Many Core Hardware Architectures

Whitepaper

Randolf Rotta, rottaran@b-tu.de
Vladimir Nikolov, vladimir.nikolov@uni-ulm.de
Lutz Schubert, lutz.schubert@uni-ulm.de

19th April 2016

Contents

1 Overview 1

2 Driving Factors 2

3 Example Architectures 5

4 Details of the Intel Knights Corner 7

5 Trends and Predictions 10

1 Overview

Probably never ending is the desire for higher compute throughput in order to be able to solve new and larger problems and to perform more complex tasks interactively. Unfortunately, computation needs energy and energy is a limited resource. Mobile devices have only a limited energy reserve and high performance computing centers have only a limited power supply that reached its capacity a long time ago [KSS08].

The limited availability of electrical energy and power forces the evolution of computing architectures towards higher energy efficiency. Specialization allows to exploit energy-saving opportunities of specific application scenarios. One instance are many core architectures for high throughput computing. These trade sequential performance, fine grained interactivity, and real-time guarantees in favor of a overall higher average throughput per watt. For example, execution pipelines increase the best-case throughput but reduce the timing predictability and increase the interrupt overhead.

2 Driving Factors

Since 1965, the consequences of Moore's Law [Moo65] promise a doubling of the transistor count per processor every two years. His prediction is based on an assumed technology down-scaling of processor components in combination with the economically most profitable balance between the system size, the market size, and the necessary investments. Technology down-scaling also provided a reduction of the necessary supply voltage and parasitic capacitances, and an increase of the clock frequency, known as Dennard scaling [DGR+74]. In combination, this provided a doubling of compute throughput every two years.

Unfortunately, at some small enough technology scale, the power efficiency lost through leakage currents is as high as the power efficiency gained through Dennard scaling. Beyond that point, the threshold and supply voltage cannot be decreased much. This leads to a quadratic increase of the power consumption along the technology scaling. Experiments indicate that the 90nm CMOS technology already reached this limit [Tay12]. A few escape routes are near-threshold computing, reducing the clock frequency to compensate the loss of power efficiency [ZDB+07], leakage-reduced transistor designs like Tri-Gate and FinFETs, and considering dark-silicon architectures [Tay12].

A significant challenge is to translate the large number of available transistors into useful compute throughput. The basic functionality of a sequential compute core can be realized with a very small number of transistors. The remaining transistors and chip area can be invested into memory, caches, logic that speeds up the sequential compute core, special purpose cores, or simply into more compute cores. Pollack's rule [Pol99] estimates that the single-thread throughput of a core grows just approximately with the square root of invested transistors.[1] In comparison, investing these transistors into more cores would provide a proportional performance grow. Hence, in a fixed transistor and power budget, many small and simple cores offer an overall higher compute throughput compared to few large cores with high single-thread performance. Borkar [Bor99, BC11] discusses such tradeoffs in more detail.

[1]Motivated by the effects of signal delays in any 2-dimensional layout of the components.

In conclusion, well-parallelizable computations can, in theory, be carried out with higher energy efficiency on many core processors. In practice, parallel computation is difficult and has its own scaling limits. Flynn [Fly72] studied the relative performance potentials of different parallel processing approaches. The next paragraphs quickly review these with respect to the practically attainable speedup. Within a fixed component budged, higher speedup translates directly into higher energy efficiency.

Pipelining the processing of several instructions (*SISD*) could provide a linear speedup with the number of pipeline stages. However, conditional jumps and dynamic data dependencies introduce so called pipeline bubbles, which limit the actual speedup similar to Amdahl's law [Amd67]

Another approach is to apply the same instruction onto multiple data streams in parallel (*SIMD*). This promises a linear speedup and generally better energy efficiency because the instructions are fetched and decoded just once per vector operation. But again, conditional jumps and dynamic dependencies can slow it down to just logarithmic speedup. Thus, pure SIMD architectures cannot provide good energy efficiency for general purpose use.

Speedup can also be achieved by applying multiple instructions in parallel to the same data stream (*MISD*). While popular in earlier computer systems like Colossus Mark II, such systolic arrays eke out a niche existence in high-speed network traffic processors nowadays. To some extend, reusing locally cached data resembles MISD. Its importance will grow because data movements make a significant portion of the energy use in modern processors [BC11].

