2016

GPUs, Coprocessors, ASICs and FPGAs

SOMMERAKADEMIE LEYSIN 2016, GREEN COMPUTING

MAXIMILIAN SCHÄFFELER

GPUs, Coprocessors, ASICs and FPGAs

1. Problems of homogeneous systems in HPC

Historically performance of supercomputers has grown by just scaling up the number of processor cores and their clock frequency (Hartenstein, 2012, p. 475-476). This way supercomputers could achieve ever higher performance, this trend did last up to today's petascale supercomputers. In recent years however, performance gains did slow especially in HPC (High Performance Computing). The old approach of scaling up the number of transistors seems to have reached a limit, but a lot of potential for future development can be seen in heterogeneous architectures. In contrast to the usual homogeneous systems heterogeneous systems combine a variety of different architectures and approaches to computation.

To get a sense of why heterogeneous architectures are beneficial or even necessary in HPC, we first have to take a look at how a normal CPU executes a given instruction. This can be illustrated with the example of the following Intel Assembler operation:

ADD EAX, EBX

This instruction will fetch the value stored in the register EBX, add it onto the current value of EAX, and then store the result back in EAX. For this to happen the CPU has to go through several steps:

1. First we have to get the instruction, this phase is called *FETCH instruction*. The CPU will try to get data at the address of the instruction pointer from the main memory/caches. This data contains the whole instruction including operands in binary.

2. Second the CPU has to understand which operation it's supposed to execute; therefore, it has to decode the instruction. This phase has become more and more complicated over time, as new opcode formats were introduced and backwards compatibility to older processors had to be maintained. Especially CISC (Complex Instruction Set Computer) architectures suffer from this, as they contain a big number of different instructions.

3. After knowing what instruction to execute and which operands the instruction will be executed on, the CPU will get the actual values of the operands. This may take different amounts of time depending on whether the operands are stored in the registers, the cache or the main memory. Some instructions require no additional operands.

4. As the last step the CPU executes the operation inside the ALU (Arithmetic Logic Unit) and will store the results back in memory. (Wikipedia, 2016)

Interestingly the actual computation takes a comparably low amount of time in the instruction cycle, this is especially important, as the rest is overhead we get for each data item and operation. Our CPU always performs a single operation on a single data item (SISD). With modern processors this is not entirely true anymore, CPUs have adopted techniques like SSE and AVX (extensions for x86 processors), which allow a single floating point operation to be performed on multiple data items at once, yet CPUs mainly are SISD (or MIMD, if multi-core) processors (Chiesi, 2014, p. 7-8).

2. Heterogeneous architectures

There are a lot of problems concerning computing with general purpose CPUs, which have to be identified, so alternative concepts of processors can be chosen with these problems in mind. Those problems are mainly present in HPC and do not impact personal computing as much, because of the lower amount of processing power that is generally needed.

The concept of a general purpose CPU comes at the cost of a lot of control overhead compared to specialized architectures. Only 5% of transistors in a general purpose CPU make up the ALU, the

remaining 95% are needed for decoding/fetching instructions, organizing memory access and other control tasks (Figure 1) (Hartenstein, 2012, p. 503). CPUs trade in performance for the ability to solve a wide variety of tasks, which is actually the main reason they could become so popular in personal computing.

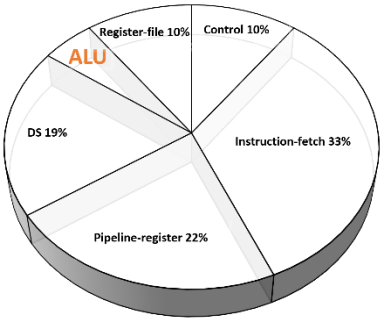

Figure 1

In the context of green computing, the central topic of our seminar at the Sommerakademie in Leysin, it is important to note that CPUs do consume a high amount of power. The power demand is generally around 100 Watts for high performance CPUs like Intel Core i7 processors. Supply costs are growing due to rising electricity prices and often do surpass the acquisition cost of a data center or supercomputer, costs are often above 1000$ a day. Another reason for increased energy consumption is the increased heat output, as clock speeds have risen quickly in the past and architectures did not get more energy efficient to compensate the higher electricity demand.

