TACC Technical Report TR-17-01

Benchmarking the Intel®Xeon®Platinum 8160 Processor

Document Revision 1.0
August 10, 2017

Antonio Gómez-Iglesias, Feng Chen, Lei Huang, Hang Liu, Si Liu, Carlos Rosales
{agomez, chenk, huang, hliu, sliu, carlos}@tacc.utexas.edu

Texas Advanced Computing Center
The University of Texas at Austin

www.tacc.utexas.edu

Contents

List of Figures

1 INTRODUCTION

This report presents a set of results for different microbenchmarks and applications on the Intel Xeon Platinum 8160 Processor, formerly known as Skylake. For simplicity, we will use both Skylake and SKX to refer to this processor. We use the Skylake nodes that will be available in Stampede2. This system will provide Intel Knights Landing and Skylake chips interconnected by a 100 Gb/sec Intel Omni-Path (OPA) network with a fat tree topology. The peak performance of the system will be 18 PF.

1.1 Hardware Characteristics

Table 1 shows the main characteristics of the processor.

Component	Value
Number of Cores	24
Clock Speed	2.10 GHz
Turbo Speed	3.70 GHz
L3 Cache	Shared, 33 MB, 11-way set associative
L2 Cache	Unified 24 MB (1 MB/core), 16-way set associative
L1 Cache	Private, 32 KB (per core), 8-way set associative
Memory Type	DDR4-2666
Memory Channels	6
PCI Express	Gen 3, x48 lanes
Number of UPI Link	3
UPI Link Speed	10.4 GT/s
Lithography	14 nm
TDP	150 W
Instruction Set Extensions	SSE4.2, AVX, AVX2, AVX-512
AVX-512 FMA Units	2

Table 1: Intel Xeon Platinum 8160 Characteristics

Effective CPU frequency depends on the instructions that are executed and the number of active cores on the chip. The most effective instruction set for a given code will depend on implementation details. Fig. 1 shows the CPU turbo frequency depending on the instruction set and the number of active cores on the chip.

In the system used, each compute node consists of two sockets of Xeon Platinum 8160 and 192 GB of main memory. As shown in Figure 2 the local disk (sda), the Omni-Path link, and the management GE network interfaces are connected to socket 0.

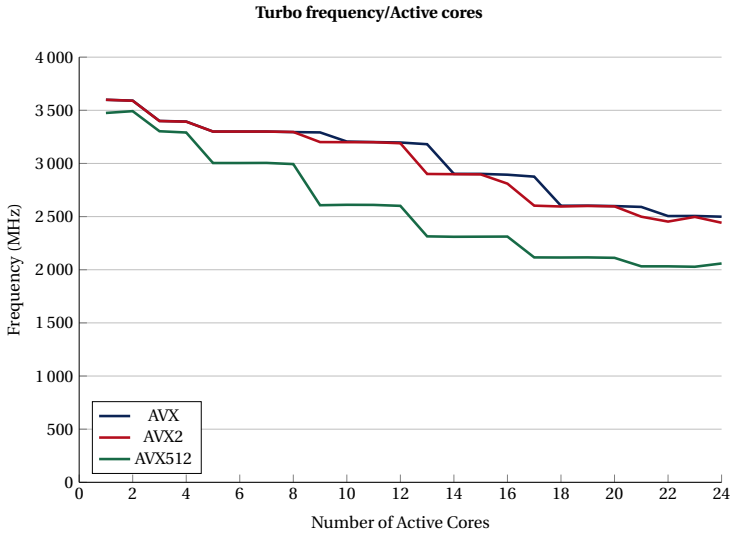

Figure 1: Dual socket SKX. Turbo frequency.

1.2 Software Stack

At the time our running our tests, the software stack was as shown in Table 2.

Element	Version
Kernel	3.10.0-514.26.2.el7.x86_64
Compiler	Intel version 17.0.4
BLAS	Intel MKL 2017 update 3
MPI	Intel MPI version 17.0.3
Omni-Path	10.4.1.0

Table 2: Software Stack as of August 10^{th}, 2017

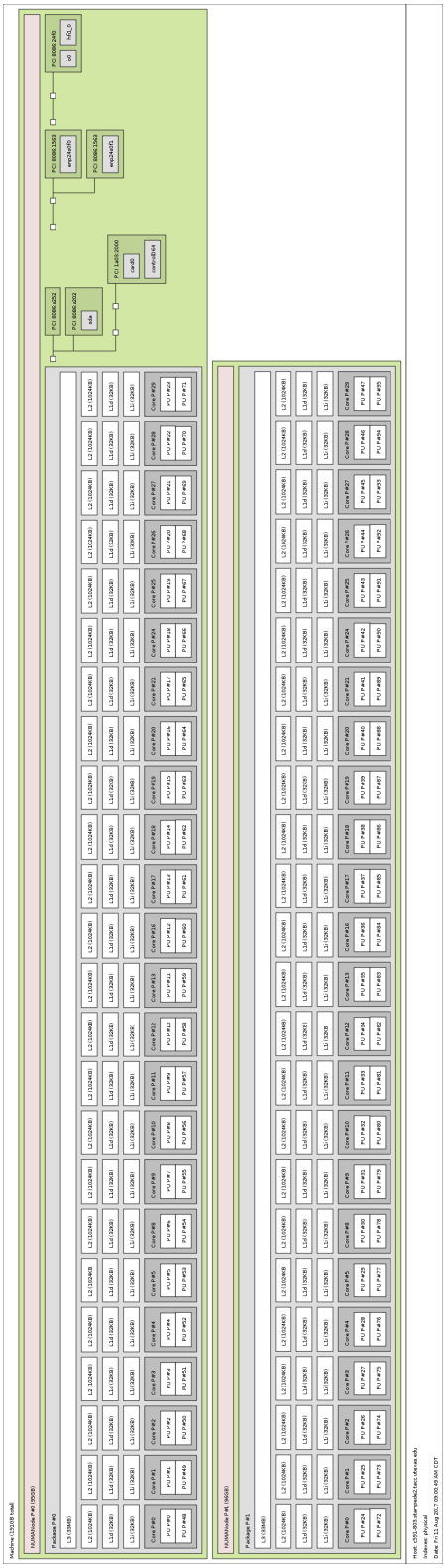

Figure 2: Hardware description for compute node from HWLOC

2 MEMORY BANDWIDTH

Memory bandwidth is critical for many scientific codes. While floating point processing power has increased tremendously over the years, memory bandwidth has grown at a much slower rate. As a consequence of this disparity in growth, modern systems tend to be fairly imbalanced when comparing the number of words that can be retrieved from memory per cycle with the number of floating point operations that can be executed per cycle. While certain codes make excellent use of the cache hierarchy (reusing cached data, ensuring contiguous memory access, etc, ...) other can't because of algorithmic limitations or because they have been written in a manner that does not use the memory subsystem optimally. Thus, having a measure of the sustained memory bandwidth of a system is an important step in the process to understand the performance one may expect for a broad range of applications.

The system memory bandwidth was measured using the STREAM benchmark revision 5.10 [9]. This benchmark is the industry standard for measuring sustained memory bandwidth. It consists of four tests, described in Table 3. For our results, we consider the output of the `Triad` test.

