An Exploration of Heterogeneous Systems

Jesús Carabaño[1,†], Francisco Dios[1,†], Masoud Daneshtalab[2], Masoumeh Ebrahimi[2]
[1]Åbo Akademi, Finland , [2]University of Turku, Finland
{jescar,fmdibu}@abo.fi, {masdan,masebr}@utu.fi

***Abstract*— Heterogeneous computing represents a trendy way to achieve further scalability in the high-performance computing area. It aims to join different processing units in a networked-based system such that each task is preferably executed by the unit which is able to efficiently perform that task. Memory hierarchy, instruction set, control logic, and other properties may differ in processing units so as to be specialized for different variety of problems. However, it will be more time-consuming for computer engineers to understand, design, and program on these systems. On the other hand, proper problems running on well-chosen heterogeneous systems present higher performance and superior energy efficiency. Such balance of attributes seldom makes a heterogeneous system useful for other fields than embedded computing or high-performance computing. Among them, embedded computing is more area and energy efficient while high-performance computing obtains more performance.**

GPUs, FPGAs or the new Xeon Phi are example of common computational units that, along with CPUs, can compose heterogeneous systems aiming to accelerate the execution of programs. In this paper, we have explored these architectures in terms of energy efficiency, performance, and productivity.

***Index Terms*— Heterogeneous Computing, GPU, GPGPU, Xeon Phi, FPGA, OpenCL.**

I. INTRODUCTION

Heterogeneous computing introduces a simple idea that is to run each task on the proper processing unit in order to achieve the optimal performance [1]. In reality, however, this is mostly impractical because heterogeneous systems require complex designs as well as advanced and intricate programming methods [10], [17-18]. In other words, both hardware and software become so complicated such that the obtained advantages may not be worthwhile regarding the development costs. Therefore, the theoretical interpretation should never be followed radically. Instead, the wise guideline would be: if the problem is considerable, a heterogeneous system might be profitable.

The most remarkable drawback is the difficulty of knowing when to utilize heterogeneous designs. This affair becomes decisive since the cost of developing heterogeneous systems could be very high, so we need to ensure that the achieved advantages will overcome this cost.

Principally in problems with high requirements, heterogeneous ideas perform well considering the fact that regular systems cannot meet those requirements [2]. For instance, on the Embedded Computing (EC) area, systems such as routers, mobile phones, mp3 players or medical devices have to process signals in real time. The simplest way would be to use general-purpose processors; however they cannot satisfy the area, response time and energy requirements that some embedded systems demand [3]. Hence, specific digital signal processors (DSP) are used instead. Scientific simulations are another example; they might take so much time while at the end of executions the results would not be useful. One option is to extend the system, but in then, it consumes considerably more area and power. For these problems, a high-performance heterogeneous system would provide the required performance to speed up simulations while saving in power consumption [14].

As a proof of the model, realistic videogames and intensive graphic applications have been executed with the cooperation of two processing units (CPU and GPU) since 90s, and their performance requirement would not have been met without such heterogeneous combination. Thus, heterogeneous computing is not a novel approach; it has been always presented wherever energy efficiency, performance or both were indispensable. The EC community has used heterogeneous designs mainly for area and energy efficiency while the High Performance Computing (HPC) community has used them for higher performance.

Heterogeneous computing has been always present although it has not received much attention in the past than today in the HPC community [26-32]. It is not obvious which factors made the concept so popular nowadays; however, the most likely reason is that few years ago the HPC market was too focused on getting more scalability by just stacking up servers of general-purpose processors. When GPUs proved to be very successful for massive parallel problems, the community took this as a revolutionary way to scale up the computational platforms. Then, in contrast to the homogeneous architectures that were dominating the market, this trend in HPC was called heterogeneous computing.

Today heterogeneous computing is clearly popular in the HPC area. Hundreds of research groups accelerate their projects with heterogeneous systems, numerous documents related to the subject are published continuously and new supercomputers increasingly incorporate heterogeneous designs, predominantly with GPUs. Modern cost-effective heterogeneous systems are commonly composed of few complex general-purpose cores and some specialized processing units, usually working as accelerators. Examples are a cluster of x86 cores along with GPUs, FPGAs and/or the brand new Xeon Phi [15].

