Theoretical Peak FLOPS per instruction set on less conventional hardware

Romain Dolbeau
Bull – Center for Excellence in Parallel Programming
Email: romain.dolbeau@atos.net

***Abstract*—This is a companion paper to "Theoretical Peak FLOPS per instruction set on modern Intel CPUs" [1]. In it, we survey some alternative hardware for which the peak FLOPS can be of interest. As in the main paper, we take into account and explain the peculiarities of the surveyed hardware.**

Revision 1.16, 2016/10/04 08:40:17
***Index Terms*—FLOPS**

I. Introduction

Many different kind of hardware are in use to perform computations. No only conventional Central Processing Unit (CPU), but also Graphics Processing Unit (GPU) and other accelerators. In the main paper [1], we described how to compute the peak FLOPS for conventional Intel CPUs. In this extension, we take a look at the peculiarities of those alternative computation devices.

II. Other CPUs

A. AMD Family 15h

The AMD Family 15h (the name "15h" comes from the hexadecimal value returned by the CPUID instruction) was introduced in 2011 and is composed of the so-called "construction cores", code-named Bulldozer, Piledriver, Steamroller and Excavator. They are sold under different brands including the server-oriented Opteron brand and the consumer-oriented Athlon and FX brand.

The specificity of the Bulldozer micro-architecture [2][3] with regards to the Floating-Point Unit (FPU) is its shared nature. In Bulldozer and its offsprings, each pair of cores share a pair of 128 bits wide FPU. Unlike Simultaneous Multi-Threading (a.k.a. HyperThreading), most of the resources are duplicated - each core has a full scheduler, integer pipeline, and so on, an approach popular at the time [4][5][6]. Only the FPUs are shared in Bulldozer. We can take a look back at the equation 1, replicated from the main paper, and see how this affects the peak FLOPS.

$$\frac{flops}{node} = \left.\begin{array}{r} \frac{flop}{operation} \\ \times \frac{operations}{instruction} \\ \times \frac{instructions}{cycle} \end{array}\right\} micro-architecture \quad \left.\begin{array}{r} \times \frac{cycles}{second} \\ \times \frac{cores}{socket} \\ \times \frac{sockets}{node} \end{array}\right\} machine \quad architecture \tag{1}$$

For the micro-architecture parts ($^{flop}/_{operation}$, $^{operations}/_{instruction}$, $^{instructions}/_{cycle}$), we need to take into account the fact that each FPU pipeline is only 128 bits wide. In the original architecture, Bulldozer, all of the extensions to SSE are supported, plus AVX. Bulldozer also support an FMA4 extension, which uses different instructions to achieve the same results as the FMA (a.k.a. FMA3) used in newer CPUs from Intel and AMD. The results for the micro-architecture are summarized in table I. As can be seen in the table, the number of $^{instructions}/_{cycle}$ is halved for both full-width AVX+FMA cases (SP and DP on a 256 bits vector),

since in this case Bulldozer requires both 128 bits pipelines to execute the 256 bits instruction. Like Haswell, results for AVX without FMA4 are exactly half that for AVX when using FMA4.

The second aspect is the fact that the FPUs are shared between pair of cores. When using more than one core - i.e. computing the per-node peak - that means it's not possible to simply multiply by the number of cores. Either the number of modules (two cores each) should be used, or a "sharing" factor of $^1/_2$ should be added. In the table II, we use a sharing factor. The table describe a node using four Opteron 6276, the maximum number of socket for this processor. The node has therefor 64 cores, but only 32 shared pair of 128 bits pipelines. Despite twice as many sockets, more than twice as many cores, and the same nominal frequency, the full node based on this processor has less than 60% of the peak performance at the nominal frequency of the much newer Intel Haswell E5-2695v3 studied in the main paper. This is because in effect, averaged across all cores, Bulldozer can only do 8 $^{flop}/_{cycle}$ vs. 32 for Haswell.

The last micro-architecture from the family is Excavator, which has support for the FMA(3) extension to AVX in addition to FMA4; this does not change the peak. The upcoming Zen micro-architecture will likely retain the pair of 128 bits wide FMA-capable pipelines, but will not share them between cores, thus doubling the peak at node level with the same number of core at the same frequency.

