigabytes.
[b] The Xeon E5-2699v3 processor's L3 cache is connected to the cores via two asymmetric unidirectional ring networks, with 10 cores on the first ring and 8 cores on the second ring. The partitioning between the two rings is denoted with a "+" sign.

2.3 Test Systems

1. The Ivy Bridge test system was a dual-socket motherboard populated with two 12-core Xeon E5-2695v2 packages running at 2.4 GHz. Memory on the test nodes was 32 GB of double data rate random-access memory, Type 3 (DDR3) synchronous dynamic random-access memory (SDRAM), configured as 8×4 GB dual in-line memory modules (DIMMs) arranged with 1 DIMM per channel to maximize potential bandwidth. The operating system was CentOS Linux 6.3 with kernel version 2.6.32. All tests were performed on a single node of the Peregrine computing cluster and did not exercise the Infiniband connectivity.

2. The 36-core Haswell test system ("Haswell36" below) comprised a dual-socket motherboard with two 18-core Haswell Xeon E5-2699v3 packages running at 2.30 GHz. Each socket was connected to 4×8 GB DIMMs, for a total of 64 GB (1 DIMM per channel).

3. The 4-core Haswell test system ("Haswell4" below) was an HP Moonshot m710 server with a single-socket motherboard hosting a 4-core Core i7-4700EQ chip (2.40 GHz), characterized as a "low-power" system. Memory was configured in 4×8 GB DIMMs, with one per channel.

Both Haswell systems ran Linux kernel 2.6.32. Where indicated, the "Full Node Ratio" is a comparison of performance on analogous workloads between the two test node types when fully loaded with work—for example, a single job on each with thread or Message Passing Interface (MPI) rank number equal to the number of available cores.

3 Benchmark Comparisons

3.1 MicroBenchmarks

3.1.1 STREAM

STREAM [1] Triad tests memory bandwidth and achievable FP performance. The primitive operation in Triad is the multiplication of an array element (C_i) by a constant scalar, the addition of an element from a second array at the same index (B_i), and the population of a third array (A) by the result. It has a regular access pattern well suited to compiler optimizations. STREAM version 5.10 was used for this benchmark. The C compiler used for this benchmark was Intel version 14.0.2. The binary was compiled with

```
-mcmodel medium -O2 -openmp -mavx
```

flags. The size of the arrays in Triad is set at a compile time by defining the STREAM_ARRAY_SIZE variable. The STREAM_ARRAY_SIZE definition specifies the number of elements in a single array. For the reported tests, the arrays A, B, and C were sized to 6 GB each (-DSTREAM_ARRAY_SIZE = 100 663 296 DP elements). The total memory occupied was therefore 18 GB. The environment variable OMP_NUM_THREADS was used to vary the number of OpenMP threads used. When relevant, the pinning of threads to cores was controlled by the KMP_AFFINITY environment variable. Threads were pinned to minimize cache contention (i.e., KMP_AFFINITY=scatter).

The STREAM benchmark was run multiple times, and the average is reported in Figure 2. A comparison between a 36-core Haswell node and a 24-core Ivy Bridge node is shown. Ivy Bridge performed better when the threads were pinned to the cores when it neared full node occupancy. The deviation of the observed maximum bandwidth from the theoretical peak on both systems is curious and notable, given the simplicity of the microbenchmark. Although we do not offer a hypothesis for the underlying reason for this behavior, we do note its potential impact on applications limited by data movement.

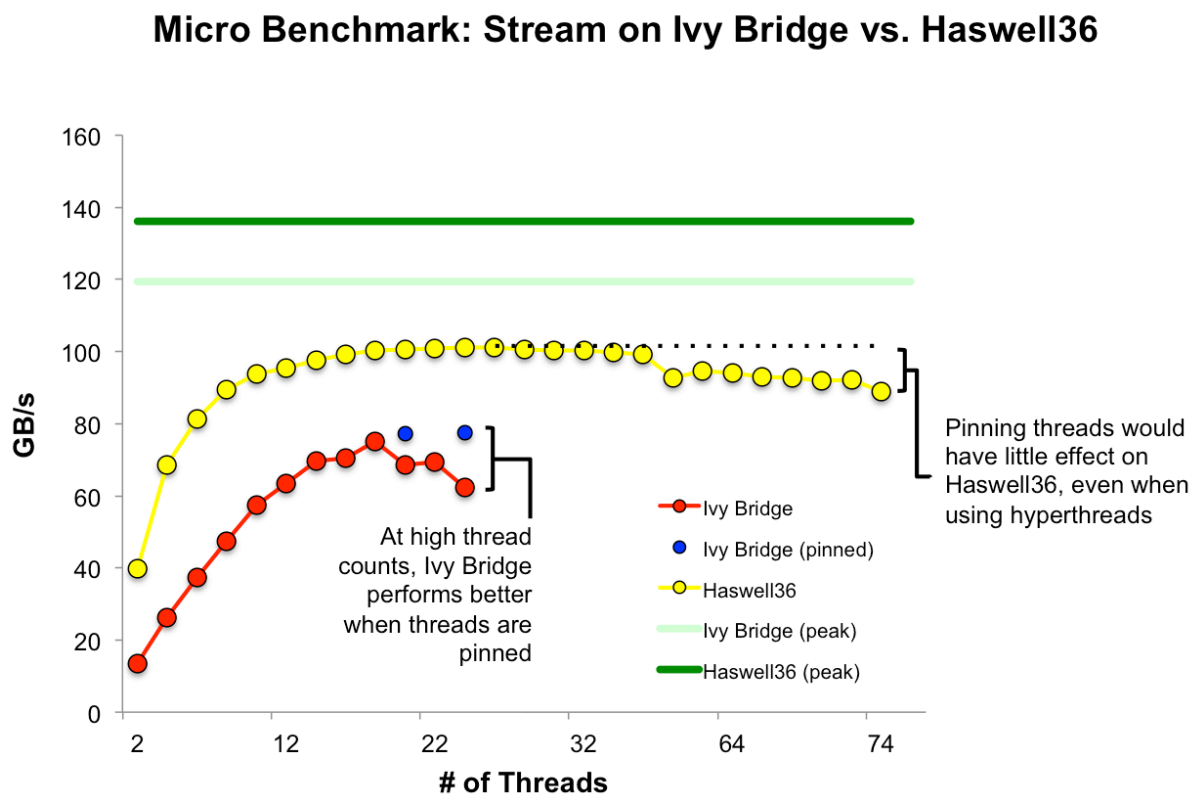

Figure 2. Microbenchmark: STREAM—Ivy Bridge and Haswell36

3.1.2 Matrix Multiply

The matrix multiply code [4] was used for this purpose. The compiler used for this benchmark was the Intel C compiler version 14.0.2. Two algorithms were tested: a "näive" implementation with simple loops written at the application level and an optimized version accessed through calls to the highly optimized Intel Math Kernel Library (MKL). The flags used to compile both the naive and optimized matrix multiply binaries were as follows:

```
-xhost -mkl -O3 -openmp -Wno-unknown-pragmas -std=c99 -vec-report2
```

In the "naïve" build, the -mkl flag affects timing functions only. The dimension of the matrix, the number of threads, and the number of iterations were variables under study, and passed as arguments to the executable at runtime. The matrix size used here was fixed at 10000×10000.