II. BACKGROUND OF HETEROGENEOUS COMPUTING FROM HPC VIEWPOINT

Since the invention of the computer in the middle of the last century, the concept of domain-specific processing unit

[†]The first two authors contributed equally to the work.

was slightly disregarded by HPC engineers. The reason was that computing capacities were increasing continuously according to Moore's Law in such way that even high complex problems of that time could be easily solved few years later [4]. Furthermore, the matter was already difficult such that attempts in that epoch were to make everything simple and generic rather than to develop new architectures. So, not many domain-specific processing units or architectures were designed; only few institutions such as NASA (MPP), prestigious universities (CM5, ILLIAC IV) or computer manufacturers (Cray-1) have done some research in this domain [36]. In all cases, there are groups who could afford the extravagance cost of these systems and who do not mind about its rapid obsolescence due to the fast improvements in all areas of computing.

Over that period, the leading goal for most hardware engineers was to improve the single-core general-purpose CPUs. In fact, significant improvements were achieved, e.g. superscalar, pipelines, multithreading, branch prediction, and out-of-order execution [33]. On the other hand, most efforts in the software field at the same time were related to sequential programming on single-core CPUs, e.g. smarter compilers, new programming languages (Assembly, Haskell, C, Java, etc.), and highly optimized libraries.

By the end of the century, the single-core era has reached its end due to physical limitations [5]. The principal causing factors were power consumption, power dissipation, and complexity. As a simplification, we could say that power consumption increases exponentially with regard to the scalar performance. Therefore, several low-frequency cores could attain the same performance as one high-frequency core, but with the great advantage of consuming less power [5]. On the other hand, the power dissipation issue was getting to such unacceptable level that higher clock rates would have needed expensive cooling systems with prices exceeding the cost of the processor itself. Furthermore, making faster cores not only exacerbate these limitations but also miniaturization, reliability, design and manufacturing costs, which increase the complexity.

With the incoming millennium, a new period that we could call multi-core era was began. The above limitations made it impossible to grow in frequency and the only escape was to grow in space. Two roads emerged: multi-core architectures (few medium-high frequency complex cores) and many-core architectures (many low-medium frequency simple cores). The first model aimed to replace the previous single-core general-purpose processors while the second was intended for embarrassing parallel problems. Over that period hardware designers had to solve further issues caused by those new architectures [37][38] e.g. shared memory protocols, routing protocols, and the cache coherence problem. On the software side, developments were about parallel programming languages (e.g. MPI, OpenMP, and TBB). With regard to HPC, the common way to get higher performance was stacking up many general-purpose computers in clusters of hundreds of them [34].

Unfortunately for the multi-core era, problems are not always parallelizable and if they are, their scalability does not usually allow further performance improvement when the number of cores increases [6]. According to this philosophy, more and simpler cores have the potential of higher performance; however what actually makes that performance gain applicable is the available parallelism in programs and algorithms. Scalar codes with no parallel sections perform worse on these architectures compared to the first generation. For this reason many-core architectures were not became as popular as general-purpose processor. In addition, multi-core architectures were not expanded to use many complex cores. The reason was that the physical limitations of the previous era were still present when trying to place many complex cores [7][37].

Meanwhile something extraordinary happened. The tendency to make more generic GPUs turned into real general-purpose GPUs, which are able to efficiently solve any problem having potential data parallelization. Besides, their programmability was significantly improved. That is why the trend of stacking up servers was questioned, when just a simple GPU could achieve the same scalability than a little counterpart cluster.

From now to a near future is a heterogeneous era, at least in the HPC domain. High performance engineers agree that systems composed by few complex general-purpose cores and some specialized processors are a real alternative to just stacking up symmetric servers as well as a promising way to achieve high performance while saving energy. With these two key benefits, heterogeneous computing is in the spotlight of the community and all indications are that this trend will be the driving force to reach exascale-level of computing.