B. Intel Knights Landing

Intel Knights Landing is the code-name for the Intel Xeon Phi 72xx family of processors. Knights Landing offers all instructions sets from the Haswell (and its successor Broadwell with the exception of TSX), with the addition of the new AVX-512 instruction set. AVX-512 leverages a new encoding scheme called EVEX, which fortunately is compatible with the VEX encoding scheme of AVX, so both can cohabitate in the same program. Knights Landing offers the same theoretical $^{flop}/_{cycle}$ for the SSE and AVX instruction sets than Haswell (and Broadwell), but with slightly longer producer-consumer latencies [7][8]. Knights Landing also offers two full-width AVX-512 pipelines, allowing up to double the $^{flop}/_{cycle}$ when the new instructions are in use. Knights Landing can therefore do up to 32 $^{flop}/_{cycle}$ in double-precision, and up to 64 $^{flop}/_{cycle}$ in single-precision. In a manner similar to Haswell, Knights Landing has both a Turbo frequency (higher than nominal) and a base AVX frequency (lower than nominal).

C. ARM Cortex A57

The ARM Cortex A57 [9] is an implementation of the ARM v8 [10] architecture. Floating-point operations in ARM v8 are done with the NEON instructions set. Like SSE, NEON has both scalar and vector instructions. Unlike SSE, the vector variants exist in both 64 bits (called the D form, referring to a double-word where a word is 32 bits) or 128 bits (called the Q form, referring to a quadruple-word). Registers are always 128 bits, and in ARM v8 the D form (64 bits) instructions works on the lower half of the 128 bits registers, never the high part.

In the Cortex A57, there is two 64 bits pipelines for NEON floating-point operations. If D form instructions are used, they can be dispatched to both pipelines simultaneously. If Q form instructions are used, then they are dispatched to both pipelines simultaneously, aggregating the two 64 bits pipeline into a single 128 bits pipeline. This is the same mechanism used by AMD in the Bulldozer, but with half the width. As shown in table III, the peak in practice is therefor the same for the D and Q forms. The only difference is that in the Q form, twice as much register space is available. And since the D form is effectively scalar for double-precision operations, there is no theoretical gain from vectorization on the A57 in double-precision.

TABLE III
THEORETICAL PER-CYCLE PEAK FOR CORTEX A57

	NEON (Scalar)	NEON D (DP)	NEON D (SP)	NEON Q (DP)	NEON Q (SP)
$^{flop}/_{operation}$	2	2	2	2	2
$\times^{operations}/_{inst.}$	1	1	2	2	4
$\times^{instructions}/_{cycle}$	2	2	2	1	1
$=^{flop}/_{cycle}$	4	4	8	4	8

TABLE I
THEORETICAL PER-CYCLE PEAK FOR BULLDOZER

	SSE (Scalar)	SSE (DP)	SSE (SP)	AVX+FMA4 (scalar)	AVX-128 +FMA4 (DP)	AVX-128 +FMA4 (SP)	AVX+FMA4 (DP)	AVX+FMA4 (SP)
$^{flop}/_{operation}$	1	1	1	2	2	2	2	2
$\times^{operations}/_{inst.}$	1	2	4	1	2	4	4	8
$\times^{instructions}/_{cycle}$	2	2	2	2	2	2	1	1
$=^{flop}/_{cycle}$	2	4	8	4	8	16	8	16

TABLE II
THEORETICAL PER-NODE PEAK FOR OPTERON 6276 (NOMINAL FREQUENCY)

	SSE (Scalar)	SSE (DP)	SSE (SP)	AVX+FMA4 (scalar)	AVX-128 +FMA4 (DP)	AVX-128 +FMA4 (SP)	AVX+FMA4 (DP)	AVX+FMA4 (SP)
$^{flop}/_{cycle}$	2	4	8	4	8	16	8	16
$\times^{cycles}/_{second}$	$2.3G$	$2.3G$	$2.3G$	$2.3G$	$2.3G$	$2.3G$	$2.3G$	$2.3G$
$\times^{cores}/_{socket}$	16	16	16	16	16	16	16	16
$\times sharing$	$^1/_2$	$^1/_2$	$^1/_2$	$^1/_2$	$^1/_2$	$^1/_2$	$^1/_2$	$^1/_2$
$\times^{sockets}/_{node}$	4	4	4	4	4	4	4	4
$=^{flops}/_{node}$	$147.2G$	$294.4G$	$588.8G$	$294.4G$	$588.8G$	$1177.6G$	$588.8G$	$1177.6G$