Function	Operation
Copy	c[i] = a[i]
Scale	b[i] = scalar*c[i]
Add	c[i] = a[i]+b[i]
Triad	a[i] = b[i]+scalar*c[i]

Table 3: STREAM benchmark operation definitions

We used the OpenMP implementation of STREAM. We increased the number of threads from 1 to 96 to explore the memory bandwidth on a single socket, on two sockets, and with hyperthreading. The results can be seen in Fig. 3. Once the number of threads reached 33, the bandwidth was saturated and did not improve. The peak in this case was 194 GB/s. Once the number of threads exceeded the number of physical cores available, the achieved bandwidth decreased.

Fig. 4 shows the memory bandwidth when using only the local or remote memory to each socket.

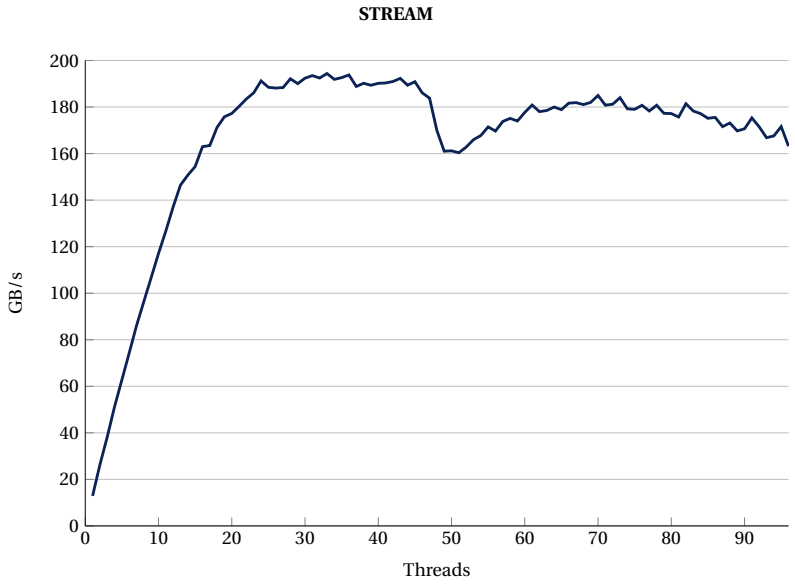

Figure 3: Dual socket SKX node. STREAM Triad

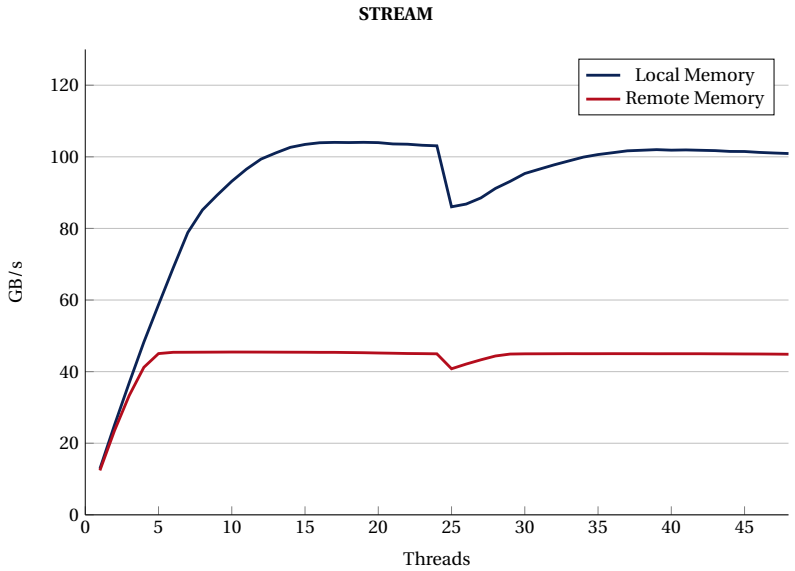

Figure 4: Dual socket SKX node. STREAM Triad. Local Memory.

3 FLOATING POINT PERFORMANCE

Floating point performance is the traditional measure of machine performance for High Performance Computing systems, and represents the overall system capability for performing mathematical operations. Typical benchmarks for overall floating point capability are the Linpack benchmark (HPL, used for the TOP500 list, see references [5] and [1]) and matrix matrix multiplication tests. These algebraic tests have highly optimized implementation with high data reuse and fairly limited memory bandwidth requirements, which are capable of achieving floating point performances very close to the theoretical peak performance of the hardware. Floating point performance is critical for scientific workloads, since they very often rely on solving an algebraic system of equations. Even if a scientific workload is not directly solving a system of linear equations it is likely that it will make extensive use of the floating point capability of the system during tasks like post-processing and data analysis. This makes the determination of the floating point capability of a compute system another essential element in an overall understanding of the system performance.

The tests performed in this section are limited to single node executions, since the purpose is to evaluate the processor floating point performance in isolation form network communication and other factors present in distributed execution. The DGEMM benchmark developed by NERSC[1] was used to measure the multithreaded performance of the BLAS call DGEMM. The code was compiled to directly call Intel MKL at runtime.

Fig. 5 shows the TFLOPS achieved by a dual socket SKX node for different matrix sizes. The command used for these results was `MKL_ENABLE_INSTRUCTIONS=AVX512 OMP_NUM_THREADS=48 ./mt-dgemm $size`, where $*size* is the x-axis value in the figure.

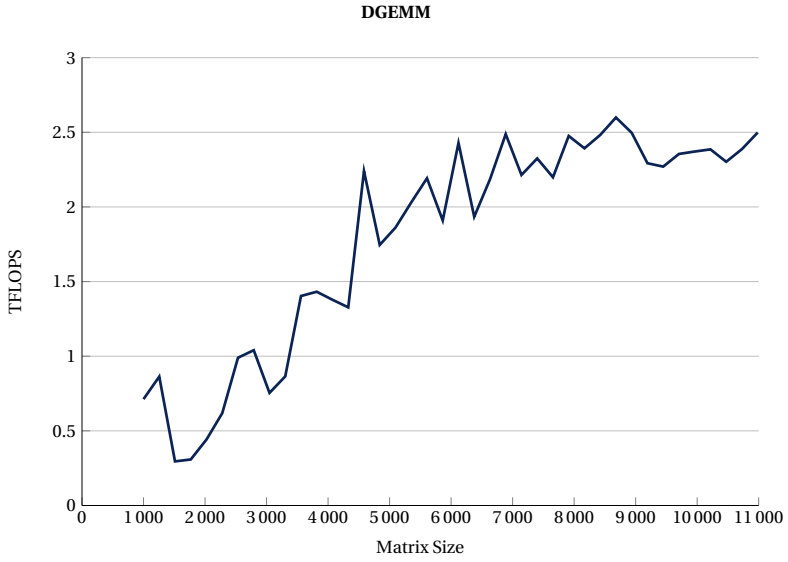

Figure 5: Dual socket SKX node. TFLOPS for different matrix sizes.