CPUs offer much less parallelism in computing when compared to accelerators like GPUs. Additionally, they are also difficult to program in a parallel way which requires skilled software developers that know how to program and manage multiple threads in an efficient way. There is a shortage of programmers that have this knowledge, it is often referred to as the "Programming wall" (Hartenstein, 2012, p. 476). This lack of parallelism weighs especially heavily in High Performance Computing, where often scientific problems are simulated or analyzed. Those applications perform a majority of their operations on multiple pieces of measured or calculated data at once. Processors which can provide this functionality in hardware are capable of doing SIMD (Single Instruction Multiple Data).

CPUs originally did not offer SIMD. Today a lot of CPUs provide support for vector operations albeit to a much lesser extent than GPUs. Intel's systems are called AVX (Advanced Vector Extensions) and SSE (Streaming SIMD Extensions), they allow you to execute a floating point operation on a number of elements at the same time. However, this is not sufficient for most problems in HPC. CPUs are built for being general purpose processing units. That means that they are very flexible and versatile but not specialized for any particular sort of problem. Their vector support simply cannot match dedicated accelerators.

2.1 Need for heterogeneous computing

In the past this limitation didn't impact the performance growth too much because Moore's Law has proven to be valid the last decades. Moore predicts that in an integrated circuit the number of transistors per area doubles every (2^{nd}) year, this results in an exponential growth which can be seen in Figure 2 (Moore, 1965). Additionally, the clock speed of processors increased dramatically since 1970. Today, the clock speed already reached its limits at around four GHz or is even reduced again by a lot to save power. Limiting factors are the increasing energy consumption at higher clock speeds as well as the higher heat dissipation, which can eventually lead to hardware damage (Chiesi, 2014, p. 4).

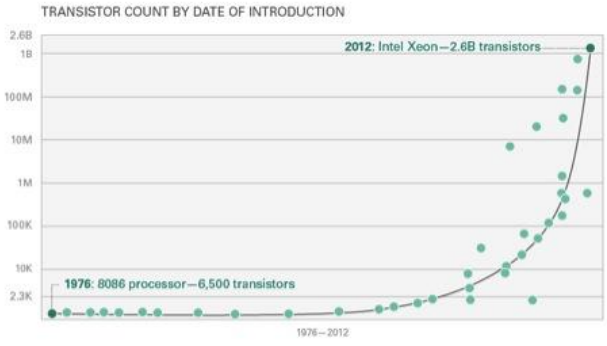

Figure 2

Meanwhile even the transistor density and therefore Moore's law are approaching physical limitations. These include for example leakage currents, which cause electrons to flow from the source to the drain of a transistor without any voltage applied at the gate, this can happen due to the small distances and quantum effects. In the end leaking currents lead to incorrect, unpredictable faults in calculations. (Hu, 2010, p. 266)
Another problem is that by doubling the number of transistors on a chip performance increases by only 40% according to Pollack's Rule because of higher control overhead inside the die. (Shekhar Borkar, 2011, p. 67-77) One possible solution to the problem is to take 2 processor cores with half the transistor density of the one discussed before. This will lead to a performance increase of 70 to 80%, significantly higher than increasing the transistor density by a factor of two. Increasing the cores of the CPU, however does not scale as well for larger numbers of processors. Amdahl's law shows that with code that has a portion of 50% that can be executed in parallel, more than 16 cores do not improve performance anymore (Chiesi, 2014, p. 6). Possible problems when parallelizing code include data dependencies which make it impossible to calculate operations independently. The development of parallelizable problems is a difficult skill to master and requires additional resources and skills which makes software development more expensive (Hartenstein, 2012, p. 476, 477).

2.2 Structure of a heterogeneous system

There are several possible solutions to increase computing performance that are promising. A very obvious way might be to introduce additional processor cores to increase parallelism. This approach is used in almost all of todays' processors and can provide a pretty significant increase in performance. When going from one processor core to two and by that doubling the number of transistors and energy consumption a speedup of up to 80% can be achieved. However, this is highly dependent on the software which needs to be optimized. In practice due to constraints when optimizing software, limits can be reached with even a low number of cores.