D. AppliedMicro X-Gene

The AppliedMicro X-Gene 1 and X-Gene 2 [11] are both, like the A57, implementation of the ARM v8 architecture. They share instructions sets and registers with the A57. However, the design of the X-Gene 1 only has a single 64 bits pipeline for floating-point operations. The Q form instructions require two consecutive cycles to execute. As shown in table IV, the X-Gene 1 only has half the per-cycle peak of the A57, and also does not benefit from vectorization in double-precision other than the gain in register space.

TABLE IV
THEORETICAL PER-CYCLE PEAK FOR X-GENE 1

	NEON (Scalar)	NEON D (DP)	NEON D (SP)	NEON Q (DP)	NEON Q (SP)
$^{flop}/_{operation}$	2	2	2	2	2
$\times^{operations}/_{inst.}$	1	1	2	2	4
$\times^{instructions}/_{cycle}$	1	1	1	$^1/_2$	$^1/_2$
$=^{flop}/_{cycle}$	2	2	4	2	4

E. Future Scalable Vector Extensions to ARM v8

In 2016, ARM announced the Scalable Vector Extensions [12], an new instruction set for the ARM v8 architecture. With registers up to 2048 bits wide and FMA support, this new instruction set could bring very high theoretical peak performance. However, each manufacturer will have to select a register width between 128 and 2048 bits, and may or may not choose to implement multiple instructions per cycle. Until some specific micro-architecture is released, it is impossible to quantify what performance SVE will offer in practice.

III. NVIDIA GPUS

Computing the number of theoretical peak FLOPS for NVidia GPU should be complicated, since one can argue about the semantic of terms such as core, thread and vectorization in the context of GPGPU programming. However, NVidia makes it much easier by fully documenting the implementation of its GPUs. The CUDA C Programming Guide [13] contains detailed description of the various generations of GPU, with code-names such Fermi, Maxwell or Pascal. Each GPU can report its "CUDA Compute Capability", which is effectively its micro-architecture. For each "Compute Capability", NVidia details how many instructions of different kind a single "multiprocessor" can do. A "Multiprocessor", in this context, is the base unit from which the actual GPUs are built. So from the

Fig. 1. Extract from the CUDA documentation

Table 2. Throughput of Native Arithmetic Instructions. (Number of Operations per Clock Cycle per Multiprocessor)

	Compute Capability								
	2.0	2.1	3.0, 3.2	3.5, 3.7	5.0, 5.2	5.3	6.0	6.1	6.2
16-bit floating-point add, multiply, multiply-add	N/A	N/A	N/A	N/A	N/A	256	128	2	256
32-bit floating-point add, multiply, multiply-add	32	48	192	192	128	128	64	128	128
64-bit floating-point add, multiply, multiply-add	16[1]	4	8	64[2]	4	4	32	4	4

table in section 5.4.1 of the CUDA C Programming Guide [14], one can look at the lines:

- "32-bit floating-point add, multiply, multiply-add" for single-precision operations;
- "64-bit floating-point add, multiply, multiply-add" for double-precision operations[1].

The relevant extract from the documentation is shown in figure 1. The number in this line and in the column for the chosen "Compute Capability" effectively indicates the product ${}^{operations}/_{instruction} \times {}^{instructions}/_{cycle}$. Since all NVidia GPUs implements a fused multiply-add, ${}^{flop}/_{operation}$ is always two. The frequency and the number of multiprocessors indicated in the specifications of the specific device will supply ${}^{cycles}/_{second}$ and $\times {}^{cores}/_{socket}$. The "Compute Capability" for each device are listed in NVidia website [15]. If the number of multiprocessors is not explicitly listed, it can be found by software or can be computed by dividing the number of "CUDA cores" from the specifications by the line "32-bit floating-point add, multiply, multiply-add" from [14].