The number of threads was varied, and the average GFlOps/s is reported in Figure 3. The improvement in efficiency by using the MKL routines was not as effective when hyperthreads[1] were used. It was only

[1] Hyperthreading officially occurs in the context of Intel Corporation's "Hyper-Threading Technology," and denotes their particular implementation of simultaneous multithreading (SMT). SMT is a general mechanism to exploit the superscalar architectures of modern processors by permitting the interleaving of instructions and data from two distinct logical threads of execution. To accomplish this, the hardware resources are presented to the operating system (OS) as two distinct logical processors, upon each of which a single execution thread can be run. We use "hyperthreading" to mean the presentation of multiple (here, two) logical processors to the OS, "hyperthread" to mean the second execution thread scheduled on a physical core by virtue of hyperthreading, and "thread" in its standard usage.

approximately two-thirds the improvement achieved with 36 cores. The Haswell36 system performed better than the Ivy Bridge with a full-node ratio of 2.193.

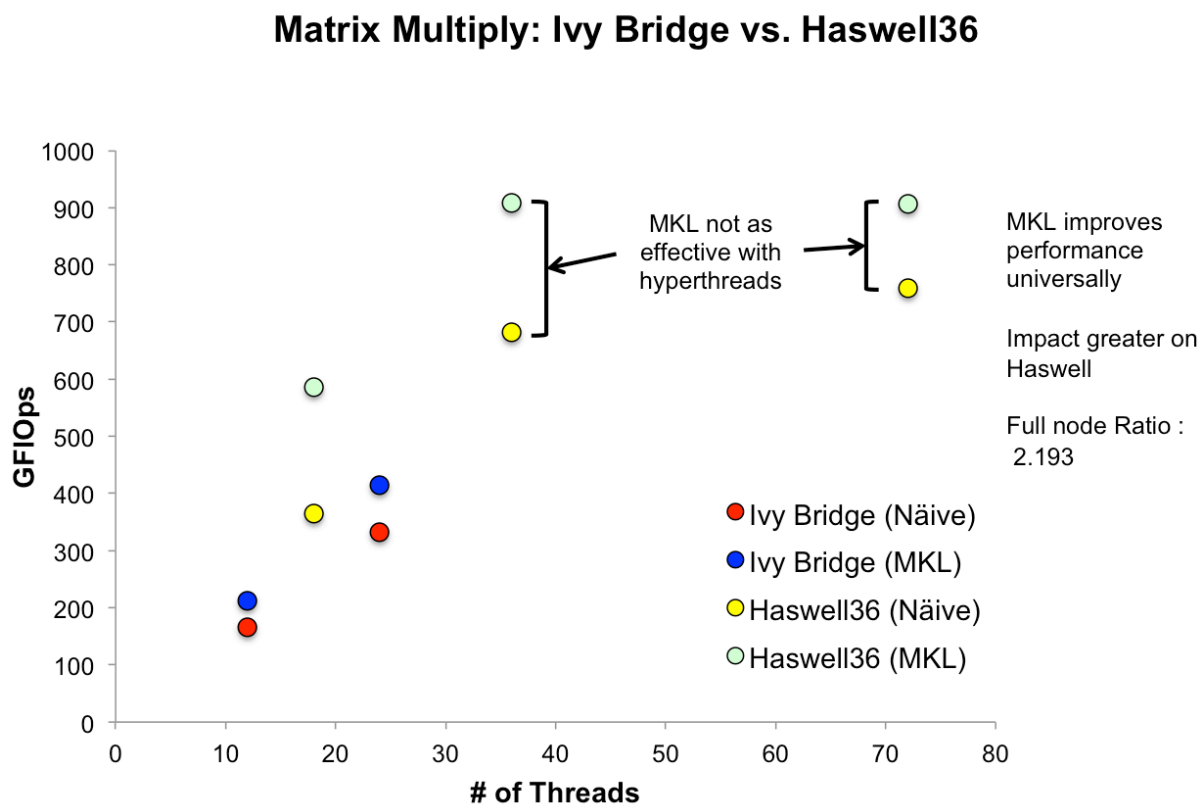

Figure 3. Matrix multiplication comparing the performance of Ivy Bridge to Haswell36 with respect to thread count and algorithm, multiplying 10000×10000 dense matrices

3.2 Application Benchmarks

Unless noted, application runs employing multiple threads in a single run did not pin the threads explicitly, and relied on the default policy of the associated runtime.

3.2.1 VASP

VASP [5-8] is a plane-wave density functional theory code reflective of materials science workloads at NREL. Benchmarking used VASP version 5.3.5, built with the Intel 14.0 compiler suite and Intel MPI 4.1.3. Intel MKL was used to supply BLAS [9, 10], LAPACK[11], and FFT[12] routines. A dual-path, auto-dispatch executable was built using the

```
-axCORE-AVX2,AVX
```

compiler directive. The binary executable was linked against the sequential MKL routines, and is thus parallel only at the MPI rank level—i.e., no threading was employed in the primary application or secondary libraries. Optimization levels and code standards were as expressed by the developer-supplied Makefile. The test computation was a generalized gradient approximation, density functional theory,

single-point calculation on a 40-atom, body-centered cubic In_2O_3 unit cell using a 4×4×4 Gamma-centered k-point mesh with 8 irreducible points; a 48^3 FFT mesh and 96^3 augmentation charge FFT mesh; 400 eV plane-wave cutoff; the Perdew-Wang 1991 exchange-correlation density functional (PW91) exchange-correlation model; an all-bands-simultaneously updated preconditioned conjugate gradient algorithm for electronic updates (IALGO=58, or ALGO=All), with step size set to 0.1; and, default task parallelization (each band on a single core, parallelization over bands).

Results are presented as the number of jobs that could be completed per day, given the parallelization and degree of node packing. "Job Load" refers to how many jobs were running concurrently on separate core sets. A "Single Job Load" thus means that only one possibly MPI-parallel job was running at a time. Thus, if N cores were available on the system and the test job were n-way parallel, (N-n) cores were left idle. For "Fully Packed Node" tests, n was chosen as an integer divisor of N, such that N/n concurrent jobs of n ranks each filled the cores and left none idle. Throughputs were calculated assuming that a single cluster user had sole possession of the node during the modeled runtime, regardless of how many jobs were running concurrently on the node, or how parallel each job was. This presentation was chosen to reflect the scientific productivity possible, and to permit a straightforward comparison between throughput-style (e.g., N concurrent single-process jobs on N cores) and performance-style (e.g., 1 N-rank job on N cores) computing. Ranks were mapped as designated, where "Scatter" maximized the NUMA distance, and "Compact" assigned MPI ranks to consecutive core IDs.

Figure 4 illustrates potential VASP throughput given a single parallel job running per hardware node. The approximate initial slopes drawn as solid lines illustrate the performance enhancement provided by the architectural features of the Haswell core compared to the Ivy Bridge. Taking a reference of 600 jobs/day, Haswell reached this target at 5 cores (600/5 = 120) and Ivy Bridge at approximately 8 cores (600/8 = 75), for a rough enhancement factor of 120/75 = 1.6. However, deviation from linear scaling is seen at approximately the same number of cores, given a single node for each test. This suggests that productive usage of the increased number of Haswell cores available may present a challenge given the node-level architectural features.

