An Overview of Selected Hybrid and Reconfigurable Architectures

SASA STOJANOVIC, DRAGAN BOJIC,
and MIROSLAV BOJOVIC, University of Belgrade, Serbia
MATEO VALERO, Polytechnic University of Catalonia, Spain
VELJKO MILUTINOVIC, University of Belgrade, Serbia

Node level heterogeneous architectures have become attractive in recent years for several reasons: Compared to traditional symmetric CPUs, they offer high performance for real applications, and can be energy and/or cost efficient. In this paper, we give an overview of the state-of-the-art in heterogeneous computing, focusing on some common architectures: The NVidia and the ATI graphics processing units (GPUs), the Cell Broadband Engine Architecture (CBEA), the ClearSpeed processor, the field programmable gate array (FPGA) accelerator solutions from Maxeler MaxNodes (MAX), the SGI systems (RASC), and the Convey Hybrid-Core Computer. We present a review of the hardware, available software tools for each solution, a quantitative and a qualitative comparison of the architectures, and we give our view on the future of heterogeneous computing.

Key Words and Phrases:
Heterogeneous Architectures, Data Flow Architectures, Reconfigurable Architectures, General Purpose GPU Computing

I. Introduction

In this comparative study we survey commercially available systems (as opposed to academic projects). For less recent contributions, and for contributions of academic research, interested readers are referred to past survey papers of these and other authors [4, 5, 9, 13, and 14]. Each one of the solutions presented here is a proven, commercially more or less successful representative of a particular computing style. Examination of their comparative advantages and disadvantages serves to envision future trends for such systems. We focus on a single node, noting that systems can be formed by connecting multiple such nodes together. In this context, the term node refers to the smallest unit of hardware that is capable of working as a standalone system.

It is important to underline the following: This survey, like any other survey, presents data obtained from original papers of authors, or web sites of vendors, who introduced the surveyed concepts/systems. Thus, data obtained from different sources were generated under different conditions, and therefore, any direct performance comparison is always questionable. In such conditions, what provides the best comparison is the insight into the facts related to the number of systems sold and the facts related to exploitation experiences of those who use the presented systems/concepts, in order to solve their critical problems on a daily basis.

II. Problem Statement

The continuous performance improvement of microprocessor technology is not likely to hold in the future. This positive trend observed in the past, known as the Mooreôs law, was due to two main factors: Increasing clock speed and improvement in instruction level parallelism.

Over the past years, the speed was increasing by constantly growing the clock frequency, while the capacity has always been boosted by the reduction of transistor size. Consequently, programs could

This work has been partially funded by the Ministry of Education and Science of the Republic of Serbia (III44009, TR32047, and III44006).
Authorsô addresses: Sasa Stojanovic, School of Electrical Engineering, University of Belgrade, Serbia, stojsasa@etf.bg.ac.rs; Dragan Bojic, School of Electrical Engineering, University of Belgrade, Serbia, bojic@etf.bg.ac.rs; Miroslav Bojovic, School of Electrical Engineering, University of Belgrade, Serbia, mbojovic@etf.bg.ac.rs; Mateo Valero, Technical University of Catalonia, Spain, mateo@ac.upc.es; Veljko Milutinovic, School of Electrical Engineering, University of Belgrade, Serbia, vm@etf.bg.ac.rs.

be speeded-up through instruction level parallelisms (ILP) in a way that was transparent to the user. Nowadays, the speed increase trend is getting more and more difficult to maintain. Enhancements of the ILP-design are not followed anymore by system performance [29].

In addition, more recently, power consumption is becoming one of the main limitations. To solve this problem, the research community and industry have started using a large amount of available chip area to implement several processors, thus creating the so called chip multicore processing systems. If there is enough parallelism, multicore solutions can achieve speedups over single cores with the same power consumption.

Reconfigurable computing implies a hardware that can be reconfigured to implement application-specific functions. The basic concept of reconfigurable computing (RC) was proposed in the 1960s, but has only recently become feasible. Reconfigurable computers consist of a flexible hardware fabric and one or more (heterogeneous and/or customized) processors. These systems customize the internal structure of the platform to suit the application, leading to faster run-times, more power-efficient processing, and better device utilization. Dynamically reconfigurable systems go even further; their internal structure may even adapt to the workload, at runtime.

The reconfigurable systems have a lot of potential for speedup, thanks to the fact that execution time of some loop with *n* iterations and *o* operations per iterations depends on the sum of *n* and *o*, while on other systems, execution time of same loop depends on product of n and o. The reason for this is a very deep pipeline implemented in FPGA, as explained in Figure 1.