The final compelling reason for heterogeneous computing comes with the limits in performance. If physical limitations make it impossible to achieve indefinite performance improvements, it is very likely that there will be a turning point (see fig. 1) after which general-purpose processors will not reach domain-specific processors anymore. In our opinion this turning point should be close to the beginning of the predicted heterogeneous era.

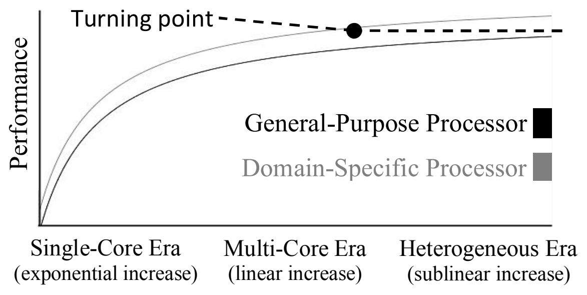

Fig. 1: Performance evolution of general-purpose processors against domain-specific processors.

III. Processing Units

In this section we give an overview of the most relevant processing units usually incorporated into high performance heterogeneous systems. We highlight their strong suits, expose their disadvantages and compare their designs.

A. Central Processing Units

As the elder of processors, CPU has been always the center of attention when talking about computer architectures. Since first models appeared, its mission has not been changed at all while its design has been enhanced in order to achieve greater performance. Significant improvements were about exploiting simple parallelism as instruction level parallelism (superscalar architectures and instruction pipelines), thread level parallelism (multithreading technologies), and data level parallelism (vector units). Additional improvements, not related to parallelism but in the level of importance, were to maximize the pipelines' flow (cache memories, branch prediction, speculative execution and out-of-order execution). In general, all the mentioned improvements have been focused to make CPUs robust and moderately fast running any code. Important features of CPU [8] can be summarized as follows:

- The degree of productivity is very high having mature programming languages with optimized compilers.
- Capable of running an operating system.
- Processing elements (few complex x86 cores) are able to execute 2 to 8 instructions per cycle with small vector units.
- Robust processor for nearly all problems and can execute any code moderately fast through an intensive use of the above technologies.
- Simple multithreading, supporting 2 threads per core.
- High clock rate which makes CPU the best choice for scalar problems but with high power consumption per thread.
- Large cache memories with up to 3 levels in order to hide memory latencies.
- The single precision float performance is theoretically up to 200 GFlops with vector units and 50 GFlops without them.
- Energy efficiency is around 2 GFlops/watt with vector units and 0.5 without them.

Table 1 shows a simple comparison between two CPUs (i7 Sandy Bridge and Xeon Sandy Bridge) regarding theoretical peak, thermal design power (TDP), and energy efficiency.

Table 1: Technical details of modern CPUs.

System	Theoretical Peak (GFlops)	TDP (Watts)	Energy Efficiency (GFlops/watt)
i7 Sandy Bridge	200~	130	1.5
Xeon Sandy Bridge	200~	87	2.3

B. Graphic Processing Units

Since first graphic cards, the design of their processors has changed significantly; from hard-coded pipelines to totally programmable massive parallel processors [9]. These changes have led to the use of GPUs as the real general-purpose processing units which can accelerate not only videos or images, but also problems where processing of large blocks of data should be done in parallel. This causes GPUs to become an important device for scientists and high performance programmers as their parallel capacity can suppose a powerful boost for embarrassing parallel problems. Therefore, GPUs are beneficial for certain types of problems in order to quickly achieve high performance in a simple way while saving power.

Beside high performance, the energy efficiency is actually the second success key. GPUs are in an order of magnitude more efficient than CPUs and due to this reason even supercomputers incorporate numerous GPUs as an affordable way to reach high performance without overflowing the system power consumption. Some of the features of GPUs [11-13] when used as accelerators are listed below:

- The degree of productivity is medium: programming languages are extensions of C, but programmers have to be aware of the underlying hardware.
- Composed of many simple cores in SIMD groups that can dispatch few thousands instructions per cycle.
- No robustness technologies, but compensated by complex multithreading supporting tens of threads.
- Medium clock rate which is inadequate for little workloads but with medium-low energy consumption per thread.
- Very wide memory bandwidth in order to feed all its cores.
- Very small on-chip memory that can behave as L1 cache or shared memory.
- Memory design goal is to save area in favor of more processing elements, which increases the latency and reduces its proficiency with complex access patterns.
- Overall design aims to data parallel problems with simple memory accesses and arithmetic intensive work.
- The single precision float performance is up to 3500 GFlops theoretically.
- Energy efficiency is about 16 GFlops/watt theoretically.