[1] http://www.nersc.gov/research-and-development/apex/apex-benchmarks/dgemm/

Fig. 6 shows the TFLOPS achieved by a dual socket SKX node when using a fixed matrix size of 10000 and increasing the number of threads from 1 to 48 (total number of cores in the the node). We did not use hyperthreading since it did not help in our preliminary tests.

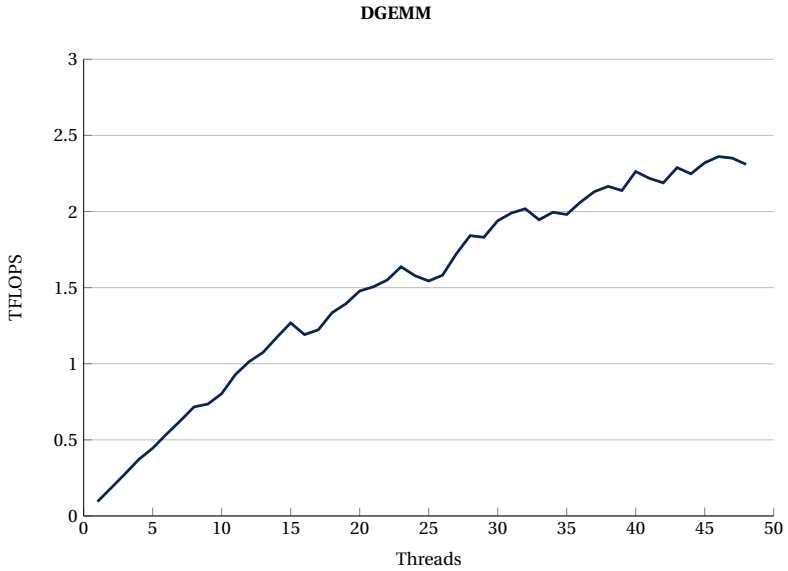

Figure 6: Dual socket SKX node. TFLOPS for different threads.

4 OpenMP

OpenMP is one of the most commonly used shared memory parallel programming models. The continuous trend of increasing core count per socket to improve system performance makes threading performance critical for many codes, both shared memory and hybrid. In this section we investigate the synchronization overhead of a wide variety of OpenMP (OMP) language constructs.

OpenMP overhead was measured using the EPCC OpenMP benchmark v3.1. For implementation details see [4]. The benchmark was executed using two different affinity settings: scatter, using `OMP_PROC_BIND=spread` and compact, using `OMP_PROC_BIND=close`.

Fig. 7 and 8 show the overhead of the most common synchronization constructs in OpenMP when using scatter and compact affinity.

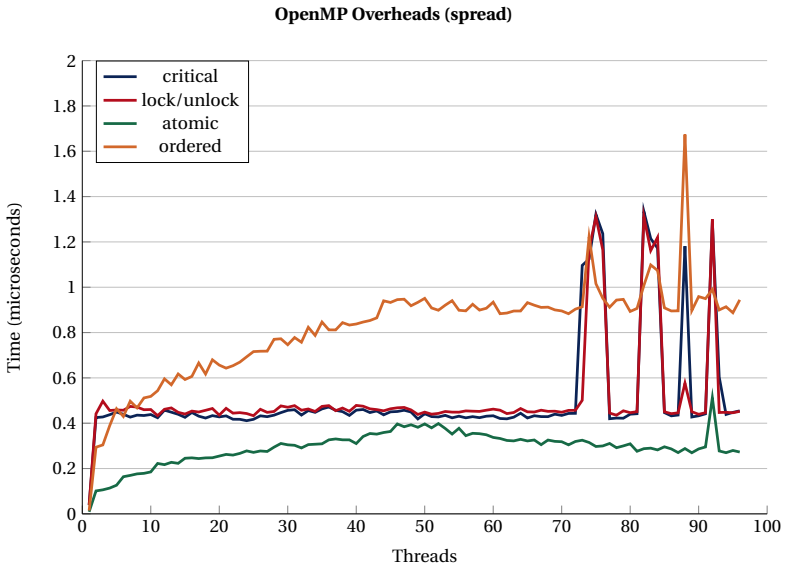

Figure 7: Dual socket SKX node. OpenMP synchronization overhead (spread).

Fig. 9 and 10 depict the overheads of the most typical OpenMP constructs for the two affinity settings studied.

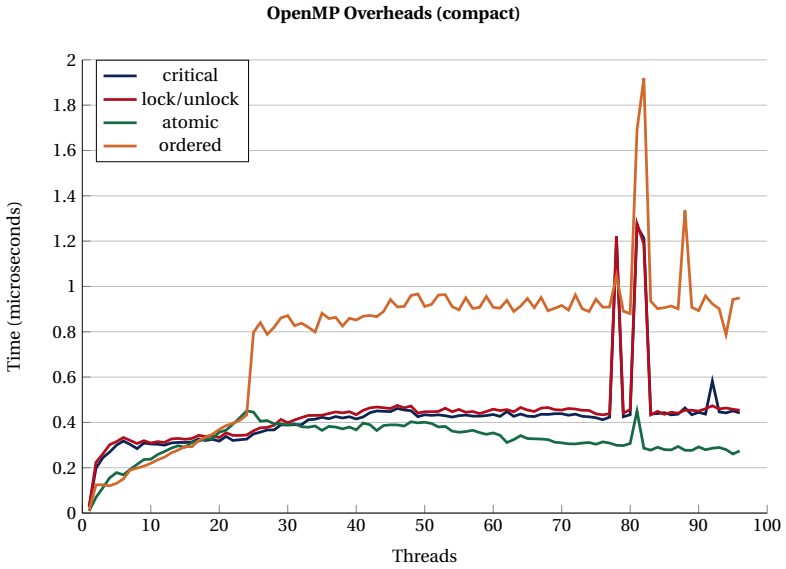

Figure 8: Dual socket SKX node. OpenMP synchronization overhead (compact).

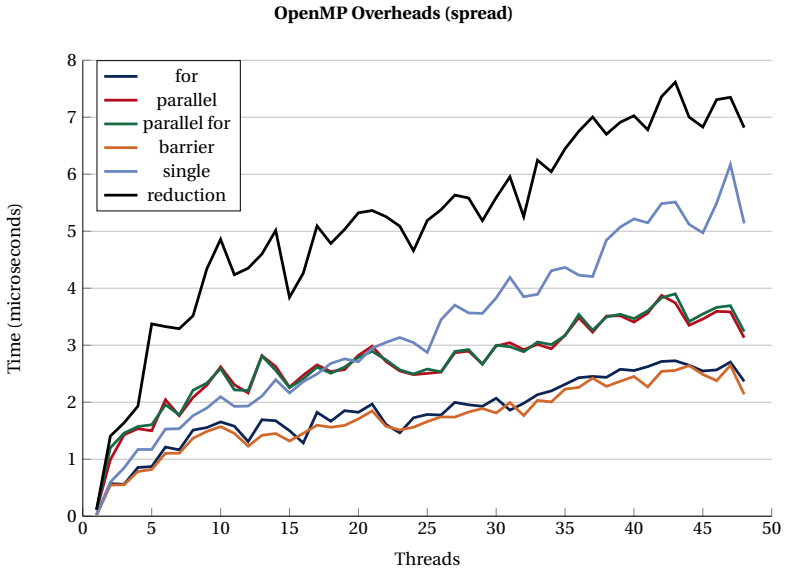

Figure 9: Dual socket SKX node. OpenMP constructs overhead (spread).