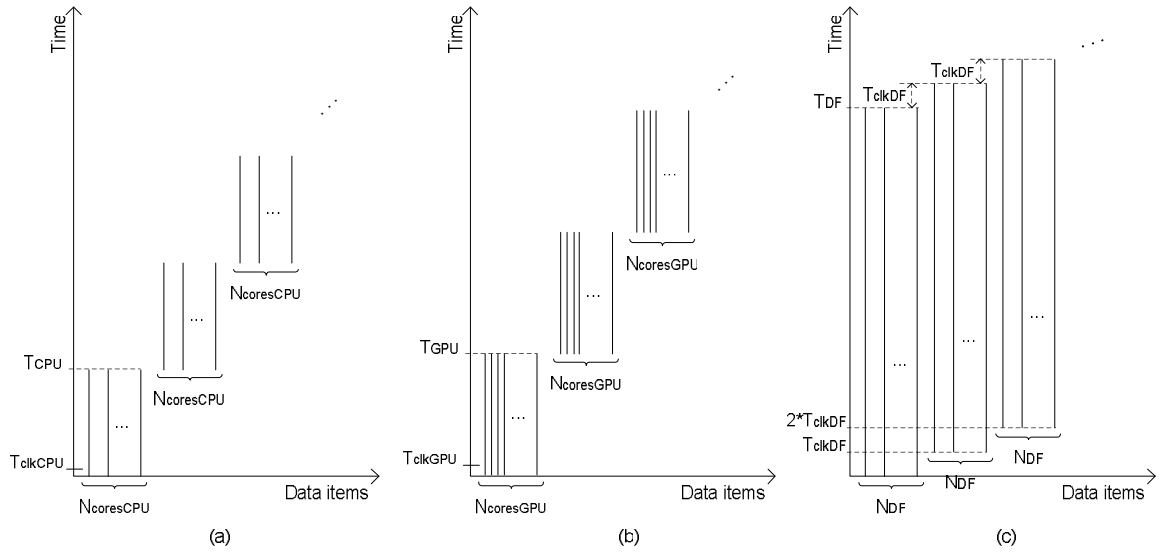

Figure 1: Graphical representation of execution units potentials on three architectural styles: (a) The multicore CPU execution (MC) model (Intel, CBEA, and ClearSpeed): $N_{coresCPU}$ ï number of available cores in the CPU, T_{clkCPU} ï clock period of the CPU, T_{CPU} ï execution time on the CPU, the case of one loop iteration; (b) The GPU execution (GP) model (NVidia and AMD ATI): $N_{coresGPU}$ ï number of available cores in the GPU, T_{clkGPU} period of clock in GPU, T_{GPU} ï execution time on the GPU of one iteration; (c) Data flow execution (DF) model (MaxNodes, SGI RASC, Convey): N_{DF} ï number of implemented data flows, T_{clkDF} ï period of clock for a data flow machine, T_{DF} ï latency of data flow pipeline. On the x-axis are input data items, and on y-axis is the execution time. Loop iterations are not data-dependent.

The execution times of Figure 1 are derived from the following formulae:

(a) $T_{CPU} = N_{OPS} * C_{CPU} * T_{clkCPU}$

$$t_{CPU} = N * T_{CPU} / N_{coresCPU}$$
$$= N * N_{OPS} * C_{CPU} * T_{clkCPU} / N_{coresCPU},$$

(b) $T_{GPU} = N_{OPS} * C_{GPU} * T_{clkGPU}$

$$t_{GPU} = N * T_{GPU} / N_{coresGPU}$$
$$= N * N_{OPS} * C_{GPU} * T_{clkGPU} / N_{coresGPU},$$

(c) $T_{DF} = N_{OPS} * C_{DF} * T_{clkDF}$

$$t_{DF} = T_{DF} + (N - 1) * T_{clkDF} / N_{DF}$$
$$= N_{OPS} * C_{DF} * T_{clkDF} + (N - 1) * T_{clkDF} / N_{DF}$$

Meaning of the variables in the above formulae is given in the caption of Figure 1.

III. Existing Solutions and Their Criticism

In this section, we give a short overview of seven existing solutions for the problem specified above, namely: CBEA, ClearSpeed, SGI RASC, Convey, Maxeler MaxNodes, NVidia GPU, and AMD ATI GPU computing. In general, the reviewed solutions are well established commercial solutions and give good results under the conditions of interest for their operating environments and problem solving domains, but each one of them has some comparative advantages and disadvantages.