Table 2 shows a simple comparison between two GPUs (Geforce GTX 680 and Tesla K20X) regarding theoretical peak, thermal design power and energy efficiency.

Table 2: Technical details of modern GPUs.

System	Theoretical Peak (GFlops)	TDP (Watts)	Energy Efficiency (GFlops/watt)
Geforce GTX 680	3000~	195	15.85
Tesla K20X	3500~	235	16.8

C. Xeon Phi

Xeon Phi is the first commercialized product based on the Intel MIC Architecture (code name Knights Corner). It is a many-core processor with PCI connection, acting as a coprocessor, intends to gain a foothold in the HPC heterogeneous computing market. Its design consists of 60 x86 cores with dedicated 512-bit SIMD units, coherent L2 cache and ultra-wide ring bus connecting processors and memory blocks.

Its future and position in the market is still uncertain; the only indisputable fact is that Xeon Phi is the direct competitor of the general-purpose GPUs. The x86 compatibility, unlike in GPUs, is its best tactic to capture the attention of programmers who do not want to spend too much time translating projects to heterogeneous systems. Another major benefit is that programs can be executed in standalone mode by the Linux system that resides in its chip and therefore, the communication overhead between the host and device is reduced. Expected features of this new device [15-16] include the following:

- The degree of productivity is between high (codes designed for x86 processors are compatible) and medium (several modes of execution, wide vector units).
- They can perform as an accelerator or as a standalone processor, with four modes of execution.
- Each core (60 simple x86) is able to execute 16 double precision float operation per cycle with its vector unit.
- No robustness technologies, but compensated with moderate multithreading that supports 4 threads per core and hardware prefetching.
- Medium clock rate which makes it inadequate for little workloads but with medium energy consumption per thread.
- Ultra-wide memory bandwidth and ring interconnection.
- Small cache memories with 2 levels and coherence at the second with higher latency than CPU caches.
- Memory design goal is to save area in favor of more cores.
- Overall design aims to solve massive parallel problems with no necessary data parallelism. Nevertheless, it is far better for data parallelism because of the wide vector units.
- The single precision float performance is up to 2,000 GFlops theoretically, but extremely lower if vector units are not used.
- Energy efficiency is about 10 GFlops/watt theoretically.

D. Field Programming Gate Array (FPGA)

A FPGA is a semiconductor device containing logic blocks whose interconnections and function can be programmed in order to create a temporal hardware design. Such design maps into the device, an application-specific circuit that could implement from simple logic gates to highly complex circuitry like x86 cores.

FPGA emerges as an evolution of the concepts of programmable device developed in PAL and CPLD. Its main purpose is to act as a programmable ASIC and, in fact, it is very often used as a prototyping platforms of ASIC designs. In heterogeneous computing a well programmed FPGA can be the most powerful accelerator with about an order of magnitude of higher energy efficiency than other accelerators whether problems suit in its model. Its features [35] are the following:

- The degree of productivity is low: very complex and slow programming languages because the engineer is responsible of not only software but also hardware design.
- There are no processing elements or cores itself; the design can be totally adapted to the needs of the problem.
- They can exploit many techniques such as heavy pipelines, parallel work lines, optimal control logic designs, tuned functional units, lookup tables, etc..
- The clock rate is slow and limited by the design, routing and placement. In turn, the energy consumption is very low.
- The memory hierarchy is totally customizable: latencies, widths, interconnections and switching protocols.
- In general they are suitable for static problems with any kind of parallelism (data and task parallelism).
- Inadequate for dynamic problems that change at run-time or have a very complex or unpredictable flow.
- The single precision float performance will depends on the design, but fabricants claim up to 1 TFlop.
- Energy efficiency is again design dependent, but if normal consumptions are about 20~40W, the efficiency could theoretically be even 25 GFlops/watt.