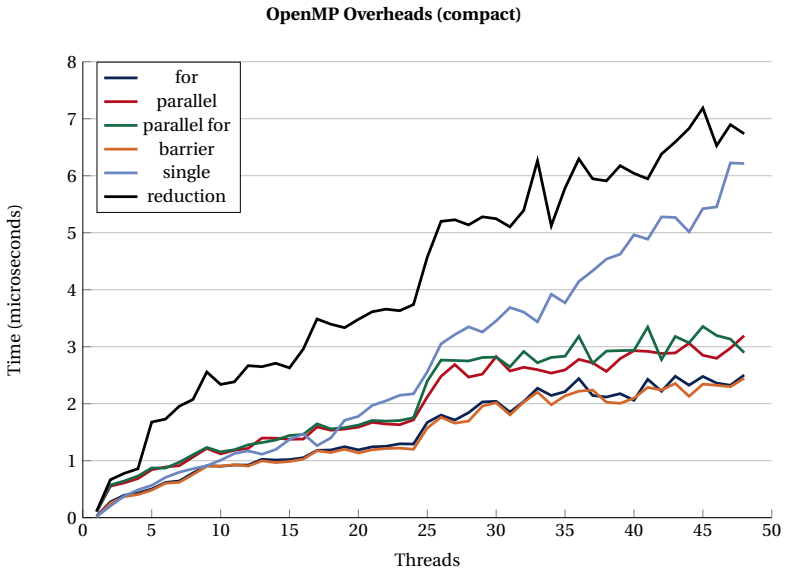

Figure 10: Dual socket SKX node. OpenMP constructs overhead (compact).

5 MPI

5.1 Internode Latency

MPI latency is of importance to codes that exchange small messages, where latency plays a more significant role than bandwidth. Codes with irregular communication patters typically benefit from short latencies. When looking a latency values off node the limiting factors will be the network interconnect and adapter, while on node the limiting factor will be the memory subsystem, including the socket to socket communication pathways.

Fig. 11 shows the latency between 2 nodes on the same rack. On both nodes, the MPI task is pinned to core 0. The latency in this case is 1.06 usec. For the case of socket 1 to socket 1 (pinning to core 24 on each node), the latency is 1.19 usec.

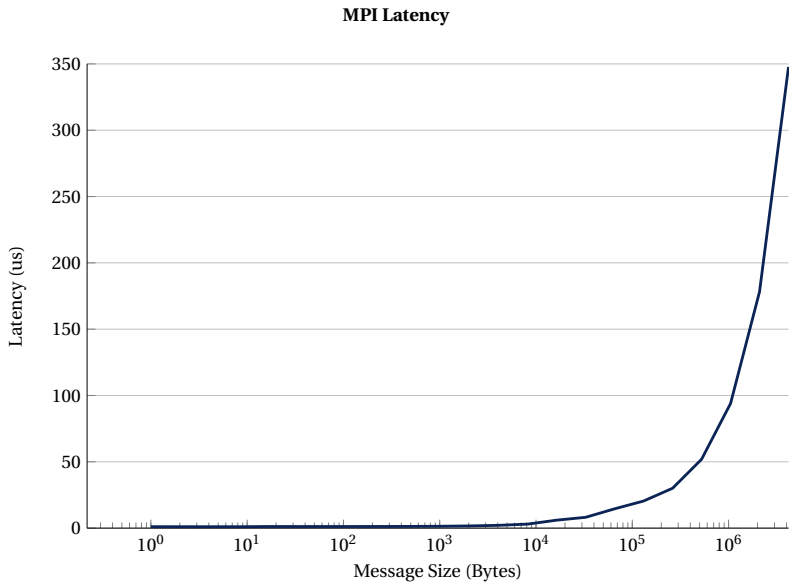

Figure 11: Effective inter-node latency for a single pair of MPI tasks

5.2 Intranode Bandwidth

On node exchanges do not go through the Omni-Path network and thus are not limited by its characteristics. The effective bandwidth achieved within a node is limited, instead, by the memory subsystem properties for the node and the memory contention when multiple tasks exchange data simultaneously.

Fig. 12 shows the effective bandwidth for a single pair of MPI tasks. Three cases are presented. In the first case the two tasks are bound to the first socket, to cores 0 and 1. In the second case the tasks are bound one to each socket and, in particular, to cores 0 and 24. In the last case the tasks

are bound to the second socket, to cores 24 and 25. The achieved bandwidth is very similar in all three cases. For these experiments, to allocate memory on the closest NUMA node to the core where the MPI task is running.

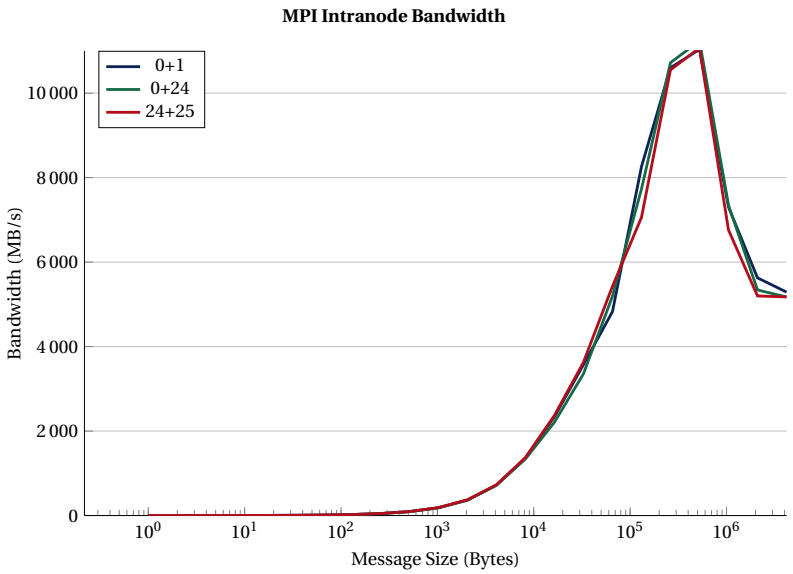

Figure 12: Effective intra-node bandwidth for a single pair of MPI tasks

5.3 Internode Bandwidth

Many scientific codes require some type of distributed memory model because of their scale and complexity. The Message Passing Interface (MPI) framework allows for communication in distributed systems, and is the de facto standard for codes working in distributed memory systems. Codes that have large messages to exchange depend heavily on the MPI bandwidth supported by the system for their performance, especially if they execute mainly blocking calls or collective operations. Understanding the achievable MPI bandwidth in a system is critical for estimating scalability of distributed codes.

Figure 13 shows the inter-node bandwidth achievable for different message size. For these tests one MPI task was bound to the first socket on each node.

Fig. 14 shows the internode bandwidth achieved running OSU benchmark 5.3.2. For these results, one MPI task is pinned to a specific core on one node and the other process is pinned to another core on the other node. A double for loop allows to explore all the combinations of sender and receiver for all the cores in both nodes.