This section starts with a classification of existing solutions in the domain of contemporary computing architectures. It continues with an overview which, for each and every example, gives the following main points: (a) The 7 Ws of the solution (who, where, when, whom, why, what, how, or a subset there off), (b) Essential elements of the approach, in terms of the architecture and the programming model, and finally a conclusion section where we critically compare the presented solutions in their current state.

III.A. Classification Criteria

The basic division is at control-flow (CF) and data-flow (DF) ñsupercomputersò. In essence, the control-flow supercomputers compile down to machine code and execute the code using one of several possible execution paradigms; the data-flow supercomputers compile down to GTL (gate transfer level), which brings a better speed/watt at the expense of more difficult programming. Therefore, one possible classification criterion is the flow concept, which results in CF and DF.

Control flow ñsupercomputersò span over a wide range of different architectures. These architectures can be composed of cores that all belong to the same architectural type, or of cores that belong to several different architectural types. Based on this, CF architectures can be further divided into two additional groups: Homogeneous architectures (HOM, e.g. Intel processors) or heterogeneous architectures (HET, e.g. NVidia GPU, AMD ATI GPU, IBM CELL, and ClearSpeed).

Homogeneous architectures are outside of the scope of this paper, and will not be further observed in this paper.

Heterogeneous control flow architectures are most often composed of one to several larger and highly sophisticated cores and a lot of simpler and smaller cores. Depending on how these two groups of cores are coupled, the HET subgroup can be subdivided into two subgroups: Loosely coupled systems (LC, e.g. NVidia GPU and AMD ATI GPU), where these two groups of cores are

implemented on two or more chips interconnected on printed boards, and tightly coupled systems (TC, e.g. IBM CELL and ClearSpead), where all cores are on the same chip.

A combination of the CPU and flexible hardware accelerators can be used to increase the computation speed of applications. One possible way to implement hardware accelerators is by using the data-flow concept.

In the case when a data-flow concept is used, FPGAs can be treated as the enabling technology in which a set of processors and a set of custom hardware accelerators are implemented and made to communicate with each other using a network. The system can be made adaptable by modifying the computation infrastructure. This can be done at compile-time using full device reconfiguration or at run-time by means of partial device reconfiguration.

Essential difference of various data-flow machines is that some of them use explicit data-flow, while the others use implicit data-flow. So, one possible classification criterion is explicit (EDF, e.g., MaxNode) versus implicit (IDF, e.g., Convey and SGI RASC).

Machines that use implicit data-flow can be further differentiated by the granularity of the model that is presented to the programmer. The first ones are seen as coprocessors, where each coprocessor has a set of instructions that can be executed (CG, e.g. Convey). The other ones are seen as some chip area on which a set of operation units is implemented and used in arbitrary combinations (FG, e.g. SGI RASC).

Our classification is given in Figure 2, together with specifications of classification criteria, outcomes generated by applying these criteria, and the representative examples discussed in this paper. To make the classification more appropriate for the focus of this paper, some issues that could be important are left out of the classification.

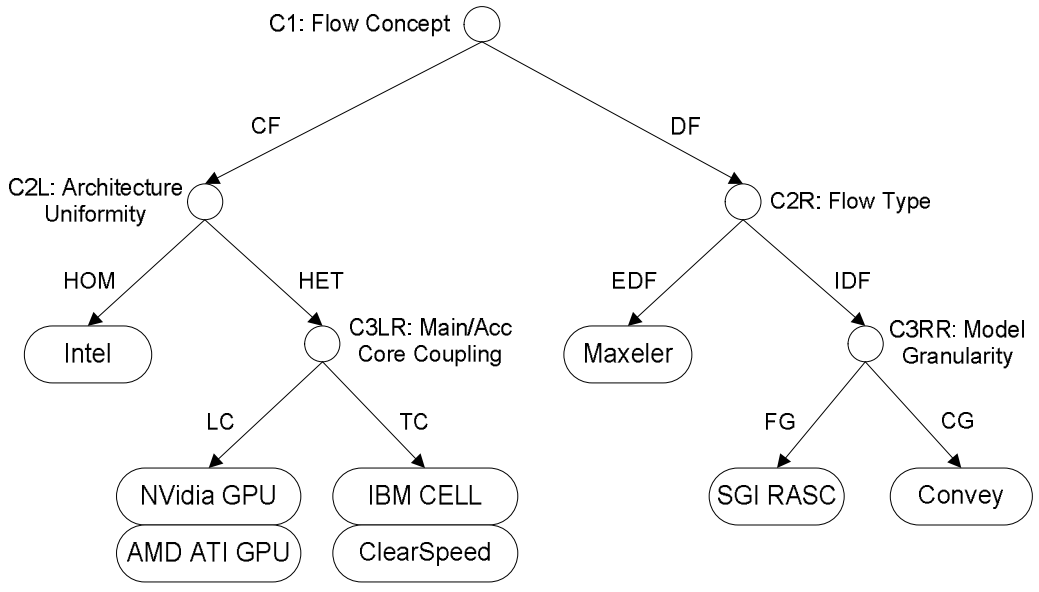

Figure 2: A classification of hybrid architectures

Issues of importance in CF supercomputers, not encompassed by the presented classification, are discussed next.