In single- and double-precision, all instructions are scalar on the NVidia GPUs, thus simplifying the discussion. However, the Maxwell micro-architecture introduced hardware half-precision instructions in "Compute Capability 5.3". The CUDA C Programming Guide announces twice as many instructions per cycle in half-precision as in single-precision, but this is misleading. In fact, the throughput in instructions is the same, but the "Compute Capability 5.3" hardware introduces support for vector instructions of vector length 2 in registers of 32 bits. If those instructions are not used (i.e. there is no vectorization, only the thread-level parallelism inherent to kernel programming in CUDA), then the peak FLOPS in half-precision is the same as in single-precision, not twice as much.

As an example, one can consider the GeForce GTX 980 device, a gaming GPU of the Maxwell micro-architecture ("Compute Capability 5.2"). As such, it has 128 SP FMA per cycle per multiprocessor, 4 (!) DP FMA per cycle per multiprocessor, 2048 "CUDA cores" translating to ${}^{2048}/_{128} = 16$ multiprocessors and a base clock of 1064 MHz. It's theoretical peak is therefore:

SP $2 \times 128 \times 16 \times 1064$, or approximately $4.36 Gflop/s$;

DP $2 \times 4 \times 16 \times 1064$, or approximately $136 Mflop/s$.

IV. FPGA

A field-programmable gate array (FPGA) is a compute device that can be reconfigured, i.e. the hardware itself can be "programmed" for specific functions. As such, the amount of floating-point capability is highly variable. A given FPGA has a fixed amount of resources, which will be used to implement the various functions needed. Only a fraction of the resources will be used for floating-point. And since the amount of resources is dependent on accuracy [16], the implementations details [17], etc., it is impossible to give a peak number for any given FPGA. One need to first implement and synthesize the design, and then compute the peak for this design from the attainable frequency and the amount of operators used in the design.

V. Conclusion

In this companion paper to "Theoretical Peak FLOPS per instruction set on modern Intel CPUs" [1], we take a quick look at some alternative hardware. Merged pipelines, half-width execution

[1]Beware that consumer-grade GPU might have degraded double-precision performance compared to their compute-oriented siblings; this is documented in footnotes of the aforementioned table.

units, shared resources between cores and unexpected ratio of single- to double-precision performance make computations of theoretical FLOPS an interesting exercise that requires some understanding of the underlying micro-architecture.

GLOSSARY

AVX The third SIMD instruction set for the x86 architecture. It uses 256 bits wide registers. The original AVX instruction set only support floating point operations. AVX2 introduced integer operations. 1, 5

AVX2 First extension to the AVX instruction set, adding integer operations. 5

Central Processing Unit The part of a computer or computing device that executes the instructions of one or more programs. In the modern era, CPUs are composed of multiple cores, plus supporting functions such as shared caches, I/O functions, and so on. The physical CPU is commonly fitted in a socket. 1, 5

core The base hardware unit required to execute a program. Each core has access to all subcomponents needed to execute user-land programs: ALU, FPU, branch, load & store, etc. A single core may have the ability to run more than one program simultaneously, in which case the core resources are shared (statically or dynamically) between the programs. This is called Simultaneous Multi-Threading, and is known commercially as HyperThreading for Intel processors. 1, 5

CPU Central Processing Unit. 1

CPUID instruction The CPUID instruction is a mechanism for x86 and x86-64 architectures by which software can tell among other things which instruction sets are supported in a particular CPU. 1

encoding scheme The encoding is the binary representation of an instruction, directly usable by the processor. An encoding scheme is the description of how to encode the various instructions. The AVX encoding scheme differs from the SSE encoding scheme, as it adds some extra bytes to enable the use of wider registers and extra operands. 2

Floating-Point Unit The computation unit responsible for floating-point operations. It can be composed of multiple execution units, each of which are sometimes also called FPU. It can be scalar (such as the x87 FPU, or SIMD (such as SSE or AVX). 1, 5