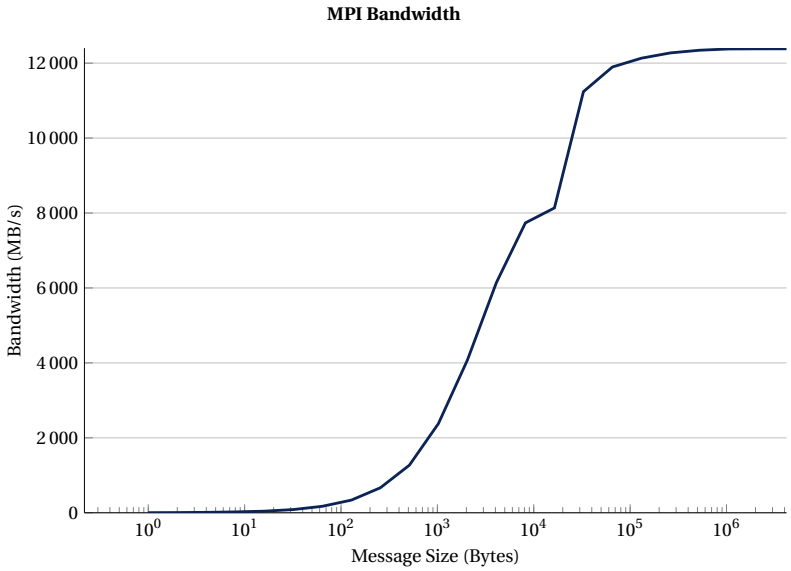

Figure 13: Effective inter-node bandwidth for a single pair of MPI tasks

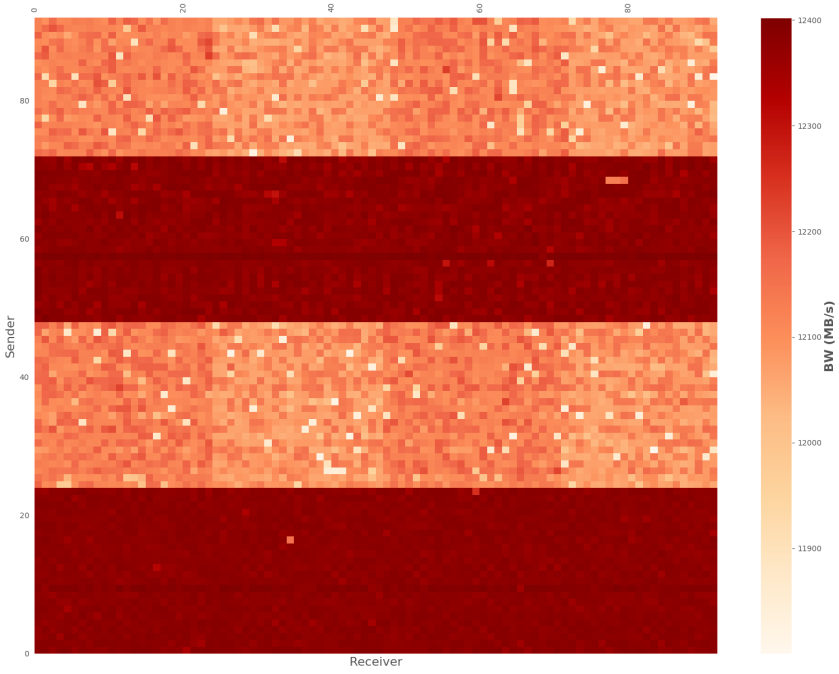

Figure 14: Internode bandwidth

6 APPLICATIONS

We chose a set of representative HPC applications. Several of these codes are among the ten most used applications in Stampede, while other are good representations of common workloads. For these results, SKX is a dual socket node where each socket presents the characteristics described on Table 1. SNB is a dual socket Sandy Bridge node as shown in Table 4. HSW is dual socket Intel Haswell node that follows the specifications in Table 5. Finally, KNL is an Intel Knights Landing chip as shown in Table 6.

Component	Value
CPU Sockets	2 × Intel Xeon E5-2680 (Sandy Bridge EP)
Memory	32 GB DDR3-1600 RAM (8x4 GB, 4 channel on-chip)
Total Number of Cores	16
Clock Speed	2.7 GHz
Turbo Speed	3.5 GHz
L3 Cache	Shared, 20 MB
L2 Cache	Unified 256 KB
L1 Cache	Private, 32 KB (per core)
Memory Type	DDR3
Memory Channels	4
PCI Express	Gen 3, x40 lanes
Number of QPI Link	2
QPI Link Speed	8 GT/s
Lithography	32 nm
TDP (socket)	130 W
Instruction Set Extensions	SSE4.2, AVX

Table 4: Intel Xeon Sandy Bridge Node Characteristics

Component	Value
CPU Sockets	2 × Intel Xeon E5-2690 v3 (Haswell EP)
Memory	64 GB DDR4-2133 RAM (8 x 8GB dual rank x8 DIMMS)
Total Number of Cores	24
Clock Speed	2.6 GHz
Turbo Speed	3.5 GHz
L3 Cache	Shared, 30 MB
L2 Cache	Unified 256 KB
L1 Cache	Private, 32 KB (per core)
Memory Type	DDR4
Memory Channels	4
PCI Express	Gen 3, x40 lanes
Number of QPI Link	2
UPI Link Speed	9.6 GT/s
Lithography	22 nm
TDP (socket)	135 W
Instruction Set Extensions	SSE4.2, AVX, AVX2

Table 5: Intel Xeon Haswell Node Characteristics

Component	Value
CPU Sockets	1
Memory	16 GB MCDRAM, 92 GB DDR4-2400
Total Number of Cores	68
Clock Speed	1.4 GHz
Turbo Speed	1.6 GHz
L2 Cache	Unified 1 MB
L1 Cache	Private, 32 KB (per core) Data and Instruction caches
Memory Channels	6
PCI Express	Gen 3, x36 lanes
Lithography	14 nm
TDP	215 W
Instruction Set Extensions	SSE4.2, AVX, AVX2, AVX-512

Table 6: Intel Knights Landing 7250 Node Characteristics

6.1 WRF

The **Weather Research and Forecasting** (WRF) Model [6] is a widely used numerical weather prediction system used for both research and operational forecasts. WRF is primarily a Fortran code implemented using MPI and OpenMP for distributed computing. The problem space on each process is divided into tiles that are processed by OpenMP threads. Ideally, the best performance is achieved when the size of the tile fits into the smallest cache. Having multiple application tiles allows WRF to obtain high levels of memory bandwidth utilization. For this investigation we use the Continental US (CONUS) 12km input benchmark.

Fig. 15 shows the performance relative to Sandy Bridge for all the chips previously introduced. The results used for this figure are achieved by the best configuration for each chip in terms of processes and threads.