Another approach would be to add specialized hardware to support the general purpose CPU with certain tasks. Those extensions for processors are called co-processors, they are more or less tied to a specific processor and offer e.g. floating point acceleration or motion tracking. The third class of processors are hardware-accelerators, specifically ASICs and FPGAs. They offer highly specialized hardware that has the highest factors of improvement.

3. Methods of hardware acceleration in HPC

This sections will present different components of heterogeneous systems in greater detail. For each type of accelerators their benefits and drawbacks will be evaluated and possible applications will be given.

3.1 Coprocessor

Coprocessors are generally defined as specialized processors that assist the main processor in certain tasks where they do provide hardware acceleration. (Britannica, 2016) They are mostly tied to a specific brand or product line with which they will work and can therefore be highly optimized.

The history of co-processors goes back to the 1970s. The Intel 80386 was an early processor of Intel. It originally shipped without hardware support for floating point operations which therefore had to be simulated in software. To achieve faster floating point performance Intel later sold the 80387, called FPU ("floating point unit") as a floating point unit. It provided hardware support for single and double precision floating point operations. The 80387 was one of the first coprocessors designed. Today floating point units are integrated into the CPU but coprocessors are still important to support other tasks. (Wikipedia, 2016)

The dominant coprocessors in modern computing are GPUs (Graphics processing unit) and sound cards. GPUs will be covered separately in section 3.2. Even Smartphones usually have coprocessors that process live sensor data, e.g. the Apple M7 or Qualcomm Snapdragon Sensor Core. They use low power and so the CPU can idle longer which saves precious battery power.

3.1.1 Intel Xeon Phi

The Xeon Phi is a product line by Intel. It consists of coprocessors for the Intel Xeon brand of processors. These processors are specialized on accelerating vector/matrix computations, as they offer a highly parallel architecture which follows the principle of SIMD. Under ideal conditions the generation *Knights Corner* from 2012 can achieve up to 1.2 TFlop/s (trillion floating point operations) per second (Wettig, 2015, p. 3). This tops the regular Xeon processors by a factor of five. The current generation can achieve up to 3 TFlop/s (Intel, 2016).

To achieve such high performance, the Intel Xeon Phi has to have a special architecture. The following description is based on the processor generation Knights Corner, which was used to build the QPACE 2 supercomputer. It consists of up to 72 processor cores, however due to production difficulties Intel was only able to deliver less cores, in the QPACE 2 build each processor has got 61 cores (Wettig, 2015, p. 3). This leads to problems when writing programs for the Xeon Phi, especially if the number of cores is a prime number. The tasks cannot be split up and parallelized that easily compared to even numbers or even powers of two. Yet in comparison to regular CPUs that mostly offer between two and sixteen cores the Xeon Phi offers high potential for parallelization. Each processor core is clocked at 1.238 GHz. The processor is accompanied by 16GB DDR5 memory on chip with a bandwidth of 352 GB/s. (Wettig, 2015, p. 3-4)

Power consumption of the Xeon Phi depends on the exact model but is generally around 300W which is relatively high when compared to less expensive offerings of NVIDIA and AMD that offer similar performance. This combined with an older processor generation used a high acquisition cost of the processor paired with the challenges in developing software contributed to the rather small success of the Xeon Phi. Intel is still investing in future generations of the Xeon Phi, namely Knights Hill and Knights Landing (Intel, 2015).

3.2 Graphics processing unit

GPUs (Graphical processing units) are one of the best known means of heterogeneous computing because they have become a part of almost every computer from smartphones to workstations. Originally CPUs were developed to accelerate computing intensive 3D applications but today they have many diverse applications especially in HPC (Chiesi, 2014, p. 13-14). This is due to their pretty general purpose architecture that allows for a lot of common science problems to be executed on GPUs.

GPUs are connected to the CPU via PCI express mostly which similarly to the Xeon Phi marks a performance bottleneck due to the limited bandwidth and the large amounts of data that need to be transferred between the processor and GPU (Chiesi, 2014, p. 16).

Graphics processing units work highly parallel according to the SIMD principle to perform a big number of matrix transformations at once which are usually needed in 3D applications and data analysis. They have multiple layers of parallelism, which help programmers organize threads that will be executed simultaneously.

