Analysis of GCC C++ and Intel C++ Stream Benchmark on KNL

This is a report detailing the analysis of data from GCC C++ and Intel C++ compilers respectively for Stream Benchmark on KNL with a minimum and maximum number of loop iterations set to 30 and 16,000,000 respectively. Also, to stretch the performance of these compilers, the maximum number of loop iterations was doubled, as such set to 32,000,000 loop iterations.

Outputs from these operations were plotted on a two-dimensional graph with Size/Array Length (number of loop iterations) plotted on the primary horizontal axis and Bandwidth (Fill (MB/s), Copy, AXPY and DOT respectively) on the vertical axis. Computation of the performance for both GCC and Intel++ compilers using before and after peak values for Fill, Copy, AXPY and DOT to compare L1 cache and L2 cache are shown below as follows;

Fill: x [i] = alpha
Fill is performing 2 memory operations. 1 read and 1 write which is actually 2 floating point operations using doubles for precision. This can be expressed mathematically as;
(Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Copy: x [i] = y [i]
Also, Copy is performing 2 memory operations. 1 read and 1 write which is again equivalent to 2 floating point operations using doubles for precision. It can be expressed mathematically as;
(Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

AXPY: y [i] = y [i] + alpha * x [i]
AXPY is performing 3 memory operations. 2 reads and 1 write which is equal to 3 floating point operations using doubles for precision. It can be expressed mathematically as;
(Number of iterations) * (8 bytes per double) * (3 doubles per iteration)

DOT: alpha = sum_i (x [i] * y [i])
Again, DOT is performing 2 memory operations. 1 read and 1 write which is equal to 2 floating point operations using doubles for precision. It can be written mathematically as;
(Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

NB: Before Peak values = L1 Cache
After Peak values = L2 Cache

GCC C++ Stream Benchmark on KNL (Maximum number of iterations = 16,000,000)

Fill: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 88722
Fill before peak = 88722 * 8 *1
= 709776 bytes ≈ 709kb

After peak, number of iterations = 132305
Fill at after peak = 132305 * 8 * 1
= 1,058,440 bytes ≈ 1mb

Copy: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 1094
Copy at before peak = 1094 * 8 * 2
= 17,504 bytes ≈ 17.5kb
After Peak, number of iterations = 2433
Copy after Peak = 2433 * 8 * 2
= 38,928 bytes ≈ 38.9kb

AXPY: (Number of iterations) * (8 bytes per double) * (3 doubles per iteration)

Before peak, number of iterations = 3065
AXPY at before peak = 3065 * 8 * 3
= 73,560 bytes ≈ 73.5kb
After peak value = 2013 iterations
AXPY at after peak = 2013 * 8 * 3
= 48,312 bytes ≈ 48kb

DOT: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 3628
DOT at before peak = 3628 * 8 * 2
= 58048 bytes ≈ 58kb
After peak, number of iterations = 88722
DOT after peak = 88722 * 8 * 2
= 1419552 bytes ≈ 1.4mb

INTEL C++ Stream Benchmark on KNL (Maximum number of iterations = 16,000,000)

Fill: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 3628
Fill before peak = 3628 * 8 *1
= 29024 bytes ≈ 29kb
After peak, number of iterations = 5410
Fill at after peak = 5410 * 8 * 1
= 43280 bytes ≈ 43kb

Copy: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 734
Copy at before peak = 1094 * 8 * 2

= 17504 bytes ≈ 17.5kb

After Peak, number of iterations = 2433

Copy after Peak = 2433 * 8 * 2

= 38,928 bytes ≈ 38.3kb

AXPY: (Number of iterations) * (8 bytes per double) * (3 doubles per iteration)

Before peak, number of iterations = 1094

AXPY at before peak = 1094 * 8 * 3

= 26256 bytes 26KB

After peak value = 3628 iterations

AXPY at after peak = 3628 * 8 * 3

= 87072 bytes ≈ 87kb

DOT: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 39897

DOT at before peak = 39897 * 8 * 2

= 638352 bytes ≈ 0.6mb

After peak, number of iterations = 88722

DOT after peak = 88722 * 8 * 2

= 1419552 ≈ 1.4mb

GCC C++ Stream Benchmark on KNL (Maximum number of iterations = 32,000,000)