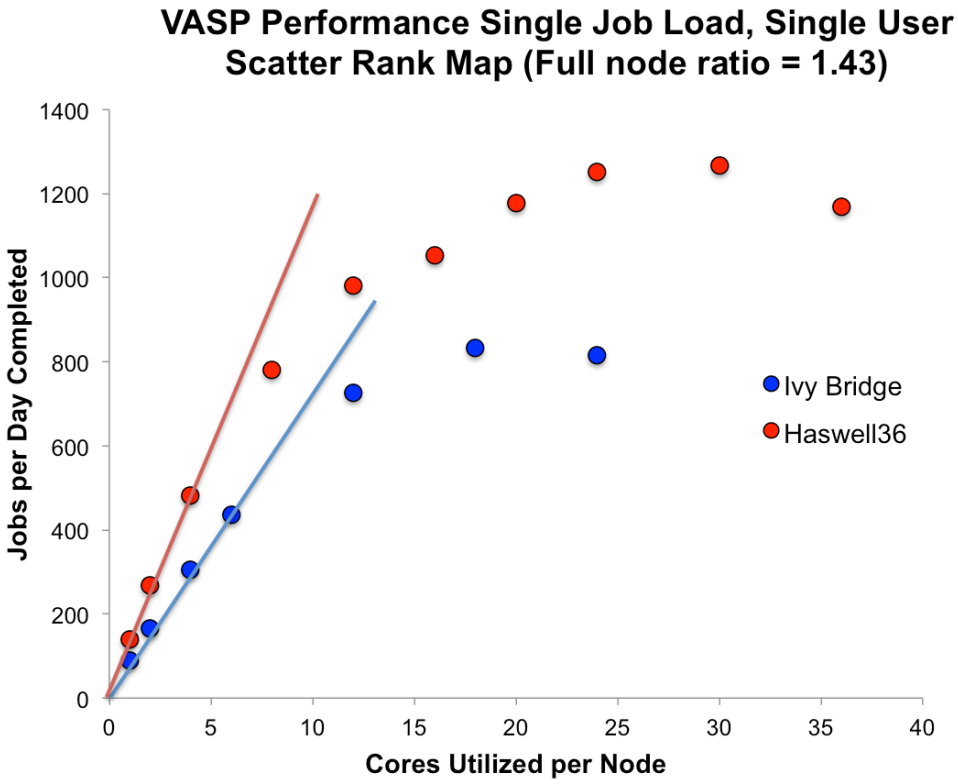

Figure 4. VASP throughput, single job per node, Haswell36 compared to Ivy Bridge

Figure 5 illustrates the throughput achievable by exploiting two levels of concurrency: the MPI parallelism within a single job, and packing multiple jobs onto a node at once. Throughput is therefore taken as the product of the job-level concurrency per node and the average sequential rate of job completion, where "sequential" refers to the job workflow (i.e., one is not started until a previous one finishes) and not MPI concurrency within a job. Thus, this reflects productivity for a single user fully packing a node with potentially multiple jobs. Running only serial jobs, one sees an almost 50% improvement with Haswell compared to Ivy Bridge running single-core jobs (i.e., where inter-task communication is not a factor), reflecting the increased number of cores per node for the Haswell36 system. Notably, it appears that resource contention balances the architectural enhancements of Haswell in this scenario. As more cores are deployed per job, scaling issues began to impact overall throughput negatively. With each fully packed node running a single parallel job, the combination of potential contention and sublinear scaling led to the user gaining less than the ratio of cores between the nodes.

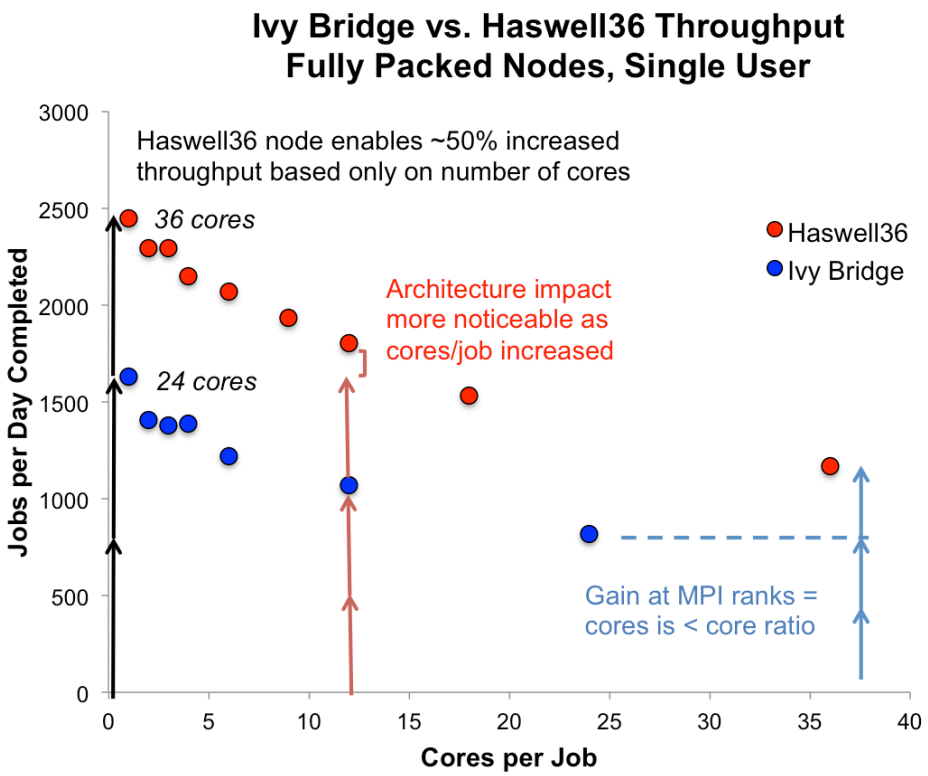

Figure 5. VASP throughput, fully packed nodes

Figure 6 shows the impact of common task mapping patterns on VASP throughput. In this analysis, a single user had dedicated access to a node and ran a single job at a time. "Compact" mapping placed MPI ranks on consecutive core numbers, such that the first package of the dual-socket node was completely filled while the second package sat idle. "Scatter" placed MPI ranks alternately on the different packages, pinning MPI ranks to cores with the least degree of resource contention possible. Ranks were mapped manually. Unsurprisingly, compact mapping showed an intermediate leveling off of performance between 8 cores and 16 cores, then an increase in slope from 16 cores to 20 cores as the new package's resources became available. Scatter mapping showed better throughput over the whole range studied (except for 36 cores, of course, when the mappings were equivalent from a hardware perspective).