Floating point operations benefit most from SIMD on GPUs. GPUs from the manufacturer NVIDIA can be programmed via the NVIDIA CUDA (Compute Unified Device Architecture)-API that provides direct access to the hardware of the graphics card but at the same time assists with splitting up tasks into different parallel threads. The highest layer of threads is a Grid that contains multiple thread blocks.

The thread blocks themselves contain Warps that are assigned to individual Streaming Multi-Processors and that contain the actual threads. The multiple layers of abstractions simplify development for GPUs (Chiesi, 2014, p. 17).

3.2.1 Architecture

The NIVIDIA Fermi (GeForce 500 series) architecture was introduced in late 2010. It was the successor to the GeForce 400 series and used in NVIDIA high-end graphics cards like the GTX580.

The Fermi graphics chips have dedicated DRAM and a 768 KB L2 cache. It also provides an interface to connect to the host, specifically a CPU. It consists of 16 Multi-Processors that again integrate 32 Streaming processors each. The streaming processors are capable of 521 single precision (256 double precision) instructions per clock cycle. Additionally, each multi-processor has 64 KB of L1 cache. Special function units provide hardware support for trigonometry, root calculation, etc. (Chiesi, 2014, p. 15-16).

NVIDIA Fermi Architecture

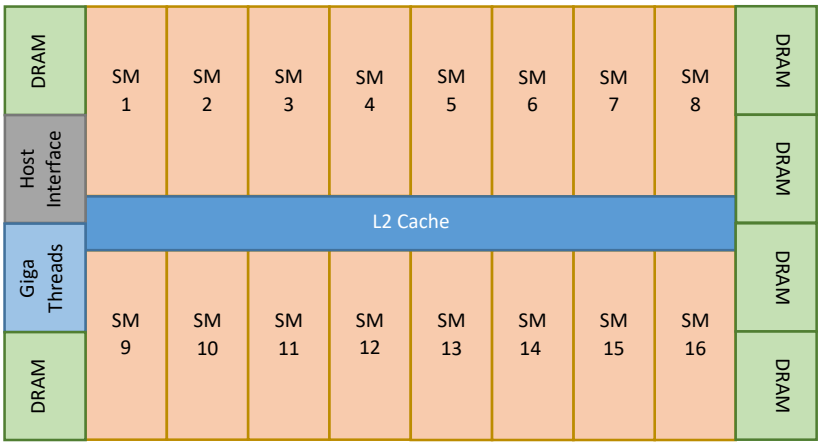

Figure 3

3.2.2 Benefits and drawbacks

Graphic cards use a lot of energy especially if compared to ASICs and FPGAs. Furthermore, the potential performance gains are several orders of magnitude smaller than using reconfigurable or hardwired accelerators, the performance bottleneck is clearly the connection to the CPU (Hartenstein, 2012, p. 507). Compared to CPUs however, their performance advantages (up to a factor of 3) outweigh the additional energy consumption. Other problems include the effort that is required to optimize programs for GPUs. The parallelization and organization of the data involved in the computations is complicated and difficult. This requires additional skilled software developers and development time or effort. On the other hand, programming and developing applications for FPGAs and ASICs is still a lot more time-consuming.

3.3 Application-specific integrated circuit

ASICs (Application specific integrated circuits) are hardware accelerators that include hard-wired logic that fixes their behavior before the point of fabrication. Their applications include small electronic devices like car components, home appliances or even bitcoin mining.

ASICs are very performant and offer quick processing of data, e.g. in airbags in cars where a quick reaction time is crucial. Because they are optimized for only a single task they are also very efficient and consume less energy than all other processor types. For example, ASICs are reported to be 500 times more power efficient for decoding the video codec H.264 than a CPU (Hartenstein, 2012, p. 489).

There are also negative aspects that are specific to using ASICs. The most important one is the difficult development. Development and design cycles are complicated, because a whole new architecture for the hardware has to be designed for each and every ASIC. This process is long and costly as it involves several iterations of design, prototyping and testing (Hartenstein, 2012, p. 489-490). The prototypes have to be manufactured which takes several months of development time, and small bugs lead to an entire new iteration so it's difficult to meet time-to-market demands.