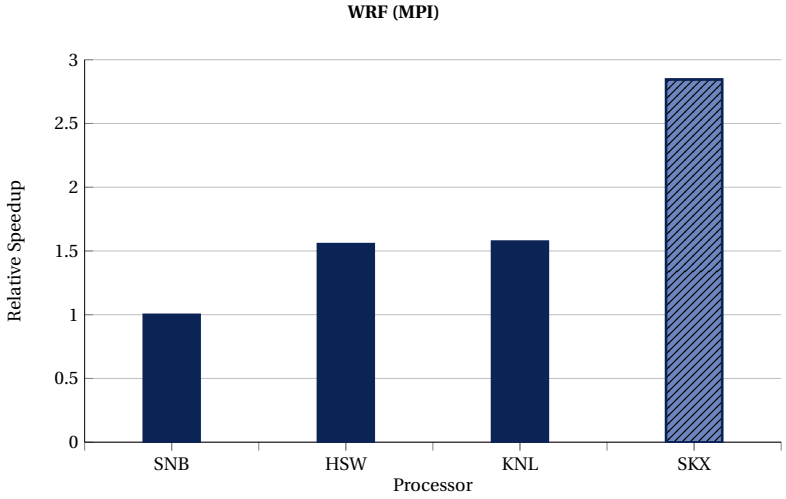

Figure 15: WRF: Speedup Relative to SNB

We also studied the impact of the different ISA (Instruction Set Architecture) in the performance of the code. The results can be seen in Fig. 16. In the figure, higher is better. It can be seen how the best performance is achieved by using the `-no-vec -xCORE-AVX512` compilation flags, with the case of `-no-vec -xSSE4.2` being very close.

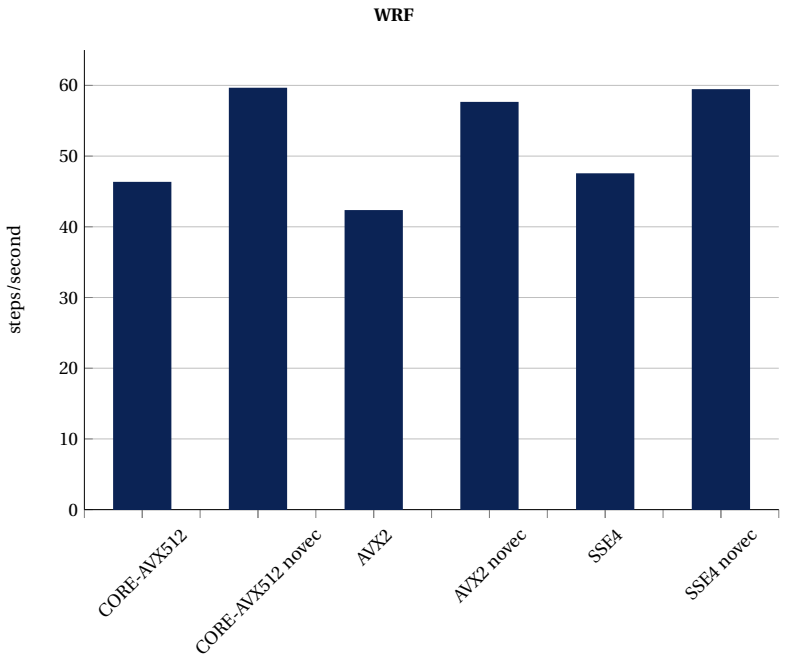

Figure 16: WRF: ISA impact

6.2 FLASH

FLASH [11] is a multi-physics framework often used for the simulation of turbulent fluid flow and transport in plasmas, including exploding stars, laser energy deposition, and nuclear burning. The benchmark case computes a driven turbulence simulation on a fixed, 64^3-point mesh per node. Version 4.2.2 of the code was used in the tests with minor modifications to improve threading and vectorization added to a single file in the `Stir` subdirectory.

As for the WRF case, we compared the performance of the different chips. The results are depicted in Fig. 17. The difference in performance is not as large as in the WRF case, even though SKX offers a much better performance than all the other chips.

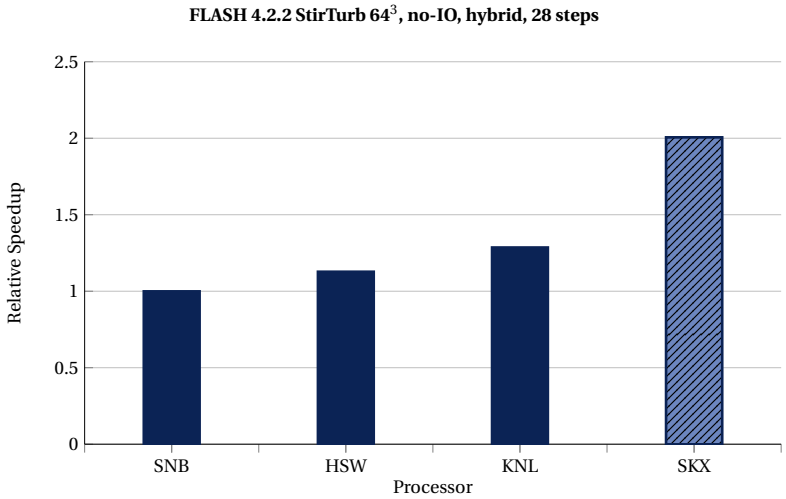

Figure 17: FLASH: Speedup Relative to SNB

Fig. 18 shows the different compilation flags used and how they affect the performance of the code. The best result was achieved when using `-O3 -no-vec -xCORE-AVX512`. It is worth noticing the difference between using or not using `-no-vec` in this case and, also, how when not `-no-vec` is specified, `-xCORE-AVX2` clearly outperforms `-xCORE-AVX512`.

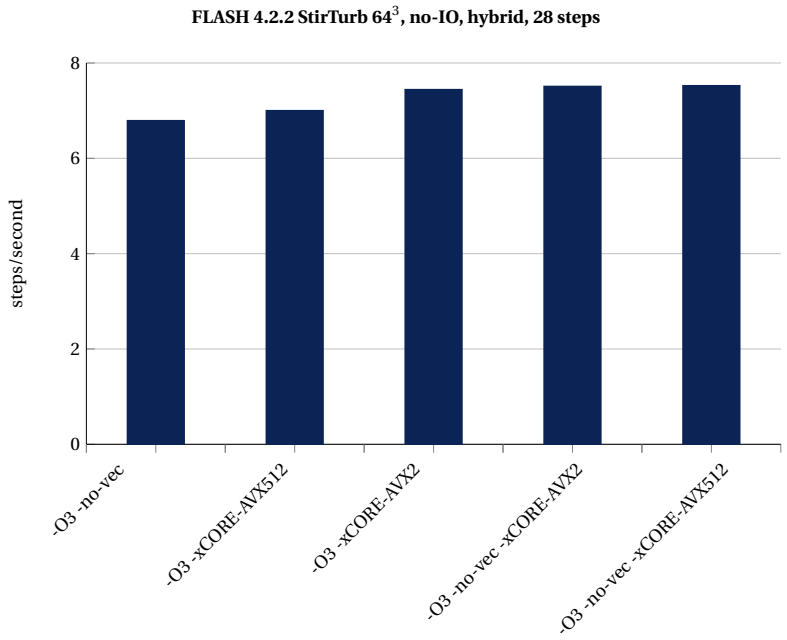

Figure 18: FLASH: ISA impact

6.3 NAMD

NAMD [10] is a parallel molecular dynamics code for high-performance simulation of large molecular systems used by many research teams around the world and the 2002 Gordon Bell Prize winner. NAMD is also the most used application on Stampede. This study uses the APOA1 and STMV20 NAMD benchmarks with modified input for the single and multi node studies respectively. The APOA1 benchmark is a 92-thousand atom molecular dynamics simulation of Apolipoprotein A-1 which is the primary component of the high-density lipoprotein cholesterol molecule. The STMV20 benchmarks is a 20 million atom simulation of the satellite tobacco mosaic virus, which is a small icosahedral plant virus which worsens the symptoms of infection by tobacco mosaic virus (TMV). A multicore version of NAMD CVS 2016-11-01 was built for the single node test, and a hybrid memory-optimized version was built for the large STMV20 case.