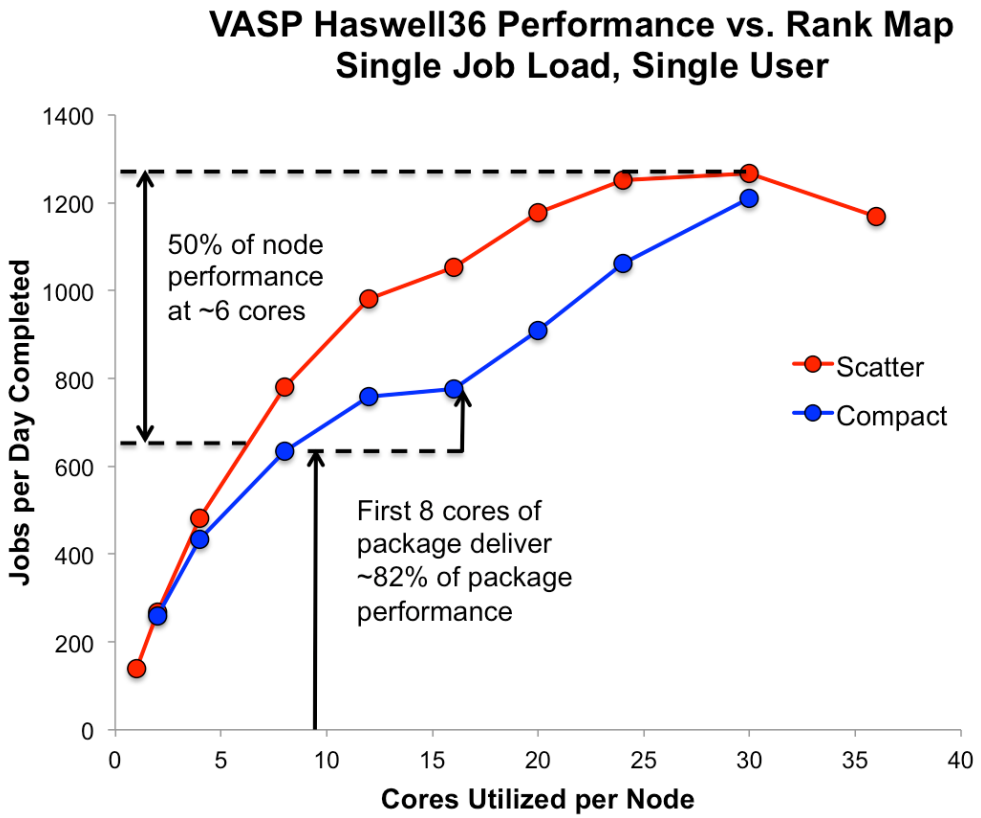

Figure 6. VASP throughput, impact of task mapping to cores

Interestingly, scatter mapping achieved half of the potential node performance in only 6 cores utilized (i.e., 3 of 18 cores per package). With compact mapping, filling half of the cores on the first package achieved 82% of the potential performance. Both observations strongly imply that the test node architecture has a large imbalance between the available CPU cores and the node-level system resources. From a price-performance perspective, the diminishing returns seen would imply that given fixed system resources on offer (e.g., memory bandwidth and latency), one might benefit more by spending on additional nodes with fewer cores, rather than nodes with very high core counts.

In Figure 7, a single package was compared for the low-power and 18-core Haswell chips. The slopes noted in the figure represent the gain in going from one tested core count (i.e., assigned MPI ranks) to the next. As might be expected, the 36-core node may have greater system resources per chip, so for two versus one core, the Haswell36 node had a larger initial slope than the Haswell4. However, in looking at the difference in throughput gains between the initial (2 versus 1 core) and almost fully loaded nodes (4 versus 2 cores on Haswell4, 16 versus 12 cores on Haswell36), one sees that much less is gained from packing the 18-core chip (4.5 jobs per day [JPD]/core at sub-saturating load) than from packing the 4-core chip (34.5 JPD/core). Again, this is interpreted as a balance-of-system outcome and that rapidly decreasing value is achieved in VASP workloads as core count per chip is increased.

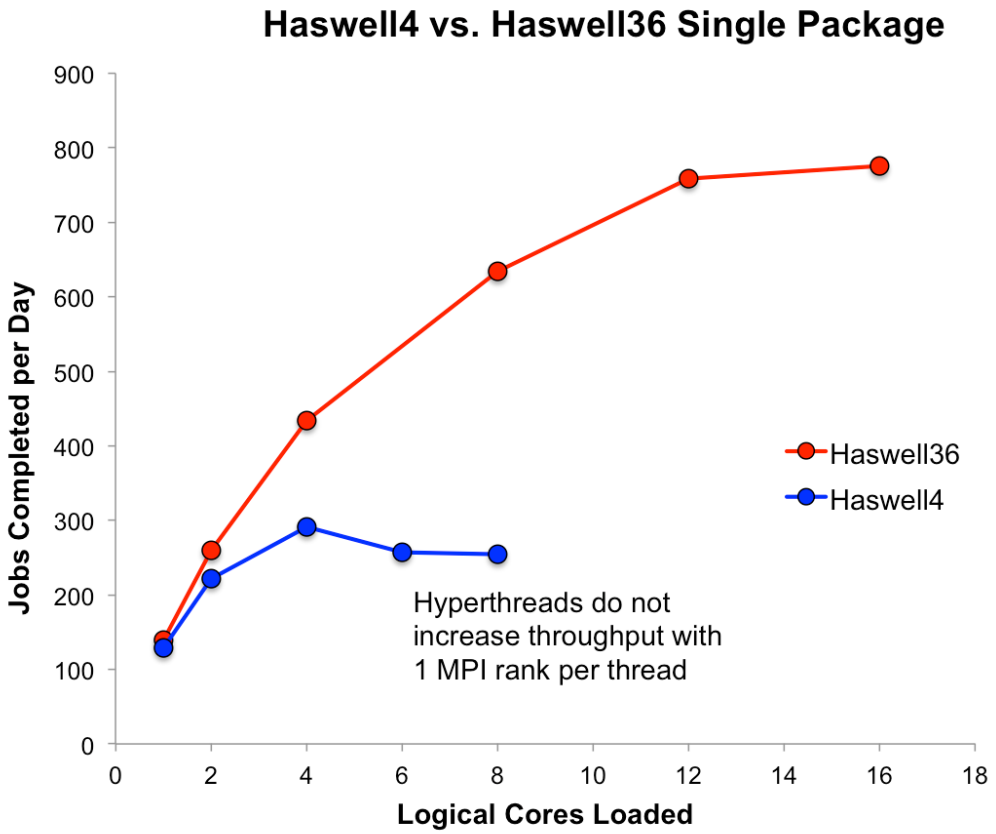

Figure 7. VASP throughput, Haswell4 compared to Haswell36 configurations

Overall, the test VASP workload showed an approximately 60% acceleration from the intrinsic difference in architecture between Haswell and Ivy Bridge. However, node-level resource limitations appeared to make that difference less impactful both as serial jobs packed a node, and as the degree of parallelism per job increased. Despite the potential doubling of FP performance offered architecturally, the actual throughput increase is projected to be approximately 1.5× under ideal conditions, and much diminished returns realized as resource contention and parallel inefficiencies come into play.

3.2.2 Gaussian

Gaussian [13] is a quantum chemistry program widely used for electronic structure calculations at NREL. This application is able to parallelize computing over multiple different hosts via Linda [14], and multiple cores within a host via threading. Gaussian 09 revision D.01 was compiled with the Portland Group F77 compiler version 12.10 for benchmarking. The test job was a density functional theory geometry optimization on a 40-atom oligothiophene model with a B3LYP/6-31G(d) model chemistry. Most Gaussian jobs at NREL are single-node calculations. In addition, a great number of Gaussian jobs at NREL are for high-throughput screening. As a result, the total number of jobs completed within a certain time is more important than the absolute calculation speed of a single job. Therefore, if not noted, the Gaussian benchmarking tasks reported in Figure 8 were done by running two identical geometry optimization jobs simultaneously on a single node. For example, the 16-thread data point was generated by running two 8-thread jobs on the same node concurrently and measuring the total number of jobs that could be completed in this manner within a day. For the 24-thread data points of Ivy Bridge and 36-thread data points of Haswell36 nodes, we also ran tasks with one job per node and four concurrent jobs per node. The number of concurrent jobs for these tasks is shown in Figure 8.

For Gaussian runs, the performance for both types of Haswell nodes was approximately 20% better than that of the Ivy Bridge nodes when running the same tasks. Thus, the difference between the Haswell and Ivy Bridge nodes for Gaussian was the smallest for all applications tested in this manuscript. However, it should be noted that the Gaussian binary used in this research was optimized for the Ivy Bridge processors, and it therefore may not be able to unleash the full potential of Haswell processors. Throughput on Haswell showed good scaling with shared-memory parallelism until approximately 18 threads per job, at which point throughput fell dramatically, implying that NREL's typical high-throughput Gaussian calculations would not benefit from using more than approximately 20 threads on this architecture. To further investigate this pattern, we ran four and one concurrent jobs and used up all available cores for both the Ivy Bridge and Haswell36 nodes. For the Ivy Bridge node, the optimal performance was achieved by running two 12-thread jobs concurrently. The performance of one job with 36 threads on the Haswell36 node was substantially slower than running on 9 cores or 18 cores, which likely reflects a limit to strong scaling of this particular job, more than an intrinsic limitation of the hardware. However, given this job's representation of typical high-throughput calculations at NREL, this observation does illustrate that job packing at lower levels of parallelism is a potential strategy to increase scientific throughput when compared to attempting greater levels of parallelism per job. In addition, for both tests of the Haswell nodes, hyperthreads did not enable higher throughput. The hardware-level impact of hyperthreads may be seen by comparing the point labeled "1" at 36 threads and the unlabeled point at 72 threads (2 jobs per node = 36 threads per job) in Figure 8. It appears that individual job performance dropped by a factor of 6 (an approximate threefold drop in overall throughput times a difference of two-fold in node packing), possibly because of increased resource contention.