E. Comparisons

With the above described features, we have compared the four processing units and represented the results in the following figures. Notice that this information has been collected for the current generation of processors and could change with the incoming generations (Intel 4th-gen, Nvidia Maxwell, etc.).

Fig. 2 shows how CPUs outperform in terms of productivity. It goes from very high productivity when using shared memory models to medium-high with message-passing languages. GPUs follow an unusual programming model that we place in the middle of the scale. It could go slightly up or down depending on the interaction with the on-chip memories. Xeon Phi processor has great productivity as a normal CPU. However, its compiler still needs important improvements and by now it does not automatically vectorize most of the codes. Thus, programmers have to vectorize by hand, what drops the productivity significantly. FPGAs get the lowest place because programmers have to care about all the aspect of the design. Their productivity goes from very low when using hardware description level languages to medium-low when utilizing high level synthesis tools.

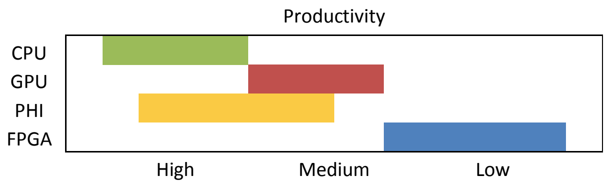

Fig. 2: Productivity comparison.

Its high clock rates, complex processing element, memory hierarchy and robustness technologies make CPUs the best options for scalar and complex codes. According to fig. 3, GPUs become the worst choice since they are specialized for massive data parallelism. Xeon Phi, with architecture somewhere between CPUs and GPUs, stays accordingly in the middle. FPGAs get a middle-low position since they have the ability to place very tuned designs that can eventually run specific codes in fewer cycles than CPUs, but in contrast their clock rate is lower.

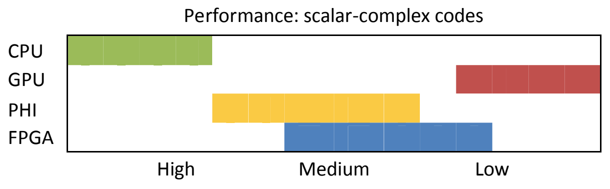

Fig. 3: Scalar-complex codes comparison.

As we can see in Fig. 4, CPUs might reach medium performance for parallel and simple codes. This occurs when taking advantage of their little vector units; otherwise this class of performance would be low for CPUs. GPUs obtain the higher place in all probability as they are designed for that intention. Xeon Phi, though having fewer cores than GPUs, still gets a considerable performance when using its wide vector units. FPGAs can route as many parallel path as needed as long as the area is not fully occupied, therefore their parallel performance becomes high in general.

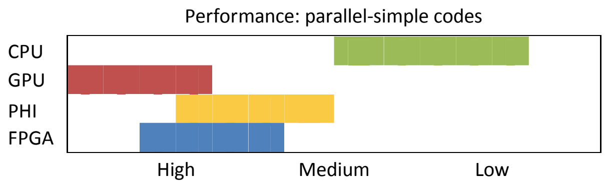

Fig. 4: Parallel-simple codes comparison.

As Fig. 5 shows, CPUs are very ineffective in terms of energy efficiency. One way to decrease their power consumption is using application-specific instructions such as vector instructions or CRC instructions; another is choosing more parallel oriented architectures with lower frequency and more cores. GPUs get high energy efficiency because, though their power consumption is one of the highest, their performance is high as well and the relation becomes profitable. The Xeon Phi reaches medium-high performances due to its daring architecture between CPU and GPU. FPGAs achieve the highest position because of their low power consumption.

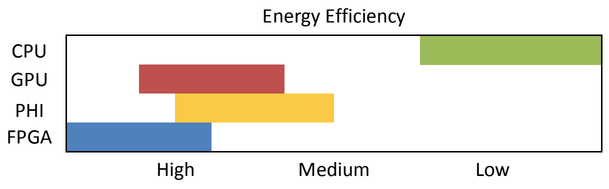

Fig. 5: Energy efficiency comparison.