This means that ASICs are not suited for products that are sold in low quantities only. However, when selling large quantities, like for example smartphones, products that target the mass market etc. ASICs can actually be more economically efficient than using a FPGAs for the task, as ASICs have far less hardware overhead and therefore use less resources (Hartenstein, 2012, p. 490).

Another downside of ASICs is their lack of versatility. They cannot be modified or updated after production. Bugs in the circuit that lead to malfunction cannot be patched, functionality cannot be upgraded when desired.

Those reasons combined led to a decline in market share which in return led to the rise of FPGAs. Their principle is far better suited for most niche or medium size applications.

3.4 Field-programmable gate array

The name FPGA (Field programmable gate array) stems from their capability to be programmed with customers "in the field". FPGAs are hardware accelerators that can be dynamically reconfigured to execute different tasks which is a huge benefit when compared to ASICs.

Sometimes a small ARM host chip is added to the FPGA that can accomplish more general purpose tasks as well as the initialization, boot-up, configuration, and I/O for the FPGA. EPROM is used to save the configuration even when no power is available.

The first FPGAs did appear on the market in 1984, made possible with the invention of programmable read-only memory. This did enable reconfiguration and permanent storage of the most recent configuration even when no power was available (Hartenstein, 2012, p. 492-493). The two most successful manufacturers of FPGAs were Altera and Xilinx; Altera has been acquired by Intel in 2015.

3.4.1 Applications

Most of the applications of FPGAs are small to medium scale. In those projects they offer low cost production because there are no custom parts needed and off-the-shelf components can be bought. This reduces development time and cost (Chiesi, 2014, p. 25).

Concrete applications include encryption and decryption, pattern matching. In general, applications with many memory accesses that are easy to parallelize are well suited for FPGAs. Those properties can often be found in scientific problems where thus huge improvements are possible. Speedup factors of up to 3 orders of magnitude are reported to be possible for certain applications (Hartenstein, 2012, p. 498), however in reality most performance gains are far lower. As of today, FPGAs are now used in 70% of all embedded devices and rising (Hartenstein, 2012, p. 490).

3.4.2 Configurable logic block

FPGAs use Configurable Logic Blocks (CLB) instead of logic gates. They can simulate any Boolean function and therefore gate with 4-6 variables/inputs (16 = $2^{(2^2)}$different configurations for 2 inputs). CLBs are lookup table based, that means they have all possible configurations of inputs and boolean functions stored with their result which maps to the output. The truth table of the function is represented by SRAM cells that make up the lookup table. The input to the gate then specifies which cell has to be read (Hartenstein, 2012, p. 493-494). Figure 4 visualizes a LUT with 2 inputs.

There are different sizes of logic blocks that vary from very fine-grained single transistors to coarse-grained function logic blocks like adders or memory cells. Fine-grained blocks reduce overhead in computation but increase the wiring effort that is needed to connect the many CLBs. This trade-off has been resolved by using medium size blocks like lookup tables. The wiring between gates can be can be configured after manufacturing as well (Hartenstein, 2012, p. 493-494).

#	g00	g01	g10	g11	f(A,B)
0	0	0	0	0	0
1	0	0	0	1	A and B
2	0	0	1	0	B disables A
3	0	0	1	1	A and B
4	0	1	0	0	A disables B
5	0	1	0	1	B
6	0	1	1	0	A exor B
7	0	1	1	1	A or B
8	1	0	0	0	not (A or B)
9	1	0	0	1	A coin B
10	1	0	1	0	no (B)
11	1	0	1	1	B implies A
12	1	1	0	0	not(A)
13	1	1	0	1	A implies B
14	1	1	1	0	not (A and B)
15	1	1	1	1	1