Finally, many core processors can process multiple data streams independently in parallel (*MIMD*). While providing the most flexible model, the actual speedup depends highly on the application. For fixed-size problems, the attainable speedup is limited through Amdahl's law [Amd67]. When increasing the problem size is possible, the speedup is limited by the memory that the application needs per core [Gus88, HW10]. Shared resource multiprocessors increase the utilization of internal logic despite pipeline bubbles. This yields better energy efficiency because unused logic components still consume energy due to leakage currents. One example is the interleaved execution of multiple threads inside a core.

The two main bottlenecks of a processor are its *communication bandwidth* and the *peak compute throughput*. The peak compute throughput is limited by the number of available scalar arithmetic units and their clock speed. The bandwidth of internal networks, to external networks and to the memory is limited by the energy efficiency of the network links [BC11]. Communication consumes power proportional to the distance and the bandwidth. Thus, within a fixed power budged, high bandwidth

Table 2.1: Example Many-Core Processors.

	CUs	HTs/CUs	SUs/CUs	SU freq (GHz)	∑ dp units	∑ HTs
Intel Haswell	2–4	2	4int+2x256bit	2.0–3.9	16–32	8–16
AMD Bulldozer 6386	16	2	2x128bit	2.8–3.5	64	32
Sparc T5	16	8	2int+1dp	3.6	16	128
IBM Cyclops64	80	2	1	0.5	80	160
Godson-T	64	1	2x256bit?	1.0	512?	64
Intel Larrabee	24–48	4	512bit	1.0	192–384	96–192
Intel KNF	32	4	512bit	1.2	256	128
Intel KNC	60–62	4	512bit	1.2	480–469	240–248
Intel KNL	72	4	2x512bit	1.00–1.5	1152	288
Nvidia Fermi GF104	16	48	32sp=16dp	1.5	256	768
Nvidia Kepler GK110	13–15	64	192sp+64dp	0.732	832–960	832–960
Kalray MPPA 256	256	1	1	0.4	256	256
Tilera TilePro64	64	1	integer	0.7–0.86	-	64
Tilera TileGx72	72	1	3dp?	1.2-1.5	216?	72
Adapteva Epiphany-III	16	1	2sp	1.0	-	16
Adapteva Epiphany-IV	64	1	2sp	0.8	-	64
Azul Vega-3	54	1	?	'slow'	?	54
Cavium Octeon III	1–48	1	?	2.5	?	1–48
Infineon Aurix	3	1	DSP+1sp	0.3	3	3

communication is possible only over short distances and long-range communication is restricted to low bandwidth. The consequence is, that in order to gain high compute throughput most instructions have to operate on local registers and memory, and reuse locally cached data as much as possible.

3 Example Architectures

Instead of diving into concrete processor architectures, this section introduces a simplified architecture model. It focuses on key design aspects that are relevant for many core processors and abstracts over too technical details. Figure 3.1 shows a visual representation of the model.

Scalar units (SU) perform floating point and/or integer arithmetic operations on scalar values. They are called ’cores’ by Nvidia and ’processing elements, by OpenCL. Several scalar units can be combined into vector units that perform the same operation in parallel on several values akin to Flynn’s SIMD architecture. However, configurable short vector units support operations on different scalar sizes, so that a 512bit unit is, for example, equivalent to 16 single precision or 8 double precision scalar units. The same applies for integer arithmetic with 1,2,4, and 8 byte granularity.

In order to tell the scalar units what to do, an instruction stream has to be interpreted. This is done by a *hardware thread* (HT). Each hardware thread has access to at least one scalar unit and represents one physical control flow. A *compute unit* (CU) is a group of hardware threads that share a set of scalar and vector units. They are called ’cores’ by Intel, ’modules’ by AMD, and sometimes they are simply called ’processor’. The time-sharing of scalar units between threads equals Flynn’s MIMD resource sharing. The compute unit may implement pipelining that combines the operations of the individual threads.