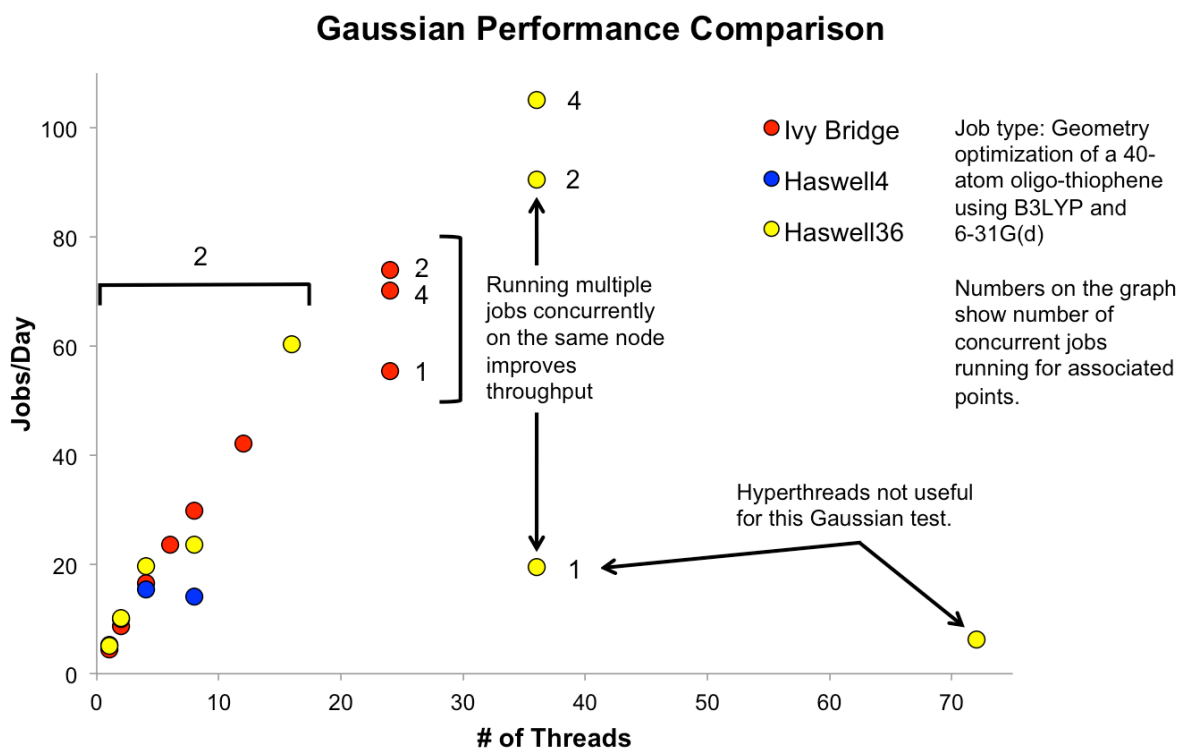

Figure 8. Gaussian throughput, multiple jobs per node, comparison among the Haswell36, Haswell4, and Ivy Bridge systems

3.2.3 LAMMPS and Amber

LAMMPS [15] and Amber [16] are widely used molecular dynamics program packages. For both LAMMPS and Amber benchmarks, only MPI parallelism was employed during the simulations, and ranks were not explicitly mapped to cores. The binaries used were the LAMMPS August 14 2013 release and Amber 14.0. Both were compiled using the Intel version 13.1.3 compiler with Intel MPI version 4.1.1.

The test model used for the LAMMPS benchmarking was a 5.2-nm cubic box of acetonitrile with 9,996 atoms. The force field used was the Gaff force field, and the Particle Mesh Ewald technique was used to compute the electrostatic potentials with a cut-off distance of 1.5 nm. The simulation ran for 50,000 steps with a time step of 0.5 fs, and the performance was measured by the number of jobs that could complete within a day.

The left panel of Figure 9 presents the performance of LAMMPS with one job running per node. For jobs that had an identical number of ranks, the Ivy Bridge node performance was only approximately 60% of the Haswell performance. Because the simulation box was cubic, for jobs with 8 threads, the system domain was decomposed into a 2×2×2 cubic grid for each MPI rank. But for jobs that had another number of threads tested in this section, no cubic grid could be decomposed. For example, for the 16-thread jobs, the grid was 4×2×2. Obviously, the cubic grid resulted in slightly better performance, and, thus, jobs with 8 ranks on the Ivy Bridge and Haswell36 nodes formed kinks on the respective performance plots. Although strong scaling was still seen up to 36 cores on Haswell36, exploiting hyperthreads had a strong negative impact on throughput. This is in contrast to Haswell4, in which hyperthreads improved throughput.

The test job used for the Amber benchmarking was a large cellulase system containing 180,038 atoms with a box size of 10 nm×10 nm×20 nm. The force field used was the CHARMM36 force field, and the Particle Mesh Ewald technique was used to compute the electrostatic potentials with a cutoff distance of 0.8 nm. The simulation ran for only 1,000 steps with a time step of 1.0 fs, and the performance was measured by the number of such jobs that could complete within a 24-hour day.

Figure 9 shows the results of Amber on the three test systems in the righthand panel. Similar to the LAMMPS case, for jobs that had an identical number of ranks, the Ivy Bridge node performance was approximately 70% that of the Haswell36 node. The throughput plot of the Haswell36 showed performance saturation at approximately 18 threads. Additional threads began to become less valuable to the throughput of single jobs. Although core oversubscription via hyperthreading did not increase performance on Haswell36, it did so for Haswell4, perhaps reflecting the different levels of resource contention on the two node types.

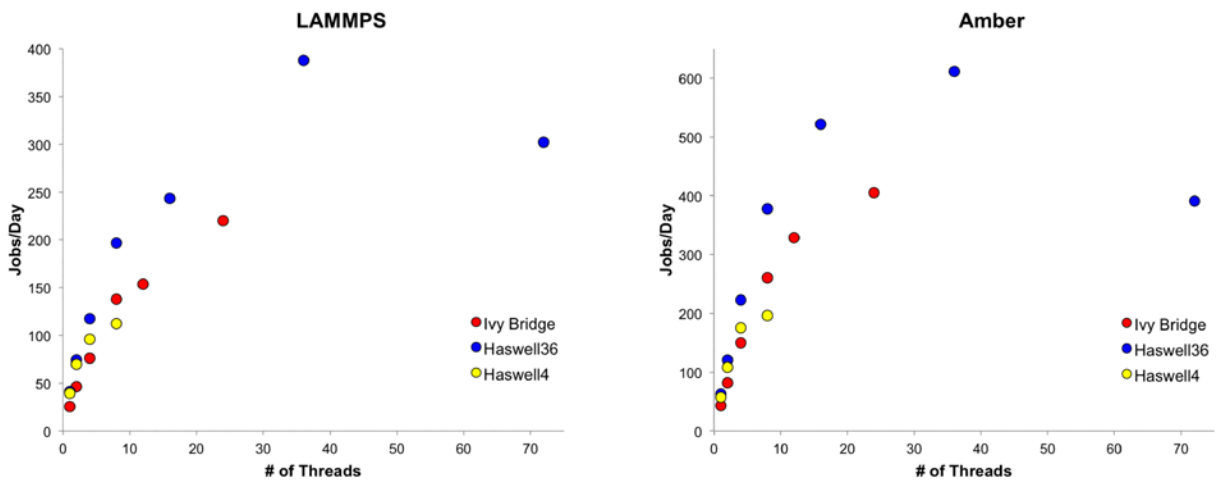

Figure 9. LAMMPS and Amber throughputs, single job per node, comparison among the Haswell36, Haswell4, and Ivy Bridge systems