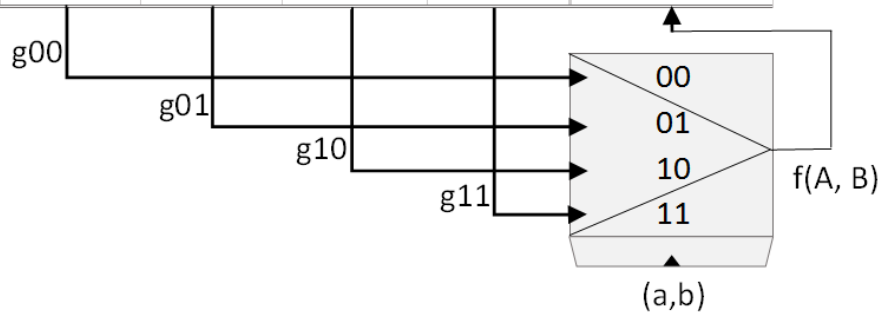

Figure 4

3.4.3 Reconfigurable computing

FPGAs are representative for the idea of reconfigurable computing which is defined as computing with circuits that can be configured using software and programming languages. The programming languages used differ a lot from imperative or functional programming languages that are used to write software and applications. The languages in use are called hardware description languages, two important ones are VHDL and Verilog (Hartenstein, 2012, p. 526-527).

There are two ways in which FPGAs have to be configured:

Configware describes the configuration of the reconfigurable computing device. This includes the Boolean function that the CLB computes which is represented by the lookup tables. In this part of the code the wiring between CLBs will also be adjusted properly.

Flowware controls the flow of data-streams through the FPGA, or defines the path that the data passes along (Hartenstein, 2012, p. 513).

Reconfiguration of Field Programmable Gate arrays is often possible even at runtime for parts of the circuit. This process takes mere milliseconds and allows for better optimization when needed (Hartenstein, 2012, p. 495).

On paper, FPGAs are clearly inferior to CPUs and ASICs in aspects like clock speed, size, power efficiency (compared to ASICs). However, according to the so-called "Reconfigurable Computing Paradox", they are superior to CPUs and ASICs in many applications (Hartenstein, 2012, p. 501-502).

4. Conclusion

If you compare the Von Neumann architecture and general purpose CPUs to modern coprocessors and accelerators, it becomes clear that accelerators will grow in importance in the future, especially in HPC. The term "von Neumann Syndrome" (Hartenstein, 2012, p. 494) sums up the problems of general purpose CPUs: massive control overhead, SISD/MIMD and few SIMD capabilities, inferior architecture. This lead to the "Reconfigurable Computing Paradox", FPGAs offer much higher performance for parallel and most scientific calculations.

There are however, a lot of challenges until FPGAs can develop from a niche product to a mass market product, that can challenge GPUs and CPUs in popularity. FPGAs are still more difficult to program than graphics cards because HDLs (hardware description languages) are used in development. Furthermore, knowledge of the underlying hardware is essential to optimize the Configware and Flowware.

The programming wall poses another problem. There are too few programmers that have knowledge and experience in writing Configware and Flowware. This problem has to be tackled in education, therefore investments in the future are needed. FPGAs won't eliminate the need for general purpose CPUs but they can supplement the shortcomings of the von Neumann architecture.

Bibliography

Britannica. (2016, 09 20). Retrieved from Coprocessor: https://www.britannica.com/technology/coprocessor

Chiesi, M. (2014). *Heterogeneous Multi-core Architectures for High Performance Computing.* University of Bologna.

Hartenstein, R. (2012). The paramountcy of reconfigurable computing. In *Energy-efficient distributed computing systems.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Hu, C. C. (2010). MOSFETs in ICs—Scaling, Leakage, and Other Topics. In *Modern Semiconductor Devices for Integrated Circuits.*

Intel. (2015). Retrieved from Intel® Xeon® and Intel® Xeon Phi™ Roadmap Update : http://www.inteldevconference.com/wp-content/uploads/2015/03/1-Roadmap.pdf

Intel. (2016, 09 15). Retrieved from Intel® Xeon Phi™ Product Family: http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

Moore, G. E. (1965, April 19). Cramming more components onto integrated circuits. *Electronics*.

Shekhar Borkar, A. A. (2011). The Future of Microprocessors. *Communications of the ACM, Vol. 54 No. 5*.

Wettig, T. (2015). *QPACE 2 and Domain Decomposition on the Intel Xeon Phi.*

Wikipedia. (2016, 09 19). Retrieved from Intel 8087: https://en.wikipedia.org/wiki/Intel_8087

Wikipedia. (2016, 09 22). Retrieved from Instruction Cycle: https://en.wikipedia.org/wiki/Instruction_cycle