Finally, *protection domains* (PD) restrict which part of the memory and communication space can be accessed by the hardware threads. Protection domains might

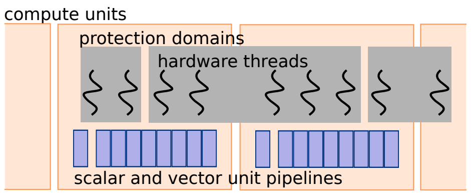

Figure 3.1: Simplified Many-Core Model.

be hardwired or configurable. For example, older GPU processors had only a single protection domain that was shared by all threads. Intel processors, including the XeonPhi series, have configurable protection domains per threads. However, sharing a single domain between all threads of a compute unit can be more efficient.

The hardware threads have access to *memory units* of one or several types. All of them are accessible from every thread, constrained only by the protection domains. However, the access method, overhead and latency may vary, which is known as Non-Uniform Memory Access (NUMA) architecture. In order to simplify the model, each type of memory partitions the threads into groups that share the same nearest memory unit. Some processor architectures like x86 multi-cores have just a single type of memory while others like GPUs come with a whole hierarchy of private, local, and global memory.

The access to the memory are usually routed through a *hierarchy of caches*. This helps to reduce the far-distance traffic by exploiting temporal and spatial locality in access patterns. Typically, this induces a hierarchical partitioning of the threads. For example, all threads of a compute unit share the same level-1 cache, groups of compute units share a level-2 cache, and so on. Caches of the same level might be grouped together into a distributed cache, where the actually responsible cache depends on the accessed address. So called, directories do not cache data but just information where that data is cached in upper cache levels.

Table 2.1 summarizes key properties of several existing and upcoming many core architectures. The first three columns show the number of compute units, hardware threads per compute unit, and scalar or vector units per compute unit.

For a naïve comparison, the ’dp units’ column gives the total number of double precision floating point (64bit) scalar units in the processor. Note that the actual scalar and vector units are quite different and difficult to compare because their usefulness depends on the target applications. Compute units with out of order execution are even more difficult to compare because they use a collection of specialized scalar and vector units.

The *computational intensity* is the ratio of useful compute operations per byte of transferred data. Many numerical HPC applications have a high floating point compute intensity whereas network packet analysis and similar applications are use mostly integer operations and have much more irregular control flows. Therefore, with a fixed communication bandwidth, HPC applications benefit from more floating point units while network applications benefit from more threads instead.

4 Details of the Intel Knights Corner

The Knights Corner (KNC) processor is available as PCIe card with on-board memory. The cards need a host computer to power up the card, load the operating system image into the card's memory, and then notifying the card's boot loader to process the image. All communication between KNC and host is handled over PCIe shared memory access and message signaled interrupts. Linux kernel modules and tools for the host are available that allow to load an operating system and to read and write to the cards memory. On top of this, the higher level SCIF kernel modules provide virtual communication channels, but need an appropriate back-end in the card's operating system.

The cores (compute units) have an in-order super-scalar architecture with two parallel pipelines that was derived from the P54C processor. The cores were extended by x86-64 support, a 512-bit vector processing unit, and four interleaved hardware threads. By design of the instruction decoder, two active threads are necessary to fully utilize the core and more are needed to mask memory access latency and other sources of pipeline bubbles.

Inside each core are two separate first level caches for instructions and data. The first level caches communicate to the core-local unified second level cache and a ring network connects these with each other. The cache coherency is implemented through a distributed third level directory. In order to avoid bottlenecks, the addresses are scattered over all parts of the directory with cache line granularity. Due to this topology, access to the local second level cache is fast, but cache misses suffer from a distance-dependent very irregular latency.

Of the 64bit logical address space, only 48bit are actually usable. These are mapped with a 4 level page table structure onto a 40bit physical address space with

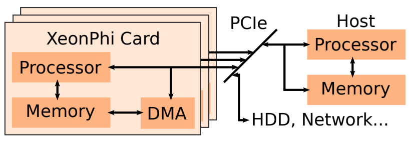

Figure 4.1: XeonPhi extension cards controlled by a host processor.

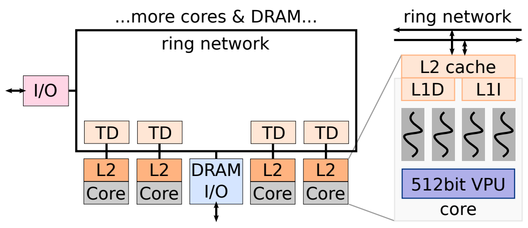

Figure 4.2: XeonPhi Knights Corner processor.

pages of 4KB and 2MB granularity. The 1GB page extension is not supported by the KNC. The physical address space contains the on-board memory, memory mapped IO registers, and direct memory access to other PCIe devices and the host memory. The latter can be configured through an system address translation table (SMPT) with 32 pages of 16GB size.