3.2.4 FAST

The FAST code [17] is a computer-aided engineering tool for wind turbine modeling. The FAST modular framework enables the coupling of code modules that simulate aerodynamic, hydrodynamic, servo-dynamic, and structural dynamic code components. FAST is a high-throughput application because the models are required to run efficiently and quickly for a large set of environment conditions and throughout many types of operational conditions. Although massively parallel simulations of coupled fluid and structural dynamics could model similar systems, the time required to perform a calculation for a single set of parameters is prohibitively large. The algorithms used by the individual components in FAST are based on specific, advanced engineering models with assumptions and simplifications germane to the wind turbine field applications of interest for NREL researchers and customers. For example, two of the modules that FAST combines are AeroDyn [18, 19], a rotor aerodynamics module, and HydroDyn [20-22], a platform hydrodynamics module for offshore systems.

Results are presented as the number of jobs that could be completed per day, given parallelization and degree of node packing. FAST runs in a high-throughput mode for optimizing wind turbine parameters. A "FAST job" is defined as 10,000 realizations certification tests 1–17 distributed with the FAST source code [23]. Parallelization is achieved by spreading concurrent realizations throughout the available logical cores on a node, up to the number of cores being tested. Given the time-to-solution, t, of a set of N realizations on N cores, the number of jobs that could be completed per day was calculated as

$$\left(\frac{N \text{ realizations}}{t \text{ seconds}} \times \frac{86400 \text{ s}}{\text{day}} \times \frac{1 \text{ job}}{10000 \text{ realizations}}\right).$$

The results presented here were obtained by building the FAST source code with version 13.1.3 of the Intel compilers. The FAST simulations required relatively small amounts of memory and input/output compared to the other HPC applications described in this report. The bulk of the node resources represented the FP operations needed to implement the algorithms in each of the modules and the overhead used by the FAST coupling framework.

In Figure 10, the performance of the jobs/day completed is shown compared to the number of cores used on a node. Each realization of the FAST suite of simulations was assigned to a particular core. These tests are referred to as "pinned." Processes were assigned to consecutive core IDs; thus, for four simulations, logical core IDs 0, 1, 2, and 3 were utilized. The performance for the Haswell36 system showed a near linear speedup as the number of concurrent jobs was increased, up to 36 cores, at which point the node was fully packed and each core occupied with work. The addition of a single task to engage the hyperthread on core 0 led to a sharp drop in performance and only a moderate gain with increasing thread number. The Haswell4 system showed similar performance characteristics as logical core occupation crossed into the hyperthreading regime. The Ivy Bridge node had hyperthreading disabled and showed a linear speedup across all tested job counts.

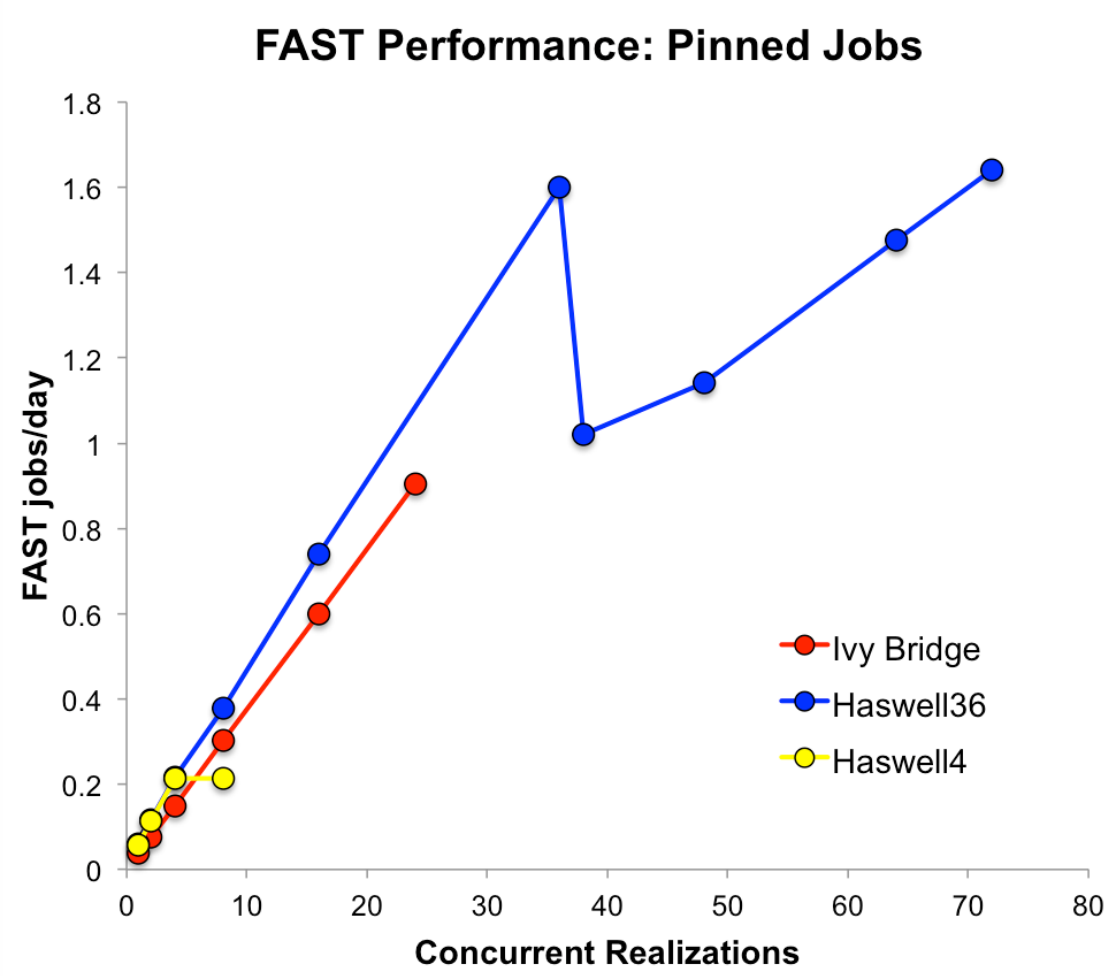

Figure 10. FAST throughput, single job per node, comparison among the Haswell36, Haswell4, and Ivy Bridge systems

Figure 11 shows the performance of jobs/day completed compared to the number of concurrent processes requested on a node. On the left, the mapping of run index to processor ID is listed as "unpinned" and means that the `taskset` utility was *not* used when submitting the jobs. Therefore, the nth simulation was assigned to a core automatically by the operating system, and the thread context was free to migrate. The performance of the Ivy Bridge node and the Haswell4 were nearly identical between the pinned and unpinned cases. However, there were some marked differences in the comparison between the pinned and

unpinned processing for Haswell36. For the pinned case (Figure 11, right), a sharp drop in performance occurred after each of the 18 cores on the two Haswell packages had a FAST executable assigned to it. This implies that one core was burdened with context switching between two pinned tasks, while the remaining 35 cores were free to execute the test workload without interruption (barring any operating system preemption). Because tasks were pinned, these 35 cores should have completed their assigned computation and sat idle while the overburdened core completed two tasks. Because timing was quantified as aggregate time-to-completion, and this time was now dictated by this overburdened core, a sudden drop in performance appears. As load was balanced and throughput increased by additional hyperthreads and concurrent tasks, the apparent performance went up because the fraction of total core idle time decreased in moving from 37 threads to 72 threads. However, a notable difference in throughput with respect to the use of hyperthreads occurred when processes were not pinned. Here, throughput increased smoothly through the range of logical core counts tested. We hypothesize that this difference from pinned behavior arose due to automatic load balancing by the operating system, decreasing the amount of idle time spent by the Haswell cores as the hyperthreads became occupied with work.