Fig. 19 shows the speedup relative to Sandy Bridge. For Sandy Bridge, 16 MPI tasks were used on a single node. In the case of KNL, we used 4 MPI tasks and 32 threads per MPI task (options `+ppn 32 +pemap 0-63+68 +commap 64-67`). For Skylake, we used 2 MPI tasks per node (options `+ppn 23 +pemap 0-45+48 +commap 46-47`). Overall, Skylake is 3.25 times faster than Sandy Bridge for this case.

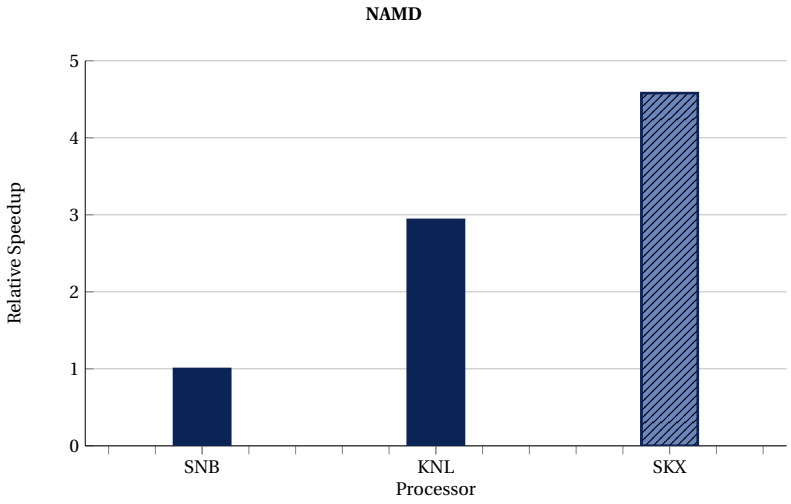

Figure 19: NAMD: Speedup Relative to SNB

Fig. 20 shows the impact of the different compilation flags in the performance. In this case, the `-O3 -xCORE-AVX512` provides the best result.

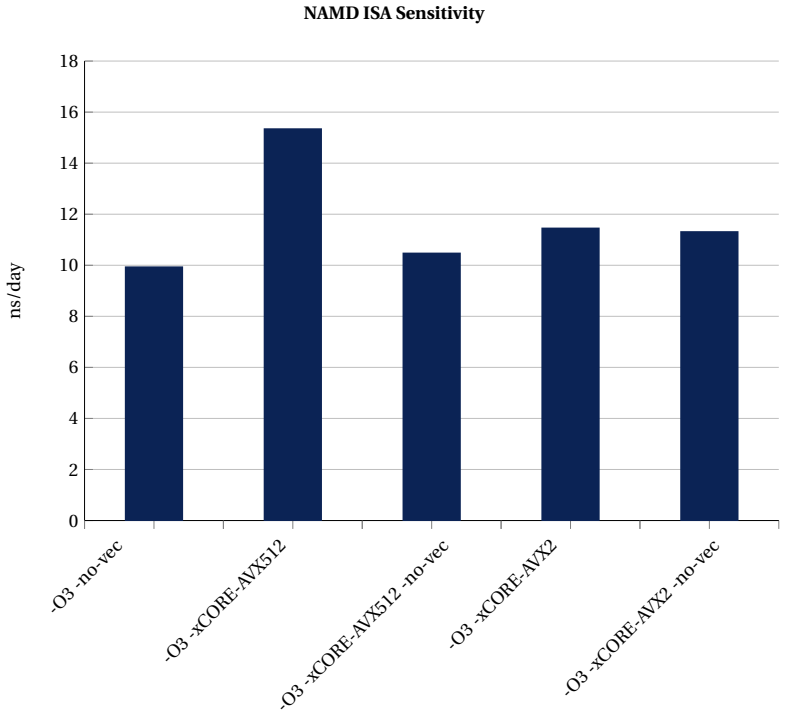

Figure 20: NAMD: ISA impact

6.4 GROMACS

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles [3]. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions. In this study, Hen egg white lysozyme (PDB code 1AKI) and water (SPC model) solutions were simulated using version 2016.3. The simulated systems consist a total of 140124 atoms for the single node test, and 1.8 million atoms for the scalability tests. All simulations were performed in the isothermal isobaric (NpT) ensemble at 300 K and 1 atm.

Fig. 21 shows the relative performance of GROMACS on a dual socket SKX node, a dual socket Sandy Bridge node, and a KNL. Overall, Skylake is 2.68 times faster than Sandy Bridge.

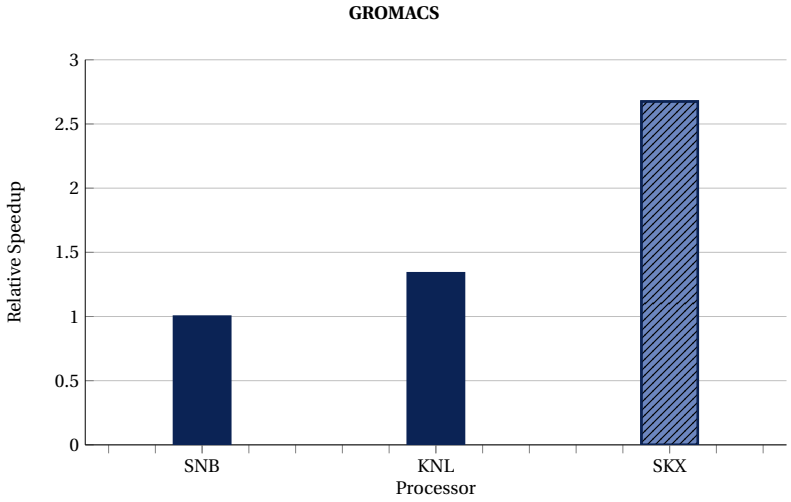

Figure 21: GROMACS: Speedup Relative to SNB

Fig. 22 depicts the performance of GROMACS for different compilation flags and ISA. It is worth noticing that GROMACS uses intrinsics for vectorization and that, at the time of running these tests, there was not an SKX optimized version of the code.