In order to speed up the address translation, each core has a Translation Lookaside Buffer (TLB) that is shared by all four threads. It can store translations for 4KB and 2MB pages and has an internal two-leveled TLB cache. The TLB of the Knights Corner processor can share entries between threads only if they use the same translation tables. In conclusion, TLB misses can be reduced by preferring 2MB pages and sharing a single logical address space between all 4 hardware threads. Sharing parts of the translation tables between cores reduces the needed cache space.

The underlying AMD64 architecture still has legacy segment registers, but their base and limit configuration is fixed to cover the complete logical address space. Just the base address of the segment registers FS and GS can be modified. With several threads in a single address space, these registers can be used for efficient memory access to thread-local data. Their advantage is that they do not waste general purpose registers and should be faster than manual address calculations from offset and base.

Unfortunately, none of the usual timer hardware (PIT, CMOS RTC, ACPI timer, HPET) is present except for the cores' local LAPIC timers and time stamp counters (TSC). A special IO register holds the core's clock frequency, which allows to calibrate the timer.

The IO registers can be accessed only through memory mapped IO (MMIO), which means that the registers reside at certain physical addresses and can be accessed by normal read/write instructions. To make them accessible, they need to be mapped into the logical address space first.

The mechanisms for synchronization between threads are surprisingly lacking.

The helpful monitor/mwait instructions and transactional synchronization extensions are missing. Thus, just the halt instructions and inter-thread interrupts can be used to put threads into sleep and wake them up later. A 64bit compare-and-swap instruction is available and a delay instruction was added that allows a thread to skip a configurable number of cycles. Some documents mention "the introduction of thread wakeup/sleep mechanisms through microcode and hardware support", which leaves some hope.

A special peculiarity of the KNC is its thermal management as most of it was moved from hardware into the operating system software. The core clock is divided down automatically during overheating (thermal events). However, this is inefficient and might be insufficient. The operating system has to lower the core frequency and voltage manually or speed up the fans if available. For this purpose, power management interrupts signal thermal events and the completion of frequency/voltage transitions.

5 Trends and Predictions

How will the many core processor landscape evolve until the completion of the MyThOS project? Moore's law estimates roughly a doubled complexity every two years. Assuming a 4 year time frame for the project, a quadrupled complexity can be expected. This increase in transistor count will have to be split across the number of compute units (cores), the number of scalar units, the bit-width of the vector units, and the number of hardware threads per compute unit. Simply increasing the number of compute units might not hit the most optimal spot.

The target market of a processor dictates its power/energy budged and the set of typical applications. The algorithms used in these applications cover a certain range of computational intensities that can be averaged into a target computational intensity. This balance point between communication bandwidth and attainable compute throughput determines roughly how much of the power budget is spent on communication versus scalar units for computation. Hence, the number of scalar units per processor is more or less fixed by the target market and the relative energy efficiency of the available technologies.

With respect to many core processors targeted by MyThOS, let's assume that the market does not shift away from HPC applications and that all basic components see similar gains in energy efficiency. Then, the number of scalar units for HPC processors might quadruple during the time of the project from the 470 of the Intel KNC to 1800–2000 on future processors. Nvidia's existing Kepler processor with 960 dp units and Intel's upcoming KNL with probably 1152 dp units point into this direction.

The quadrupled scalar units could be organized simply into larger vector units while the number of compute units stays fixed. The past evolution from 128 to 512bits is an example for this approach. However, the within-vector dependencies increase with larger vectors and require special attention, the instruction sets becomes more complex with larger vectors, and not all parts of an application can be parallelized equally well. Thus, vector units wider than 1024 or 2048bits seem unlikely.