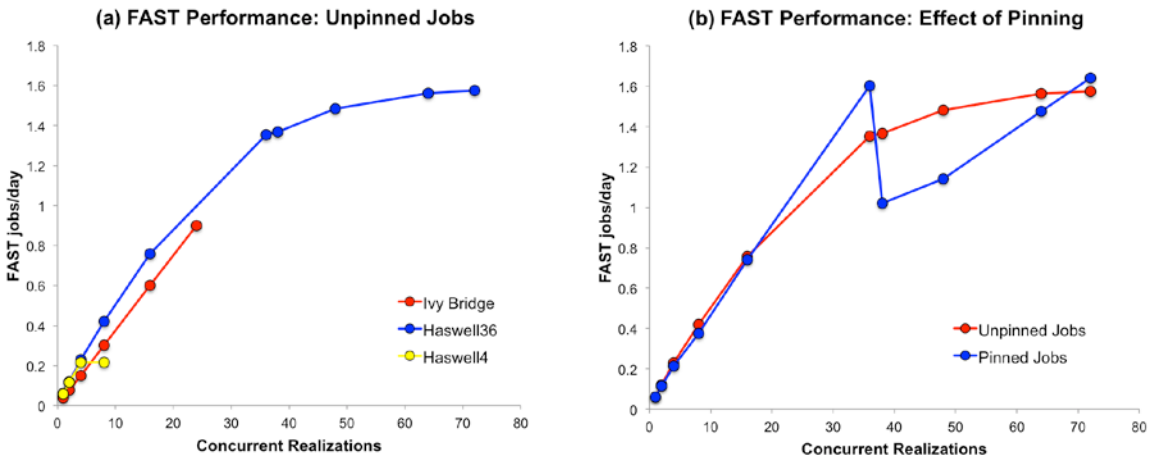

Figure 11. FAST performance variation with thread count when unpinned (left) and pinned (right) on the Haswell36 system

Because the FAST code is typically run in a high-throughput workflow and serial work quanta complete in an acceptable time, the need for parallelization for a single FAST run is minimal. From the profiles shown in Figure 11, one is presented with a dilemma in achieving peak performance: either pin tasks and leave hyperthreads idle, or let the operating system assign tasks to cores dynamically and use all available hyperthreads. For the average user who is not primarily tasked with understanding delicate aspects of performance computing, we anticipate that a clearer recommendation may be made with the latter scenario. Enabling hyperthreading at an administrative level is thus advisable, allowing the throughput-focused user to tune their throughput monotonically with added hyperthreads while permitting the performance-focused user the freedom to manipulate their runtime configuration as desired. It is worth noting that the performance of Haswell4 levels off at the maximum number of physical cores (with hyperthreading on) in both the pinned and unpinned cases. This could be due to differences in how resources are allocated for the dual-package architecture on the Haswell36 system as opposed to the single-package architecture for the Haswell4 system.

4 Discussion

For all applications tested, the Haswell microarchitecture showed increased performance per core and overall job throughput when compared to the Ivy Bridge architecture. The architectural improvements in Haswell—including a deeper scheduler and reorder buffers, more general FP execution units, an FMA operation in hardware, and enhanced memory store bandwidth—were expected to have a positive impact on the FP-focused and data-heavy scientific workloads explored. From the perspective of optimizing throughput on a community system, additional factors of interest in purchasing, configuration, and operation are the impacts of core density, hyperthreading, and job packing on a compute node.

For the STREAM microbenchmark, the primary performance indicator observed was the pinning of threads to cores. Ivy Bridge performed better when the threads were pinned to cores, even when the physical cores were oversubscribed. In the case of Haswell36, pinning the threads did not alter the performance. A small dip in performance was seen on the Haswell36 node when hyperthreads were employed, losing approximately 5% of the capability seen in the range of 16–36 threads. The matrix multiplication benchmark served to illustrate two points. First, a substantial performance advantage was seen using MKL routines to optimize the MM operations compared to coding the multiplication as a set of nested loops as would be intuitive to a domain scientist. In the case of Ivy Bridge, an approximate 20%–25% boost in throughput was obtained when employing MKL routines instead of the naïve implementation. Notably, the relative advantage was substantially greater on Haswell36, potentially illustrating the greater effects of data movement as core counts grow. A second point is that MKL routines did not increase performance once each Haswell36 physical core hosted multiple threads, whereas the naïve algorithm benefited slightly from hyperthreads. This is not an entirely surprising pattern: hyperthreads are expected to be effective in mitigating the performance impacts of unoptimized algorithms or workflows, but they should offer little to highly optimized codes.

The VASP benchmark offers three main lessons. First, both the Ivy Bridge and Haswell systems show significant leveling-off of performance gains once a node is more than half filled, even with processes mapped in a "scattered" fashion on the cores. Although this is not entirely due to resource contention (i.e., the scaling of a distributed-memory parallel job with rank count also contributes), this observation nonetheless points to decreasing returns on investment for nodes with high core counts that are intended to serve common workloads, many of which do not possess strong performance scaling with core count. As a corollary, particularly if attention is not paid to physical process mapping, the bulk of potential performance is reached with only a fraction of the available cores on a node (Figure 6). Second, to the extent that the Haswell4 is representative (Figure 7), hyperthreading offers little detriment or benefit to throughput of multiple-process jobs. The choreography of lightweight threads might offer improvements via latency hiding; however, packing hyperthreads with heavier processes (e.g., MPI ranks or independent jobs) slightly decreases throughput—whatever computational latency may be hidden by context switching is more than made up for in the demands of that switching itself. Finally, maximizing hardware value as core counts per node increase will likely involve robust job scheduling at core-level granularity, permitting multiple jobs, and even multiple users, to run concurrently on a single node. Although resource contention in this case may increase time-to-solution somewhat, the greater resource availability (i.e., up to the full number of cores per node in the case of a purely serial workload) would be expected to more than compensate.

The Gaussian benchmark shows a small (approximately 20%) difference between the Ivy Bridge and Haswell processors, indicating that the tested version of Gaussian with the representative job does not benefit extensively from the Haswell microarchitecture. In addition, the typical high-throughput job at NREL is not memory-limited. Therefore, running multiple jobs on the same node could improve the overall job completion rate and effective utilization of the cluster. For the Haswell36, the throughput increase was significant compared to running one 36-thread job on the node. This pattern will of course

depend on job size and shared-memory parallel scalability of the algorithm employed. We nonetheless feel comfortable recommending preliminary benchmark studies for high-throughput Gaussian calculations run on future architectures to find an optimum combination of thread count per job and job count per node. This suggests that Haswell nodes with lower core counts per node may be more suitable for high-throughput computing, to the degree that this application and test job reflect a typical quantum of work. Finally, hyperthreads only improved Gaussian throughput when they were limited by compute resource (versus memory or input/output bandwidth, for example). With large core counts, the bottleneck appears to change from FP speed to data movement, which blunts the improvements hyperthreading can bring.