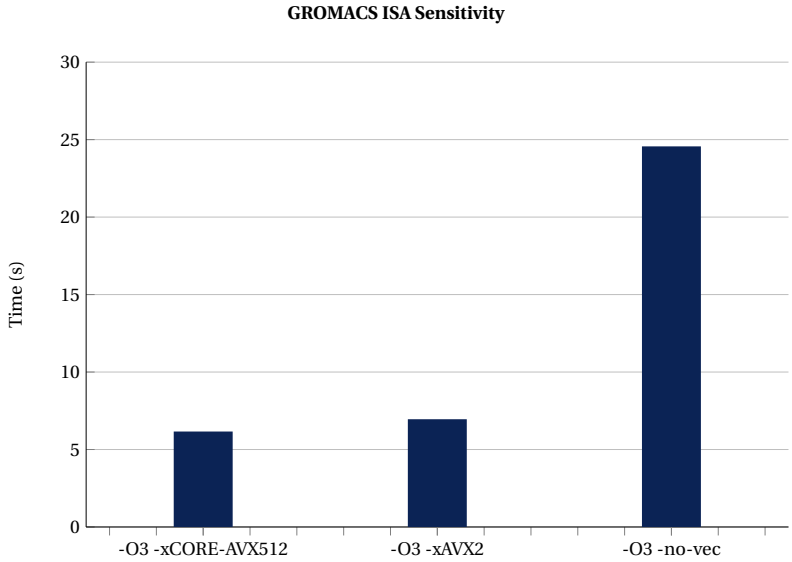

Figure 22: GROMACS: ISA impact

6.5 Python

We evaluated the performance of Python following the existing benchmarks at TACC [7] that can be found at [8]. These tests are sequential codes representative of the type of applications that many users of HPC systems run. Fig. 23 and 24 show the results of the Julia and Pairwise benchmarks with their different implementations in the Sandy Bridge, KNL and Skylake nodes. Since these are sequential codes, KNL provides the worst results. And even though the CPU frequency of a SKX core is lower than Sandy Bridge, the optimizations introduced in the hardware by Intel allow the codes to run faster than on SNB.

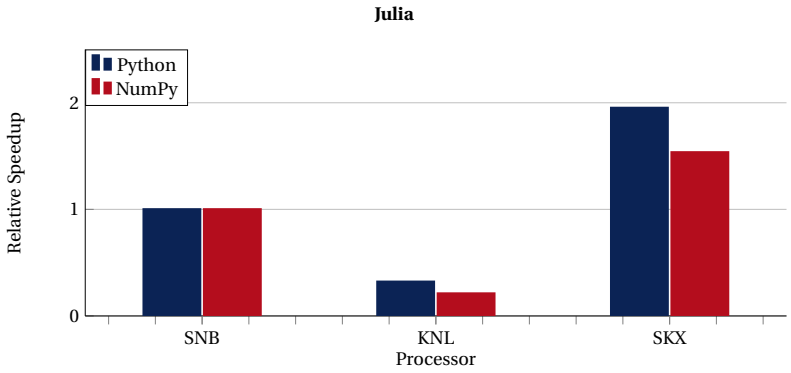

Figure 23: Julia: Speedup Relative to SNB

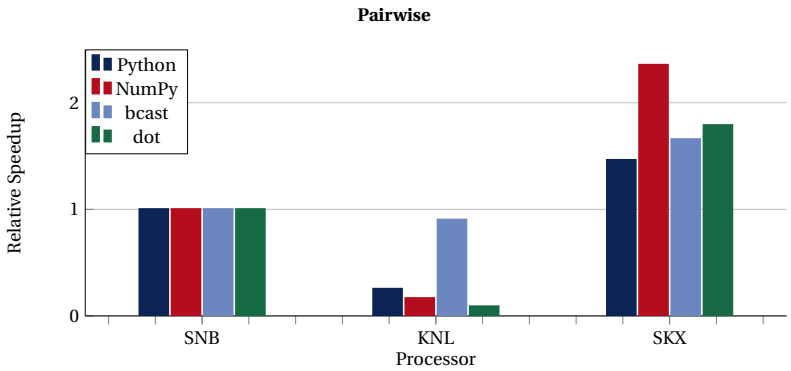

Figure 24: Pairwise: Speedup Relative to SNB

6.6 tblastx

One of the most popular tools for sequence similarity searches is the Basic Local Alignment Search Tool (BLAST) by NCBI [2]. We consider a threaded implementation of NCBI BLAST since the MPI version has not been actively maintained in several years. The results reported only consider the `tblastx` tool. Fig. 25 shows the relative performance to Sandy Bridge on the chips of interest. Because of the characteristics of the code, KNL performs poorly. And, at the same time, SKX runs 2.8 times faster than SNB.

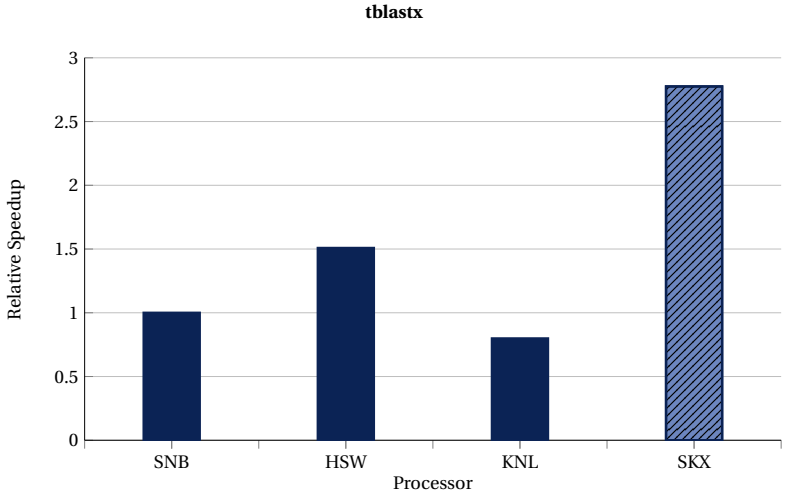

Figure 25: tblastx: Speedup Relative to SNB

REFERENCES

[1] TOP500 list. www.top500.org, 2017.

[2] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403 – 410, 1990.

[3] H.J.C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 91(1):43 – 56, 1995.

[4] J. Mark Bull and Darragh O'Neill. A microbenchmark suite for openmp 2.0. *SIGARCH Comput. Archit. News*, 29(5):41–48, December 2001.

[5] J. J. Dongarra. The linpack benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing*, pages 456–474, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[6] W. C. Skamarock et al. A description of the advanced research WRF version 3. Technical report, National Center for Atmospheric Research, 2008.

[7] Todd Evans, Antonio Gómez-Iglesias, and Cyrus Proctor. PyTACC: HPC Python at the Texas Advanced Computing Center. In *Proceedings of the 5th Workshop on Python for High-Performance and Scientific Computing, PyHPC 2015, Austin, Texas, USA, November 15, 2015*, pages 4:1–4:7. ACM, 2015.

[8] Antonio Gómez-Iglesias. Python benchmarks. https://bitbucket.org/agomezig/python-benchmarks, June 2015.

[9] John D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, 1991-2017. A continually updated technical report http://www.cs.virginia.edu/stream/.

[10] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kalé, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[11] David Radice, Sean M. Couch, and Christian D. Ott. Implicit large eddy simulations of anisotropic weakly compressible turbulence with application to core-collapse supernovae. *Computational Astrophysics and Cosmology*, 2(1):1–17, 2015.