Among the control-flow machines, two basic paradigms exist for parallel computation on multiprocessors. The first paradigm is *message passing interface (MPI)*, usually available where communication among processors happens in a network. The second paradigm, the *shared memory* or *symmetrical multi processing (SMP)* is applied in bus-based systems, where communications are performed via the shared memory.

Heterogeneous multiprocessing refers to the use of different processing cores to maximize performance. This paradigm can be used for both, MPI or SMP.

In most of the existing research efforts, the purpose is to have an efficient mapping of a set of threads onto a set of processors. Two possibilities have been presented that address this purpose: *Simultaneous multithreading (SMT)* [12, 18, and 20] and *chip multiprocessor (CMP)* [3, 17, 19, 23, and 29].

An SMT chip is based on a superscalar processor with a set of units for parallel execution of instructions. Given a set of threads, resources are dynamically allocated by extracting several instructions to be executed in parallel. Threads that need long memory access are preempted to avoid idle states of processors. This paradigm can also be used with both, MPI or SMP.

Instead of using only one superscalar core to execute instructions in parallel, the CMPs use many processor cores to compute threads in parallel. Each processor has small first-level local instruction and data caches. A second level cache is available for all the processors on a chip. The CMPs target applications consist of a set of independent computations that can be easily mapped to threads.
Existence of independent computations is usually the case in database or web server applications where transactions do not rely on each other. In CMPs, threads having long memory access are preempted to allow others to use the processor [4].

Issues of importance in DF supercomputers, not encompassed by the presented classification are discussed next.

The type of interaction with the host machine can also be used as a classification paradigm. Some data-flow machines use Front Side Bus (FSB). Others use standard peripheral buses, e.g. Peripheral Component Interface Express (PCIe). Still the others use a custom bus, e.g. Non Uniform Memort Access Link (NUMA Link).

III.B. Presentation of Existing Solutions

We shall now present details of systems selected as commercially successful. The NVidia Fermi GPU and the AMD ATI Cypress GPU are examples of heterogeneous systems in which a CPU is accompanied by a highly multithreaded single instruction/multiple data (SIMD) accelerator. The CBEA and ClearSpeed are representatives of heterogeneous multicore processors. Finally, the Maxeler Dataflow Engines (MaxNodes), the SGI RASC, and the Convey coprocessor belong to the domain of symmetrical multicore processors accompanied by reconfigurable hardware accelerators.

III.B.1. NVidia Fermi GPU

A GPU is a symmetric multi-core processor that is exclusively accessed and controlled by the CPU, making the two a heterogeneous system. The GPU operates asynchronously from the CPU, enabling concurrent execution and memory transfer. The graphics processing unit (GPU), first invented by NVIDIA in 1999, is the most pervasive parallel processor to date. Todayôs GPUs greatly outpace CPUs in arithmetic throughput and memory bandwidth, making them the ideal processor to accelerate a variety of data parallel applications [28].

The Fermi based GPU (see Figure 3), implemented with 3.0 billion transistors, features up to 512 CUDA cores. A CUDA core executes a floating point or integer instruction per clock for a thread. The 512 CUDA cores are organized in 16 SMs of 32 cores each. Each CUDA processor has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU). The GPU has six 64-bit memory partitions, for a 384-bit memory interface, supporting up to a total of 6 GB of GDDR5 DRAM memory. A host interface connects the GPU to the CPU via PCI-Express. The GigaThread global scheduler distributes thread blocks to SM thread schedulers [28].

The CUDA is the hardware and software architecture that enables NVidia GPUs to execute programs written with C, C++, Fortran, OpenCL, DirectCompute, and other languages. A CUDA program calls parallel kernels. A kernel executes in parallel across a set of parallel threads. The programmer or compiler organizes these threads in thread blocks and grids of thread blocks. The GPU instantiates a kernel program on a grid of parallel thread blocks [28].

Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block, program counter, registers, per-thread private memory, inputs, and output results. A thread block is a set of concurrently executing threads that can cooperate among themselves through barrier synchronization and shared memory. A thread block has a block ID within its grid. A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls [28].