The molecular dynamics packages LAMMPS and Amber show similar results, with the tested Haswell processors showing a 60%–70% improvement compared to Ivy Bridge when running jobs of the same configuration for both packages.

Relative to the Ivy Bridge system, the FAST benchmark demonstrates a 23% performance advantage for Haswell at equivalent core counts, in good agreement with the improvement seen for the Gaussian benchmark. This performance increase is realized when the number of threads used is limited to the physical core count. If the mapping between running executable and threadID is a sequential, linear layout, then the best throughput for large numbers of serial FAST jobs does not require the larger thread count available via hyperthreading. Even when the mapping between the executing job and threadID is managed by the runtime environment, the maximum performance achieved with hyperthreads is either less than or equal to that without hyperthreads. It is important to note that the difference in peak throughput achieved between the Ivy Bridge and Haswell36 test systems is substantially larger at each system's respective maximum core counts. This highlights a scenario in which a much greater (i.e., 50%) core number on a single node can be a "win," namely when there is little resource contention among running processes and when inter-processor communication is not dense.

5 Conclusions

Two generations of node configurations hosting the Haswell microarchitecture were tested with a suite of microbenchmarks and application examples and compared to a current Ivy Bridge production node on NREL's Peregrine HPC cluster. These two configurations represent a "low-energy" design with a low core count, and a high-performance design with 36 cores per node that represents a 50% increase compared to the Ivy Bridge test system. A primary conclusion from this study is that the additional cores are of little value to individual task performance—limitations to application parallelism, or resource contention among concurrently running but independent tasks, limits the effective utilization of these added cores. Further, active use of hyperthreading had an almost uniformly negative impact on the tested applications (the exception in MM; see Figure 3), as might have been anticipated based on historical behavior with scientific applications. It offered no additional performance in applications beyond the assignment of a single process or thread per physical core. Thus, some energy and monetary savings might be realized by the acquisition of chips without this capability. However, given the capability, enabling simultaneous multithreading should not in itself lead to performance issues. Finally, although pinning threads or processes to hardware cores offered some benefit (for example, with STREAM), it was not so much under these test cases that productivity would be affected substantially by not pinning.

These observations offer some guidance to the procurement of future HPC systems at NREL. First, raw core count must be balanced with available resources, particularly memory bandwidth. The commodity enterprise computing ecosystem appears to be at a point in its evolution where balance-of-system is a primary determinant of value. This consideration implies that a substantial productivity enhancement is possible given a fixed budget by acquiring greater numbers of more balanced nodes, rather than by focusing on processor capability. Second, while enabling hyperthreading at the hardware level did not in itself lead to lower performance, the additional threads continue to be largely irrelevant to the workloads that are commonly seen at NREL, and that were tested here. HPC codes optimized for performance tend not to benefit from the latency hiding afforded by hyperthreads, and for most real-world technical computing conditions they are at best neutral and potentially detrimental overall. A possible exception to this conclusion arises from multiple uncoordinated processes run on a single compute node, wherein exploitation of hyperthreads can achieve maximum throughput without imposing the hardship of thread pinning and other performance-focused concerns on the user. Finally, perhaps the most impactful enhancement to productivity might occur through enabling multiple concurrent jobs per node. Except in rare cases of superlinear speedups, overall peak throughput with N processing elements should occur when N independent processes are mapped to them. This follows from parallel processing requiring some finite degree of coordination among ranks or threads, while consuming cores equal to the number of ranks or threads. So although the same amount of productive work might be done in a serial or a parallel job, the latter adds additional requirements for coordination. The results of this are shown in Figure 5, where maximum throughput is achieved with serial jobs. Maximizing the value of NREL resources thus entails both a technical achievement (transparent assignment of work to resources with minimal user input to or knowledge of the underlying computing architecture) and an education component (given a large enough workload, one can achieve more by doing many slow things at once than by doing fast things in consecutive order).

References

1. 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).
2. R. Myslewski, "Deep, Deep Dive Inside Intel's Next-Generation Processor," *The Register* (2012), accessed August 26, 2015. http://www.theregister.co.uk/2012/09/20/intel_haswell_microarchitecture_deep_dive.
3. D. Kanter, "Intel's Haswell CPU Microarchitecture," *Real World Technologies* (2012), accessed August 26, 2015. http://www.realworldtech.com/haswell-cpu/.
4. F. Masci, *Benchmarking the Intel® Xeon Phi™ Coprocessor* (2013), accessed April 13, 2015. http://web.ipac.caltech.edu/staff/fmasci/home/miscscience/MIC_benchmarking_2013.pdf .
5. G. Kresse, and J. Furthmüller, "Efficiency of ab-Initio Total Energy Calculations for Metals and Semiconductors using a Plane-Wave Basis Set," *Computational Materials Science* 6(1996): 15–50.
6. G. Kresse and J. Furthmüller, "Efficient Iterative Schemes for *ab Initio* Total-Energy Calculations Using a Plane-Wave Basis Set," *Physical Review B* 54(1996): 1,1169–86.
7. G. Kresse and J. Hafner, "*Ab Initio* Molecular Dynamics for Liquid Metals," *Physical Review B* 47(1993): 558–61.
8. G. Kresse, *Ab Initio Molekular Dynamik für Flüssige Metalle* (Technische Universität Wien: 1993).
9. C.L. Lawson, et al., "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Transactions on Mathematical Software* 5(1979): 308–23.
10. L.S. Blackford, et al., "An Updated Set of Basic Linear Algebra Subprograms (BLAS)," *ACM Transactions on Mathematical Software* 28(2002): 135–151.
11. E. Anderson, et al., "LAPACK: A Portable Linear Algenbra Library for High-Performance Computing," in *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing* (1990).
12. J.W. Cooley and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation* 19(1965): 297–301.
13. M.J. Frisch, et al., *Gaussian 09, Rev. D.1*. (Wallingford, CT: Gaussian, Inc., 2009).
14. S. Ahuja, N. Carriero, and D. Gelernter, "Linda and Friends," *Computer* 19(1986): 26–34.
15. S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics* 117(1995): 1–19.
16. D.A. Case, et al., *AMBER 2015* (San Francisco: University of California, 2015).
17. *NWTC Information Portal (FAST)*, March 19, 2015, accessed July 22, 2015. https://nwtc.nrel.gov/FAST.
18. D.J. Laino, and A.C. Hansen, *User's Guide to the Wind Turbine Dynamics Aerodynamics Computer Software AeroDyn* (Subcontract Report) (Golden, CO: National Renewable Energy Laboratory, December 2002).
19. P.J. Moriarty, and A.C. Hansen, *AeroDyn Theory Manual* (Technical Report), NREL/EL-500-36881 (Golden, CO: National Renewable Energy Laboratory, December 2005).

20. J.M. Jonkman, "Dynamics Modeling and Loads Analysis of an Offshore Floating Wind Turbine" (Ph.D. diss., University of Colorado, Department of Aerospace Engineering Sciences, 2007).
21. J.M. Jonkman, *Dynamics Modeling and Loads Analysis of an Offshore Floating Wind Turbine* (Technical Report), NREL/TP-500-41958 (Golden, CO: National Renewable Energy Laboratory, November 2007).
22. J.M. Jonkman, "Dynamics of Offshore Floating Wind Turbines—Model Development and Verification," *Wind Energy* 12(2009): 459–92.
23. J.M. Jonkman, and M.L. Buhl, , Jr., *FAST User's Guide* (Technical Report), NREL/EL-500-38230 (Golden, CO: National Renewable Energy Laboratory, 2005).