IV. Results and Discussions

In order to present reasonable conclusion, we have explored publications claiming speedups for different problems through the use of our three heterogeneous models: CPU + (GPU / Xeon Phi / FPGA). Therein CPU becomes the leader while the secondary device acts as an accelerator.

A. Graphics Processing Units

GPUs are very suitable for high performance computing where problems have possible massive data parallelization.

For a very widespread problem as dense linear algebra, the comparison between an Nvidia library for GPU (CUBLAS) and Intel library for CPU (Intel MKL, carefully optimized math operations for CPU) shows that GPU is up to 17x faster than CPU [20].

Folding@home is a project of Stanford University that computes protein folding by using distributed computing. The statistics show that every client who computes protein folding using a GPU contributes, in average, 35x faster of what a CPU user does [21].

Another example from the applications of GPUs is the HotSpot technique, a simulation that tries to estimate the temperature of a chip by using structured grids. This is a very important simulation since nowadays one of the most transcendent limitations of chips is temperature. According to [22], GPUs offer a speedup of x25 over CPUs.

B. Xeon Phi

The same environment for GPU is suitable for Xeon Phi as well, with the advantage that it is not necessary to use a different programming model than CPU.

Financial predictions could be a very important field for Xeon Phi since Monte Carlo algorithms are very common in financial services. A comparison of Xeon Phi versus two Intel Xeon E5-2670 processors for Monte Carlo shows a speedup of 10.75x [23].

Another field of application is high performance computing domain such as astrophysics and genetics. A simulation of N-body, problem related to interacting particles influenced by gravitational, electric or magnetic fields (e.g. used in astrophysics for galaxy movement), shows a speedup of 6x for Xeon Phi versus two Intel Xeon E5-2680 processors [24].

C. *Field Programming Gate Array (FPGA)*

An important application for FPGAs is cryptography. For instance, servers that need to handle lots of encrypted authentications are benefited by using FPGAs. Advanced Encryption Standard (AES) is a very widespread symmetric-cryptography algorithm for encrypting data. FPGAs offer a good performance for this algorithm: 520x speed up versus E5503 Xeon Processor single core, and 15.75x versus AMD Radeon HD 7970. Besides the performance speedup, there are other characteristics of FPGAs that make them suitable choices for cryptography [25].

V. Conclusions and Summary

According to the exposed results, we conclude that the heterogeneous computing paradigm has proved to be very effective for high performance environments. The programming languages and compilers for heterogeneous computing have clearly improved and only minor improvements are to be done. However, we think there is still a lack of a unifying model for those programming languages.

OpenCL is not yet mature enough to exploit all the advantages of the presented architectures. Intel, AMD, Nvidia and Altera [19] have announced their support to OpenCL. Thereby, if it finally becomes the standard, programming for this model would be easier for heterogeneous systems while relaxing the necessary knowledge about the underlying hardware.

To summarize the topic in few points, we list the most important concepts which should be considered about heterogeneous computing as follows:

A. *Performance and energy efficiency*

To improve both performance and energy efficiency, heterogeneous computing suggests executing each task with the processing unit that can better performs that task. This will guarantee a good utilization of the architecture's capabilities and hence achieving high efficiency.

B. *Each code for its proper architecture*

Simple codes do not take advantage of complex architectures; neither does simple memory accesses for complex memory hierarchies and thus quite area is unused. Similarly, complex codes or memory accesses do not perform well in very simple systems so efficiency is reduced.

C. *Abundant parallelism*

Simulations, data mining, bioinformatics, astronomy, energy research, algebra, finances, etc. have abundant data parallelism and can perform more efficiently on SIMD architectures.

Servers, databases, complex software and in general parallel tasks with individual flow will perform more efficiently on MIMD architectures.

We recommend functional parallelism for clusters, while data parallelism for GPUs, Xeon Phi and FPGAs.

D. *Productivity and programming models*

Despite the advantages, heterogeneous computing has very different architectures and often different programming languages as well. This obviously reduces the productivity since it requires a better understanding of both problem and system. A possible future option is OpenCL, which aims to generate binary codes for different processing units regardless of the underlying hardware.