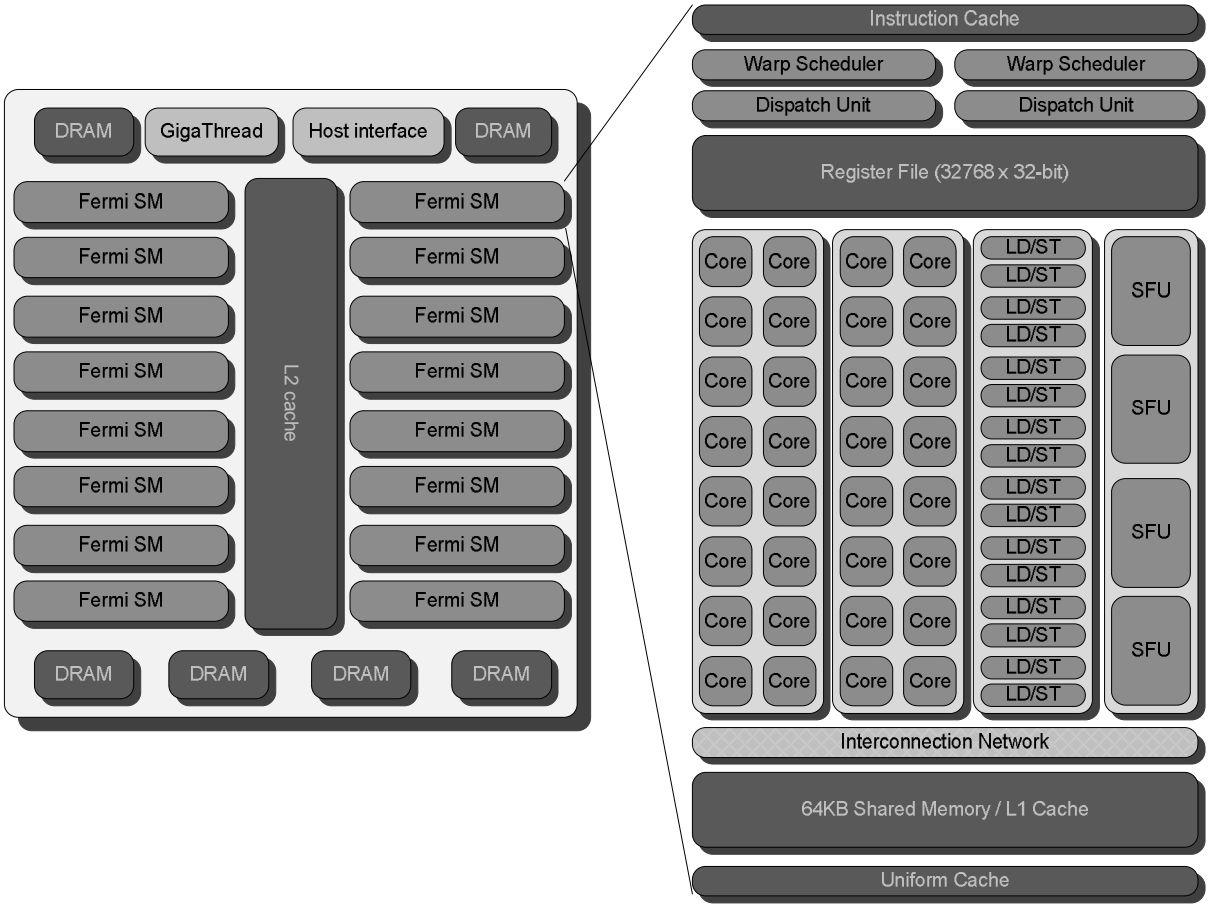

Figure 3: The Fermi architecture

In the CUDA parallel programming model (shown in Figure 4), each thread has a per-thread private memory space used for register spills, function calls, and C automatic array variables.

