Portable Methods for Measuring Cache Hierarchy Performance

Tom Deakin
University of Bristol
Department of Computer Science
Bristol, UK
tom.deakin@bristol.ac.uk

James Price
University of Bristol
Department of Computer Science
Bristol, UK
j.price@bristol.ac.uk

Simon McIntosh-Smith
University of Bristol
Department of Computer Science
Bristol, UK
s.mcintosh-smith@bristol.ac.uk

There has been a recent influx of different processor architecture designs into the market, with many of them targeting HPC applications. When estimating application performance, developers are used to considering the most common figures of merit, such as peak FLOP/s, memory bandwidth, core counts and so on. In this study we present a detailed comparison of on-chip memory bandwidths, including single core and aggregate across a node, for a set of next-generation CPUs. We do this in such a way as to be portable across difference architectures and instruction sets. Our study indicates that, while two processors might look superficially similar when only considering the common figures of merit, those two same processors might have radically different on-chip memory bandwidth, a fact which may be crucial when understanding observed application performance. Our results and methods will also be made public on GitHub in time for SC'17, allowing the procedure for evaluating cache bandwidths to be simplified for the community.

1 PROCESSOR ARCHITECTURE

We take the latest CPU offerings from different vendors: Intel and IBM. Aside from the headline grabbing differences such as clock speed, core count or use of MCDRAM (in the case of Intel Xeon Phi), the processors have somewhat subtle differences in their cache hierarchy. All the processors tested offer at least two levels of cache with some differences in capacity, as shown in Table 1. However it is also the number of load and store units in the architecture of the core itself, along with vector instruction set widths, which effects the bandwidth from the various levels of cache.

Intel Xeon (Broadwell). We tested the 18-core, E5-2695 v4 @ 2.1 GHz processor, in a dual-socket configuration. The core supports the 256-bit AVX2 instruction set. Each core contains two load/store units and one store unit [6].

Intel Xeon (Skylake). We tested the 20-core, Gold 6148 @ 2.4 GHz processor, in a dual-socket configuration. The core supports the AVX-512 instruction set. Each core contains two load/store units and one store unit [6].

Intel Xeon (Knights Landing). We tested the 64-core, 7120 @ 1.30 GHz processor. The core support the AVX-512 instruction set. Each core contains two load/store units [7].

IBM Power 8. We tested the 10-core @ 3.7 GHz processor, in a dual-socket configuration. The core supports 128-bit vector instructions [4]. Each core consists of four load units and 2 load/store units, however these are organised evenly into two pipelines.

SC'17, November 12–17, 2017, Denver, Colorado
2017. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 METHODOLOGY

Although there are a plethora of cache bandwidth benchmarks available (e.g. [1, 2]), they are typically written in assembly, targeting a particular vector instruction set or a single vendor. This means that we cannot use them on multiple architectures, nor vary the instruction set used; for example we wish to test the effect of vector width on cache bandwidth by varying the instruction set targeted by compiler auto-vectorisation. We therefore modify the BabelStream benchmark [3], which has previously been used to measure (main) memory bandwidth across diverse multi- and many-core architectures in a range of parallel programming models. Our modifications are based on the OpenMP version of BabelStream as this had the greatest coverage of all architectures tested.

Measuring cache bandwidth presented a number of challenges. Firstly the overheads of the OpenMP parallel region were much too large when running small array sizes designed to measure L1 bandwidth. We therefore had to remove the parallel aspect of the code and have a serial binary. In order to run on multiple cores therefore we use `mpirun` to run the benchmark on each core simultaneously, thus preventing the clock speed entering turbo mode on Intel architectures. We then collect the results from one core (the first).

Running all the BabelStream kernels in succession showed that the timer itself was causing overheads. We therefore run just Triad in a tight loop (running this one kernel many times) and time the total runtime, calculating the bandwidth from this rather than from each kernel individually as usual.

Non-temporal stores are usually enabled with STREAM to ensure best cache usage, however this means that the cache hierarchy is skipped and data is written directly to main memory. This causes the data to no longer be resident in a particular level of cache. We therefore ensured that non-temporal stores were not generated, via the use of a compiler flag (with the Intel compiler) or through use of a compiler which will not generate them anyway (GCC or XL). Note that when considering aggregate bandwidth from all cores accessing main memory, the lack of streaming stores results in a lower bandwidth than the standard BabelStream will report. This is because using cache for memory stores results in cache pollution; for this study we require data to stay in cache and therefore non-temporal stores inhibit residency.