FPU Floating-Point Unit. 1, 5

Graphics Processing Unit A computing device originally dedicated to the generation of images to be rendered on a screen. In the modern computing era, GPUs are massively parallel computation engines capable of doing many kind of computations. 1

instruction set A group of instructions that are available - or not - in a micro-architecture. The instruction set with the mandatory instructions is the base instruction set, such as x86 or its 64 bits variant x86-64. Additional instruction sets such as x87, SSE or AVX provide additional instructions to expand the capability of a processor. 5

NEON The SIMD instruction set for the ARM v7 and ARM v8 architecture. It uses 128 bits wide registers. In ARM v7 NEON does not support double-precision operations, but it does in ARM V8. . 2

SSE The second SIMD instruction set for the x86 architecture. It uses 128 bits wide registers. The original SSE instruction set only support integer and single-precision floating point operations. SSE2 introduced double-precision floating-point operations. Both SSE and SSE2 are required in x86-64 (64 bits) processors. 1, 2, 5

SSE2 First extension to the SSE instruction set, adding double-precision floating point operations. 5

x87 It is the common abbreviation for the first, now obsolete, floating-point extension to the x86 architecture. See e.g. [18] for details. 5

REFERENCES

[1] R. Dolbeau, "Theoretical Peak FLOPS per instruction set on modern Intel CPUs," 2015. [Online]. Available: http://www.dolbeau.name/dolbeau/publications/peak.pdf

[2] M. Butler, L. Barnes, D. D. Sarma, and B. Gelinas, "Bulldozer: An approach to multithreaded compute performance," IEEE Micro, vol. 31, no. 2, pp. 6 – 15, 2011.

[3] M. Butler, "Bulldozer: a new approach to multi-threaded compute performance," in Hot Chips 22 Symposium (HCS), 2010 IEEE. IEEE, 2010, pp. 1–17.

[4] R. Dolbeau and A. Seznec, "CASH: Revisiting hardware sharing in single-chip parallel processor," Journal of Instruction-Level Parallelism, vol. 6, pp. 1–16, 2004.

[5] R. Kumar, N. P. Jouppi, and D. M. Tullsen, "Conjoined-core chip multiprocessing," in Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2004, pp. 195–206.

[6] A. Shayesteh, "Factored multi-core architectures," Ph.D. dissertation, University of California Los Angeles, 2006.

[7] A. Fog, "Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA cpus," Copenhagen University College of Engineering, 1996 – 2014. [Online]. Available: http://www.agner.org/optimize/instruction_tables.pdf

[8] Intel®, "Intel® Xeon Phi™ Processor Software Optimization Guide," no. 334541-001, June 2016. [Online]. Available: https://software.intel.com/sites/default/files/managed/11/56/intel-xeon-phi-processor-software-optimization-guide.pdf

[9] ARM®, "Cortex-A57 processor." [Online]. Available: https://www.arm.com/products/processors/cortex-a/cortex-a57-processor.php

[10] R. Grisenthwaite, "Armv8 technology preview," in IEEE Conference, 2011.

[11] A. Y. G. Singh, G. Favor, and A. Yeung, "AppliedMicro X-Gene 2," in HotChips, 2014.

[12] N. Stephens, "Technology update: The scalable vector extension (sve) for the armv8-a architecture," 2016. [Online]. Available: https://community.arm.com/groups/processors/blog/2016/08/22/technology-update-the-scalable-vector-extension-sve-for-the-armv8-a-archit

[13] NVidia, "CUDA C Programming Guide." [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[14] ——, "CUDA C Programming Guide: 5.4.1. Arithmetic Instructions." [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#arithmetic-instructions

[15] ——, "CUDA GPUs." [Online]. Available: https://developer.nvidia.com/cuda-gpus

[16] D. M. Muñoz, D. F. Sanchez, C. H. Llanos, and M. Ayala-Rincón, "Tradeoff of FPGA design of a floating-point library for arithmetic operators," Journal of Integrated Circuits and Systems, vol. 5, no. 1, pp. 42–52, 2010.

[17] B. Lee and N. Burgess, "Parameterisable floating-point operations on FPGA," in Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on, vol. 2. IEEE, 2002, pp. 1064–1068.

[18] Wikipedia. x87. [Online]. Available: https://en.wikipedia.org/wiki/X87