Fill: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 88676

Fill before peak = 88676 * 8 *1

= 177352 bytes ≈ 177kb

After peak, number of iterations = 205655

Fill at after peak = 205655 * 8 * 1

= 1645240 bytes ≈ 1.6mb

Copy: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 1322

Copy at before peak = 1322 * 8 * 2

= 21152 bytes ≈ 21kb

After Peak, number of iterations = 3065

Copy after Peak = 3065 * 8 * 2

= 49040 bytes ≈ 49kb

AXPY: (Number of iterations) * (8 bytes per double) * (3 doubles per iteration)

Before peak, number of iterations = 38236

AXPY at before peak = 38236 * 8 * 3

= 917664 bytes ≈ 0.9mb

After peak value = 88676 iterations

AXPY at after peak = 88676 * 8 * 3

= 2128224 bytes ≈ 2mb

DOT: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 58229

DOT at before peak = 58229 * 8 * 2

= 931664 bytes ≈ 0.93mb

After peak, number of iterations = 88676

DOT after peak = 88676 * 8 * 2 = 1418816 bytes ≈ 1.4mb

INTEL C++ Stream Benchmark on KNL (Maximum number of iterations = 32,000,000)

Fill: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 3065

Fill before peak = 3065 * 8 *1

= 24520 bytes ≈ 24kb

After peak, number of iterations = 4668

Fill at after peak = 4668 * 8 * 1

= 37344 bytes ≈ 37kb

Copy: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 868

Copy at before peak = 1322 * 8 * 2

= 21152 bytes ≈ 21kb

After Peak, number of iterations = 3065

Copy after Peak = 3065 * 8 * 2

= 49040 bytes ≈ 49kb

AXPY: (Number of iterations) * (8 bytes per double) * (3 doubles per iteration)

Before peak, number of iterations = 2013

AXPY at before peak = 2013 * 8 * 3

= 48312 bytes ≈ 48kb

After peak value = 3065 iterations

AXPY at after peak = 3065 * 8 * 3

= 73560 ≈ 73.5 kb

DOT: (Number of iterations) * (8 bytes per double) * (2 doubles per iteration)

Before peak, number of iterations = 38236

DOT at before peak = 38236 * 8 * 2

= 458832 bytes ≈ 45.9kb

After peak, number of iterations = 88676

DOT after peak = 88676 * 8 * 2

= 1418816 bytes ≈ 1.4mb

From the above calculations, it is clear that the before and after peak values of Fill, Copy, AXPY and DOT does not exactly match the expected L1 and L2 cache values of the GCC and Intel compilers which are equally 32K and 1024K for L1 and L2.

However, we clearly experience more optimized computation from the Intel++ compiler. This is especially obvious in AXPY, since it is performing 3 memory operations – 2 reads and 1 write as it is multiplying and adding as well as reading to memory. Even though the L1 and L2 cache does not exactly match, AXPY gets more effective bandwidth because it is performing more computations, hence the CPU tends to be more at work. Also, most modern hardware are designed to perform 2 reads and 1

write to memory efficiently. Thus, we can infer that AXPY has been optimized in hardware.

In the case of Fill, Copy and DOT, they are mostly reading chunks of data from memory and writing them back to another location in memory. The CPU is basically inert and not very good at moving data, rather it is great at computing data(numbers).

We can conclude that any time we tend to only read and write data, odds are that we are most likely, not going to get a good bandwidth, whereas if we were performing computations, the reverse is likely to be the case.

Referring to the Intel++ Compiler, just after peak values for Fill, Copy, AXPY and DOT (3065, 1322, 2013 and 38236) we experience a sharp decline and at after peak values (4668, 3065, 3065 and 88676) a plato/flat line. This is largely because all data is moved from L1 cache into L2 cache and instead of an increase in speed we experience a decline. The bandwidth's performance is dependent on how fast we can move data from L1 into L2 and not the operations being performed with the data.

After points 88676, 38236, 38236 and 38236 for Fill, Copy, AXPY and DOT we see a similar decline and at 205655, 205655, 88676 and 88676 for Fill, Copy, AXPY and DOT respectively we see similar flat line denoting a transition of data from L2 cache into main memory.