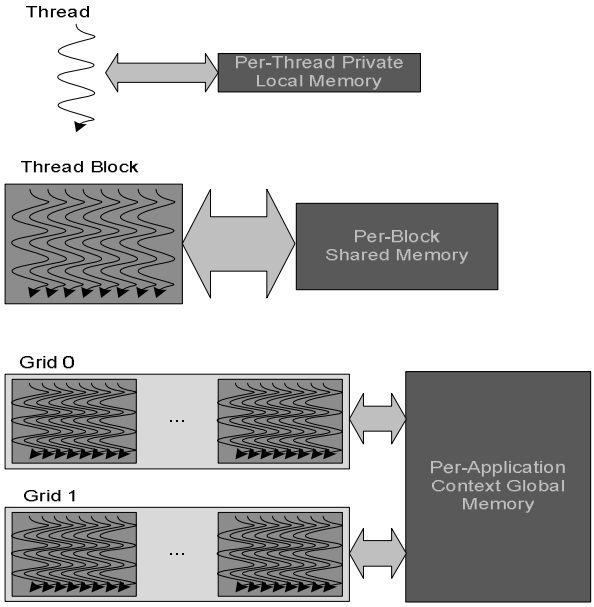

Figure 4: The CUDA programming model

Each thread block has a per-Block shared memory space used for inter-thread communication, data sharing, and result sharing in parallel algorithms. Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization [28].

The CUDAôs hierarchy of threads maps to a hierarchy of processors on the GPU; a GPU executes one or more kernel grids; a streaming multiprocessor (SM) executes one or more thread blocks; and CUDA cores and other execution units in the SM execute threads. To improve performance by memory locality, the SM executes threads in groups of 32 threads called a warp [28].

III.B.2. AMD ATI

The other GPU available on the market, and also relatively successful, comes from AMD ATI. The first ATIôs GPGPU appeared in 2004. Code name Cypress represents the most successful AMD ATIôs GPU architecture of today.