We then ran the benchmark in the way just described on each processor, testing all array sizes from one to 2^{25} double

Processor	L1	L2	L3 (shared)	L4	DDR
Broadwell	32 KiB	256 KiB	45 MiB	-	64 GiB
Skylake	32 KiB	1024 KiB	27.5 MiB	-	96 GiB
Knights Landing	32 KiB	1024 KiB (shared per tile)	16 GiB (MCDRAM)	-	96 GiB
Power 8	64 KiB	512 KiB	16 MiB	128 MiB	256 GiB

Table 1: Cache capacity of processors

precision elements in order to capture bandwidth from all cache levels. We also repeated the experiment targeting each of the supported instruction sets; for example we measured cache bandwidths on Skylake for 512-bit vectors (AVX512), 256-bit vectors (AVX/AVX2), 128-bit vectors (SSE4.2) and no vectorisation. Our approach allows the targeting of many architectures and instruction sets which would be infeasible with benchmarks written in assembly or intrinsic functions.

3 FINDINGS

When considering the bandwidth available to each core from different levels of cache, the number of load and store units in the core as well as the clock speed and vector widths greatly affect the bandwidth, particularly when considering L1 bandwidth. Xeon cores have two load and one store units per core, and along with the FMA unit should therefore be able to perform one iteration of the Triad kernel in a single clock cycle assuming the work has been pipelined optimally. However the number of address translation units may limit this [5]. On Power 8 whilst there are twice as many units available per core compared to Xeon, due to the processor being configured in SMT8 mode, we believe only half are available despite running a single thread per physical core [8].

The number of memory operations possible per clock cycle is equal to the total number of load/store units, and therefore by multiplying by the clock speed and the transfer size (typically the vector width) the theoretical memory bandwidth can be obtained. However running on a single core causes the clock frequency to increase, while conversely running vector instructions may cause the clock speed to be reduced. These factors highlight the importance of using all cores simultaneously in these measurements.

We also observed that using AVX-512 instructions showed a big increase in measured L1 bandwidth compared to shorter vector widths, however this effect is reduced at each subsequent higher level of cache.

The capacity of the cache also shows that for arrays of fixed size, different processors can offer increased bandwidth as a result of residency in the hierarchy. For example the Power 8 has 64 KiB of L1 and therefore the Triad arrays of size 2048 doubles (totalling 48 KiB) fit in L1 cache on this processor achieving 93 GB/s bandwidth per core; whereas on Broadwell with 32 KiB L1 cache these arrays reside in L2 and for this problem size 46 GB/s per core L2 bandwidth is available. This therefore shows that optimisations such as cache blocking to fit in lower levels of cache may be less beneficial for applications running on processors with larger cache capacities as the data may already be resident.

We also consider the aggregate bandwidth by multiplying our measurements on a single core by the number of cores (recall that these measurements were taken when all the cores were being used simultaneously). This aggregate bandwidth shows the total memory bandwidth through the processor (processors on a dual-socket system except Knights Landing). For processors such as the Intel Xeon Phi with a relatively low clock speed, and therefore low single core bandwidth, due to the number of cores the aggregate bandwidth is similar to that available from Intel Xeon processors at L1 cache, and competitive at higher levels of cache.

REFERENCES

[1] 2013. pmbw — Parallel Memory Bandwidth Benchmark / Measurement. https://panthema.net/2013/pmbw/. (2013).

[2] 2017. LIKWID — Performance monitoring and benchmarking suite. https://github.com/RRZE-HPC/likwid. (2017).

[3] Tom Deakin, James Price, Matt Martineau, and Simon McIntosh-Smith. 2017. Evaluating attainable memory bandwidth of parallel programming models via BabelStream. *International Journal of Computational Science and Engineering* (2017).

[4] Michael Gschwind. 2016. Workload acceleration with the IBM POWER vector-scalar architecture. 60 (03 2016), 14:1–14:18.

[5] Johannes Hofmann, Georg Hager, Gerhard Wellein, and Dietmar Fey. 2017. An Analysis of Core- and Chip-Level Architectural Features in Four Generations of Intel Server Processors. Lecture Notes in Computer Science, Vol. 10266. Springer International Publishing, Cham, 294–314.

[6] Intel Corporation 2016. *Intel 64 and IA-32 Architectures Optimization Reference Manual* (248966-033 ed.). Intel Corporation.

[7] Jim Jeffers, James Reinders, and Avinash Sodani. 2016. Knights Landing architecture. In *Intel Xeon Phi Processor High Performance Programming* (second ed.), Jim Jeffers, James Reinders, and Avinash Sodani (Eds.). Morgan Kaufmann, Boston, 63–84.

[8] Balaram Sinharoy, JA Van Norstrand, Richard J Eickemeyer, Hung Q Le, Jens Leenstra, Dung Q Nguyen, B Konigsburg, K Ward, MD Brown, José E Moreira, et al. 2015. IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development* 59, 1 (2015).

[9] W. J. Starke, J. Stuecheli, D. M. Daly, J. S. Dodson, F. Auernhammer, P. M. Sagmeister, G. L. Guthrie, C. F. Marino, M. Siegel, and B. Blaner. 2015. The cache and memory subsystems of the IBM POWER8 processor. *IBM Journal of Research and Development* 59, 1 (Jan 2015), 3:1–3:13.