Portable methods for measuring cache hierarchy performance

University of BRISTOL
Department of Computer Science

Tom Deakin, James Price and Simon McIntosh-Smith
tom.deakin@bristol.ac.uk

UOB HPC

BabelStream

- Portable version of the STREAM benchmark
- Designed to work across different multi- and many-core devices
- Code and results are open source:

http://uob-hpc.github.io/BabelStream/

Methodology and challenges

- There are lots of cache bandwidth benchmarks, but mostly written in platform-specific **assembly code** – these are no good for cross-platform comparisons
- Want bandwidth from all cores, but OpenMP parallel overheads are too large during fine measurements
- No way to explicitly pin data in a specific level of cache
- Validation of measurement – base on theoretical limits

- We run the STREAM Triad kernel in a tight loop to ensure cache residency:

```
start_timer();
for (int t = 0; t < ntimes; t++)
  for (int i = 0; i < n; i++)
    a[i] = b[i] + scalar * c[i];
end_timer();
```

- We adapt BabelStream to this end:
  - Base on original C++ and OpenMP version
  - Use 2MB aligned arrays
  - Remove OpenMP, as overheads too large
  - Instead run simultaneously on each physical core (via mpirun) – needed to prevent Turbo clock speeds
  - Do not use hardware threads
  - Avoid streaming-stores so that writes to cache remain resident at the correct level
  - Collect result from one core (from all running)

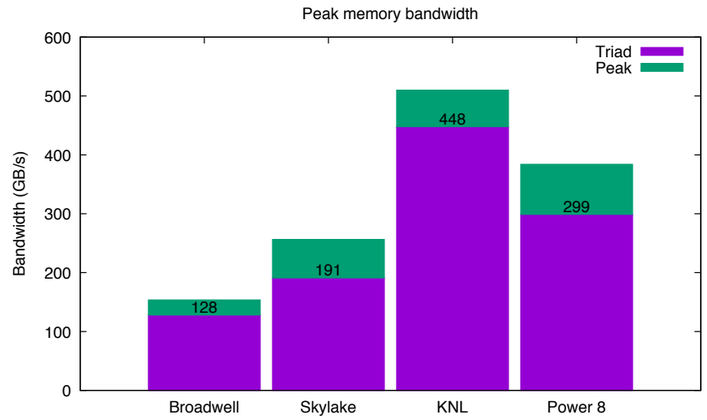
- Run on a variety of problem sizes to capture performance of each cache level

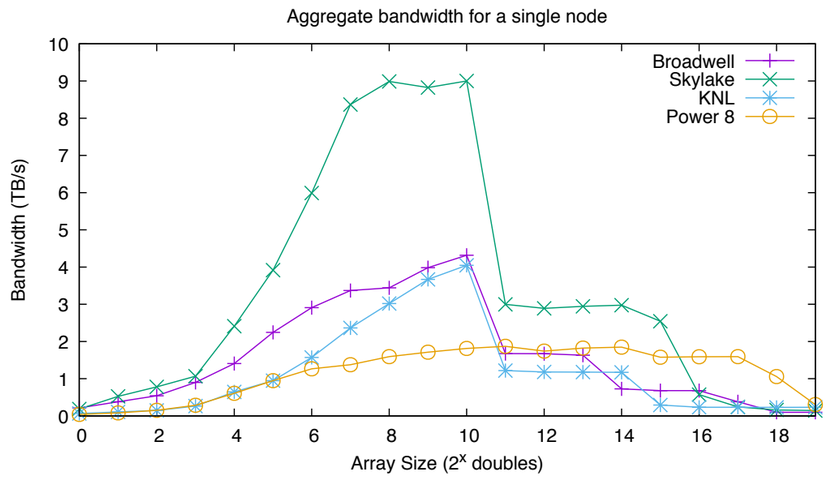
Allows for an identical code base for cross-platform, cross-architecture, reproducible, cache bandwidth benchmarking

Hardware

- Intel Xeon Skylake (dual-socket)
  - 20-core Gold 6148 @ 2.4 GHz
  - 512 bit vectors
- IBM Power 8 (dual-socket)
  - 10 core @ 3.7 GHz
  - 128 bit vectors
- Intel Xeon Broadwell (dual-socket)
  - 18 core E5-2695 v4 @ 2.1 GHz
  - 256 bit vectors
- Intel Xeon Phi Knights Landing (single socket)
  - 64 core 7210 @ 1.3 GHz
  - 512 bit vectors

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





Micro-architecture effect on cache bandwidth

- Load/store units
  - The number and width of load/store units effects the theoretical cache bandwidth
  - Xeon: 2 loads + 1 store per core
  - Power 8: 2 loads + 2 load/stores per core, but only half per thread in SMT modes
  - More load/store units implies higher potential cache bandwidth
- Clock speed
  - With faster clock speeds, load/store units and caches run faster
  - Therefore higher clocks imply more cache bandwidth
- Cache capacity
  - Larger arrays stay in cache longer with bigger caches
  - Higher bandwidth measured for a fixed size
  - For example, when the three arrays are each of size 2^{11}, they are still in L1 on Power 8, but fall out to L2 on the other devices

Effect of vector widths

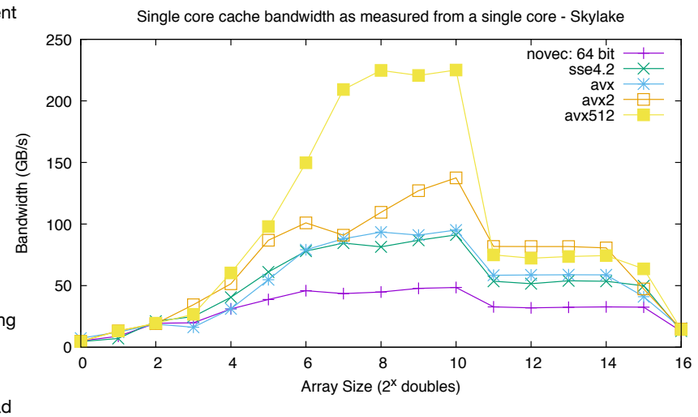
- Use compiler auto-vectorisation to target different instruction sets
- AVX-512: 512 bit (8 doubles)
- AVX2: 256 bit (4 doubles), plus FMA
- AVX: 256 bit (4 doubles), no FMA
- SSE4.2: 128 bit (2 doubles)
- No vectorisation

- One AVX-512 load instruction loads an entire cache line

- Measured L1 bandwidth highly dependent on vector width

- Less difference in L2, as long as we're vectorising

- The FMA instruction in AVX2 helps improve bandwidth to L1 as only one floating point instruction between each load instruction instead of two



Aggregate cache bandwidth

- Shows aggregate bandwidth from a dual-socket node in all cache levels
  - KNL single socket only
  - Assuming a future dual-socket ThunderX2
- Multiply core count by single core bandwidth measurement to get aggregate
  - Single core bandwidth measured but ran on all cores
  - Higher core count results in larger aggregate bandwidth, e.g. KNL
- Little difference in bandwidth between L1 and L2 on Power 8
- KNL and Broadwell show similar L1 aggregate bandwidth despite disparity in clock speed
  - Skylake has same vector width as KNL but faster clock
- Far RHS would be main memory bandwidth
  - As non-temporal stores disabled results would misrepresent expected figures
  - Bar chart above gives the true main memory bandwidth