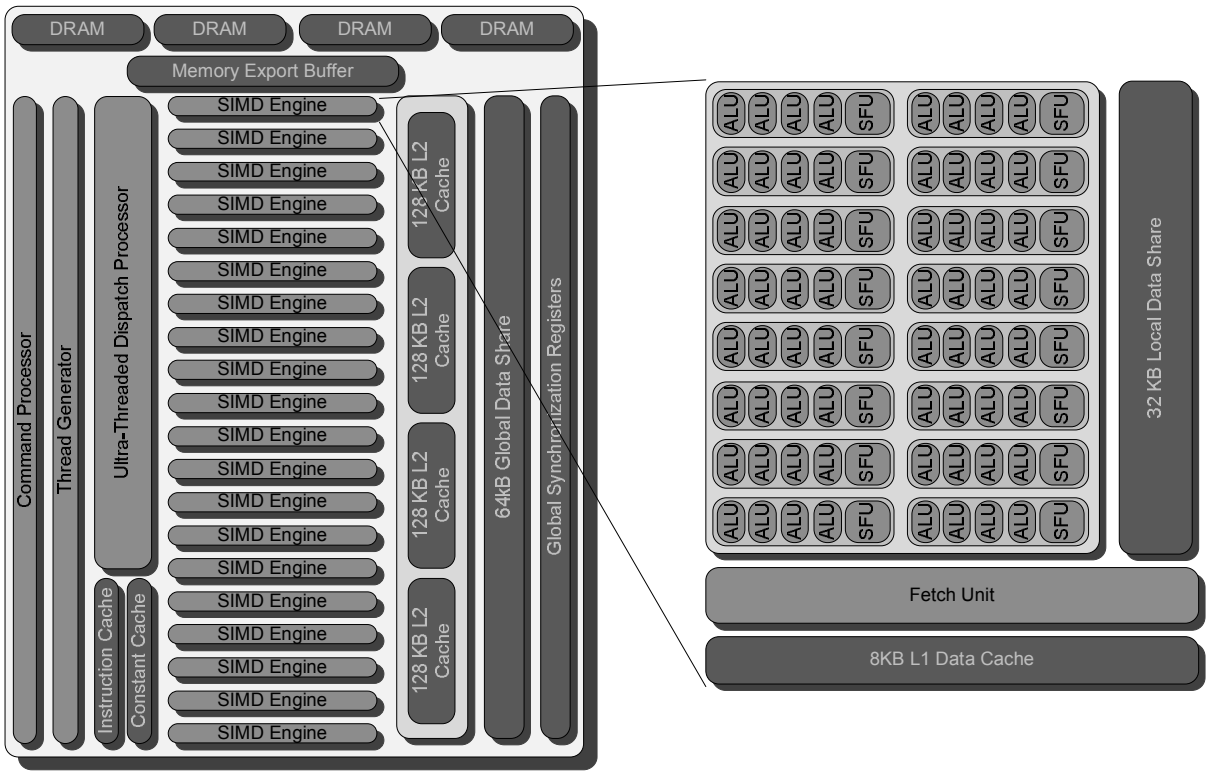

Figure 5: The Cypress architecture

The Cypress based GPU (Figure 5), implemented with 2.15 billion transistors, has 320 cores with 1600 processing elements (ALU). They are organized as 20 SIMDs, each one made of 16 units, what AMD calls thread processors. These processors, as the primary execution units, are superscalar and five ALUs wide. The fifth ALU is a superset of the others, capable of handling more advanced operations like trascendentals (e.g., sin, cos, arc tan, sqrt, pow, etc.). The execution units are pipelined with eight cycles of latency, but the SIMDs can execute two hardware thread groups, or "wavefronts" in AMD parlance, in interleaved fashion, so the effective wavefront latency is four cycles. In every cycle, an execution unit can issue 4 single precision operation, or two single precision multiply add operation, or two double precision operations, or one double precision multiply add operation, and one more single precision operation or some transcendental function. Every SIMD is equipped with 32KB of local memory and with 8KB L1 cache. There is also a global shared memory of 64KB, and four banks of L2 cache, 64KB each. Processor has four 64-bit interfaces to the external memory, that gives the total bandwidth of 156.3 GB/s [1 and 37].

AMD GPUs are supported by Open Calculation Language (OpenCL), an industry standard API that is open, multiplatform development platform for heterogeneous architectures. It is very similar to NVidiaôs CUDA, but not only intended for use with GPU accelerators.

Since OpenCL is meant to target not only GPUs but also other accelerators, such as multi-core CPUs, flexibility is given in the type of compute kernel that is specified. Compute kernels can be thought of either as data-parallel, which is well-matched to the architecture of GPUs, or task-parallel, which is well-matched to the architecture of CPUs.

A compute kernel is the basic unit of executable code and can be thought of as similar to a C function. Execution of such kernels can proceed either in-order or out-of-order depending on the parameters passed to the system when queuing up the kernel for execution. Events are provided so that the developer can check on the status of outstanding kernel execution requests and other runtime requests.

In terms of organization, the execution domain of a kernel is defined by an N-dimensional computation domain. This lets the system know how large of a problem the user would like the kernel to be applied to. Each element in the execution domain is a work-item and OpenCL provides the ability to group together work-items into work-groups for synchronization and communication purposes.

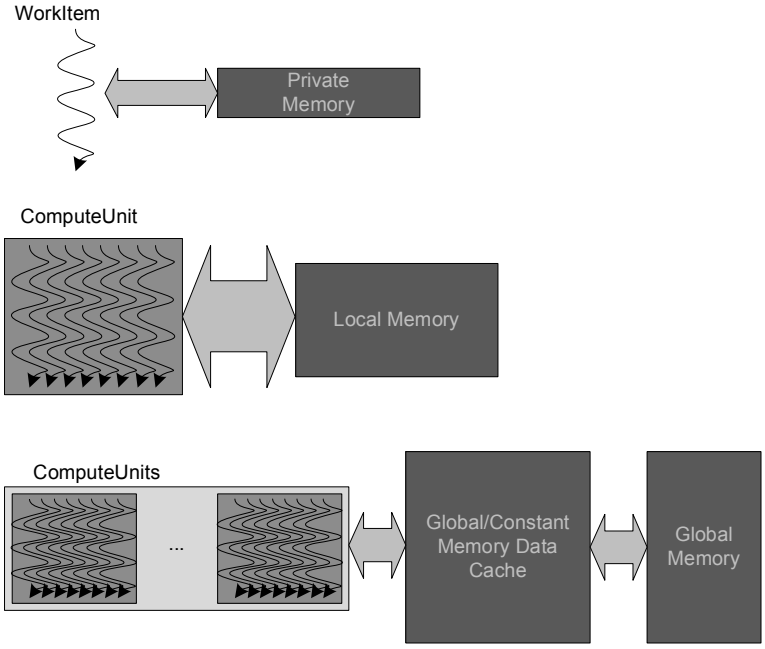

Figure 6: Memory hierarchy in OpenCL

OpenCL defines a multi-level memory model with memory ranging from private memory visible only to the individual compute units in the device to global memory that is visible to all compute units on the device. Depending on the actual memory subsystem, different memory spaces are allowed to be collapsed together.

OpenCL 1.0 defines 4 memory spaces: Private, local, constant and global. Figure 6, shows a diagram of the memory hierarchy defined by OpenCL.

Private memory is memory that can only be used by a single compute unit. This is similar to registers in a single compute unit or a single CPU core.

Local memory is memory that can be used by the work-items in a work-group. This is similar to the local data share that is available on the current generation of AMD GPUs.

Constant memory is memory that can be used to store constant data for read-only access by all of the compute units in the device during the execution of a kernel. The host processor is responsible for allocating and initializing the memory objects that reside in this memory space. This is similar to the constant caches that are available on AMD GPUs.