VI. Future Expectations

Our opinion is that heterogeneous computing will become more important in the coming years. After the boom of GPGPU computing at the end of 2009, the community has not talked about revolution anymore. However, all HPC groups are increasingly incorporating heterogeneous systems to their laboratories, what means this is not just a fad.

The future limitations on the chip miniaturization seem to be an important ally of heterogeneous computing. As the performance improvement is reduced year by year, the importance of heterogeneous computing will probably grow inversely. If eventually a time arrives when transistors cannot be reduced anymore, then the only alternative to achieve higher performance will be by expanding the chip area. Nonetheless, when chip area cannot be expanded due to other limitations or when area efficient solutions are demanded, heterogeneous computing will be the only way to increase efficiency, and thereby it will become significantly important, as we show in fig. 6.

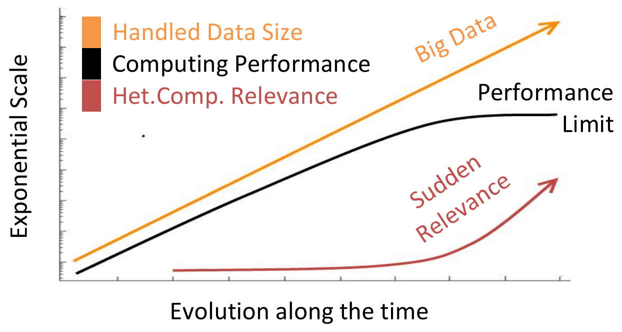

Fig. 6: Our prediction about heterogeneous computing relevance.

References

[1] A.A Khokhar et al. "Heterogeneous computing: Challenges and opportunities," in Computer, 1993, vol. 26, no 6, p. 18-27.

[2] J. Manyika et al. "Big data: The next frontier for innovation, competition, and productivity". McKinsey Global Institute, 2011, pp. 1-137.

[3] K.G. Shin and P. Ramanathan. "Real Time Computing A New Discipline Of Computer Science And Engineering," in Proceedings Of The IEEE, 82(1), 1994, pp. 6-24.

[4] G. Moore, "Progress In Digital Integrated Electronics," in proceeding of International Electron Devices Meeting, vol. 21, 1975, pp. 11-13.

[5] M. Horowitz, “Scaling, power, and the future of CMOS,” in proceeding of VLSID '07, 2007, pp. 23.

[6] G.M. Amdahl, “Validity of the single processor approach to achieving large-scale computing capabilities,” in proceeding of AFIPS ’67, 1967, pp. 483-485.

[7] H. Esmaeilzadeh, “Dark Silicon and the End of Multicore Scaling,” in proceeding of ISCA '11, 2011, pp. 365-376.

[8] “Intel® 64 and IA-32 Architectures Optimization Reference Manual,” Internet: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html, Apr. 2012 [May. 2013].

[9] J. Nickolls and W.J. Dally, “The GPU Computing Era,” in proceeding of IEEE Micro, 30 (2), March 2010, pp. 56-69.

[10] “Cuda C Programming Guide,” Internet: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, [May. 2013].

[11] “Gefore GTX 680 Whitepaper,” Internet: http://www.geforce.com/Active/en_US/en_US/pdf/GeForce-GTX-680-Whitepaper-FINAL.pdf, pp. 3-12 [May. 2013].

[12] “Tesla KSeries Overview,” Internet http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf, [May. 2013].

[13] “Tesla K20X Board Specification,” Internet: http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v05.pdf, Nov. 2012 [May. 2013].

[14] “Titan Supercomputer,” Internet: http://www.olcf.ornl.gov/titan, [May. 2013].

[15] “Intel MIC Xeon Phi,” Internet: http://www.intel.com/xeonphi, [May. 2013].

[16] “Xeon Phi Datasheet,” Internet: http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-datasheet.pdf, [May. 2013].

[17] “OpenCL 1.2 Specification,” Internet: http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf, Sep. 2012 [May. 2013].