Using multiple vector units of a smaller size inside a compute unit is a viable alternative. Increasing the number of vector units in a core with out-of-order execution is doable because the control flow analysis can pick independent vector operations for concurrent execution. With in-order execution, however, their number is limited by the number of parallel pipelines. For example, the Intel KNL is expected to have dual-pipeline superscalar cores and, thus, can operate at most two independent vector units. Nvidia's stream processors organize the vector units into multiple internal pipelines and, instead of out-of-order execution within threads, they pick operations from many threads. In-order execution is more energy efficient than out-of-order execution and independent pipelines could as well be split into independent cores. In conclusion, the number of scalar/vector units per compute unit will stay very low. Another possibility would be the inclusion of more units than can be operated in parallel akin to the dark silicon approach. Such units could support special purposes like transcendental functions. However, this aspect is more a question of how the vector units are counted.

With two vector units per compute unit, the projected 2000 scalar units can be operated by 62 cores with 1024bit vectors or by 124 cores with the existing 512bit vectors.

Finally, the number of hardware threads depends on the applications' memory access patterns, memory latency, and other architecture details. A minimal number of active threads per compute unit is needed in order to keep all available scalar units busy, that is to reach the peak compute throughput. This is determined by the pipeline length and the instruction scheduler of the compute unit. For example, the Intel KNC needs at least two and Nvidia Kepler GPUs at least four active threads in each clock cycle. In order to saturate the memory/communication bandwidth, enough outstanding requests need to be generated. A thread executed in an in-order dual-pipeline can issue just one or two memory requests and, then, has to wait until these are served, which depends on the memory access latency. Thus for a given peak bandwidth, the number of necessary threads can be calculated from the latency, the cache line width, and the number of outstanding requests per thread.

Some applications may benefit from many threads, but in general the coordination overhead grows with the number of threads and, thus, less required threads is often better. This can be achieved, for example, by increasing the cache line and vector width. But wider cache lines reduce the chances for spatial reuse, decreasing the efficiency. The number of outstanding requests can be increased, for example, with cache prefetchers and vector scatter/gather operations for applications with predictable access patterns. Both are already present. Finally, the latency can be reduced by larger caches, more caching levels, and a faster network. But only applications with high temporal cache reuse benefit from more and larger caches, which leaves the network. For example, the Intel KNL is expected to use a faster

2D mesh network instead of the previous ring network. This should result in a higher bandwidth and shorter distances between compute units and memory. The latter, however, might be compensated by the larger number of compute units. In conclusion, it seems likely that the number of threads per compute unit stays similar, while the number of compute units increases.

Improvements beyond the sole advances of the hardware's technology scaling will require progress on the side of application software and runtime environments. Amdahl's Law warns that the hardware gains from more threads and wider vectors can be canceled entirely due to insufficiently parallelized software.

Application software can aim towards improved spatial and temporal reuse of cached data including sharing of cached data between nearby compute units in order to reduce the far-distance bandwidth utilization. They should also increase the use of vector instead of scalar operations for data processing in order to increase the throughput per thread and the number of outstanding memory requests.

Finally, operating systems and runtime environments can help, for example, by following means: Focus on localized coordination and communication in order to keep synchronization latency low, for example through support for hierarchical organization mechanisms. Provide low overhead and low latency offloading primitives that enable the runtime environment and the application to effectively parallelize smaller tasks in order to make use of the many threads without the need for too fine grained data decomposition. Use offloading to parallelize management tasks in order to reduce waiting and interruption times for the application threads. And enable the effective use of nearby memory to keep memory access latency low.

Bibliography

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[BC11] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011.

[Bor99] Shekhar Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999.

[DGR+74] R.H. Dennard, F.H. Gaensslen, VL Rideout, E. Bassous, and AR LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.

[Fly72] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[Gus88] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, May 1988.

[HW10] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2010.

[KSS08] S. Kamil, J. Shalf, and E. Strohmaier. Power efficiency in high performance computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8. IEEE, 2008.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, pages 114–117, April 1965.

[Pol99] Fred J. Pollack. New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)(abstract only).

In *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 32, pages 2–, Washington, DC, USA, 1999. IEEE Computer Society.

[Tay12] Michael B. Taylor. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 1131–1136, New York, NY, USA, 2012. ACM.

[ZDB$^+$07] Bo Zhai, Ronald G. Dreslinski, David Blaauw, Trevor Mudge, and Dennis Sylvester. Energy efficient near-threshold chip multi-processing. In *Proceedings of the 2007 international symposium on Low power electronics and design*, ISLPED '07, pages 32–37, New York, NY, USA, 2007. ACM.