[18] “Heterogeneous System Architecture: a technical review,” Internet: http://developer.amd.com/wordpress/media/2012/10/hsa10.pdf, Aug. 2012 [May. 2013].

[19] “OpenCL for Altera FPGAs: Accelerating Performance and Design Productivity,” Internet: http://www.altera.com/products/software/opencl/opencl-index.html, [May. 2013].

[20] “CUDA 5.0 Math Libraries Performance Report,” Internet: https://developer.nvidia.com/sites/default/files/akamai/cuda/files/CUDADownloads/CUDA_5.0_Math_Libraries_Performance.pdf [May. 2013].

[21] A. Beberg and V. S. Pande. “Folding@home: lessons from eight years of distributed computing,” In proceeding of IEEE International Symposium on Parallel & Distributed Processing, 1-8 (2009)

[22] P. Springer, “Berkeley Dwarfs on CUDA,” Rwth Aachen University. Internet: http://hpac.rwth-aachen.de/people/springer/cuda_dwarf_seminar.pdf, pp. 14-15 [May. 2013].

[23] “Intel® Xeon Phi™ Product Family Performance,” Internet: http://www.intel.com/content/dam/www/public/us/en/documents/performance-briefs/xeon-phi-product-family-performance-brief.pdf, Apr. 2013 [May. 2013].

[24] “Test-driving Intel Xeon Phi coprocessors with a basic N-body simulation,” Internet: http://research.colfaxinternational.com/file.axd?file=2013%2f1%2fColfax_Nbody_Xeon_Phi.pdf, Jan. 2013. [May. 2013].

[25] “40Gbit AES Encryption Using OpenCL and FPGAs,” Internet: http://www.nallatech.com/images/stories/technical_library/white-papers/40_gbit_aes_encryption_using_opencl_and_fpgas_final.pdf, [May. 2013].

[26] Q. Liu and W. Luk, “Heterogeneous Systems for Energy Efficient Scientific Computing,” in proceeding of ARC'12, 2012, pp. 64-75.

[27] A.R. Brodtkorb et al, “State-of-the-art in heterogeneous computing,” in IOS Press, 18(1), 2010, pp. 1-33.

[28] B. Gaster et al., “Heterogeneous Computing with OpenCL,” 2013.

[29] S. Che et al., “Rodinia: A Benchmark Suite for Heterogeneous Computing,” in Proceedings of the IEEE International Symposium on Workload Characterization, 2009, pp. 44–54.

[30] R. Inta et al., “Chimera: an off-the-shelf CPU/GPGPU/FPGA hybrid computing platform,” International Journal of Reconfigurable Computing, 2012, pp. 2-2, January 2012

[31] R. Inta and D. J. Bowman, “An FPGA/GPU/CPU hybrid platform for solving hard computational problems,” in Proceedings of the eResearch Australasia, Gold Coast, Australia, 2010.

[32] S. Che et al., “Accelerating compute-intensive applications with GPUs and FPGAs,” in Proceedings of the Symposium on Application Specific Processors (SASP '08), Anaheim, Calif, USA, 2008, pp. 101–107.

[33] J.L. Hennessy and D.A.Patterson, “Computer architecture: a quantitative approach,” Morgan Kaufmann, 2011.

[34] M. Baker, “Cluster computing white paper,” arXiv preprint cs/0004014, 2000.

[35] V. Betz et al., “Architecture and CAD for deep-submicron FPGAs,” Kluwer Academic Publishers, 1999.

[36] A.R. Hoffman and J.F. Traub, “Supercomputers: directions in technology and applications,” National Academies Press, 1989.

[37] M. Daneshtalab et al., “Memory-Efficient On-Chip Network with Adaptive Interfaces,” IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems (IEEE-TCAD), Vol. 31, No. 1, pp. 146-159, Jan 2012.

[38] M. Ebrahimi et al., “HAMUM – A Novel Routing Protocol for Unicast and Multicast Traffic in MPSoCs,” in Proceedings of 18th IEEE Euromicro Conference on Parallel, Distributed and Network-Based Computing (PDP), pp. 525-532, February 2010, Italy.