Finally, global memory is memory that can be used by all the compute units on the device. This is similar to the off-chip GPU memory that is available on AMD GPUs [27].

On Figure 7 is shown an example of vector addition.

```
__kernel void vec_add (__global const float *a,
                       __global const float *b,
                       __global float *c)
{
        int gid = get_global_id(0);
        c[gid] = a[gid] + b[gid];
}
```

Figure 7: The OpenCL example: Vector addition

III.B.3. Cell Broadband Engine Architecture (CBEA)

The Cell Broadband Engine Architecture (CBEA), developed by Sony, Toshiba, and IBM, was conceived as the next generation chip architecture for multimedia and compute-intensive processing [34 and 35]. It was first used in the Sony PlayStation 3 game console. Currently, IBM offers two processors based on the CBEA: The Cell Broadband Engine (Cell/B.E.), which is used in Sonyôs PlayStation 3 and IBMôs Cell blades, and the PowerXCell 8i processor, used in Roadrunner, the worldôs first petaflop supercomputer.

Figure 8 shows the major components of a Cell/B.E. processor: The main processing element (the Power processor element, or PPE), the parallel processing accelerators (synergistic processor elements, or SPEs), the on-chip interconnect (a bidirectional data ring known as the element interconnect bus, or EIB), and the I/O interfaces (the memory interface controller, or MIC, and the Cell Broadband Engine interface, or CBEI).

The PPE contains a 64-bit PowerPC processor unit (PPU), two separate 32-Kbyte level 1 caches for instructions and data, and a unified 512-Kbyte level 2 cache for instructions and data. The PPU supports two-way simultaneous multithreading. The PPE can complete two double precision operations per clock cycle, resulting in a peak performance of 6.4 Gflops at 3.2 GHz.

Each SPE is an accelerator core based on a single-instruction multiple data (SIMD) reduced instruction set computer processor. The SPE operation semantics are similar to those of the PowerPC SIMD extensions. The SPE has a unified 128-entry 128-bit wide SIMD register file to store operands of all data types. Each SPE can directly address a local store of 256 Kbytes for instruction and data references.

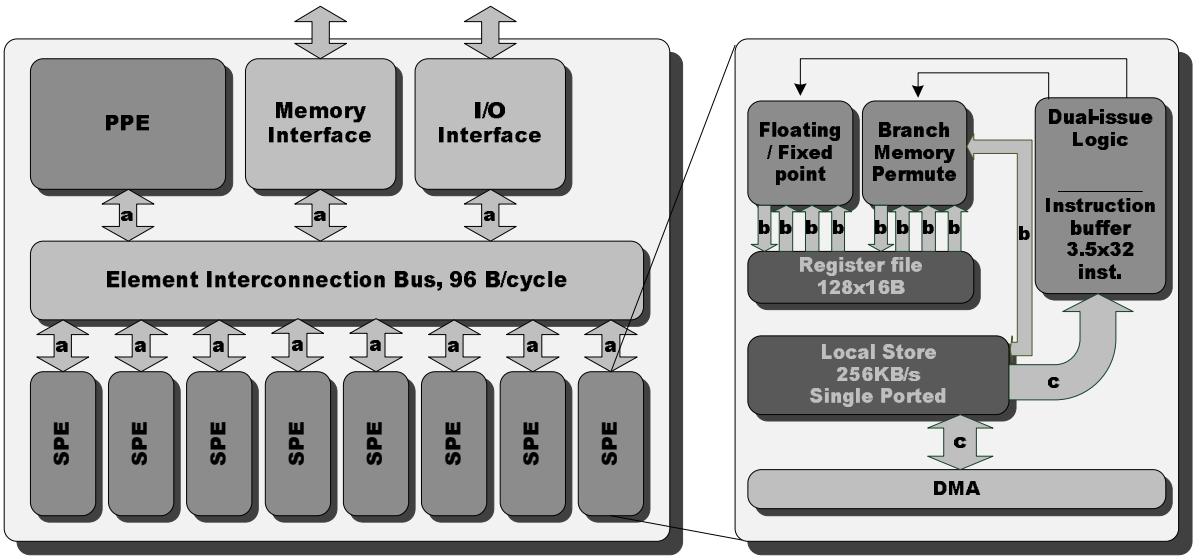

Figure 8: The Cell BE Architecture

This local store is explicitly managed by softwareð that is, the compiler or programmerð by way of MFC block transfers between system memory and the local store, as shown in Figure 9. The MFC can issue up to 16 simultaneous direct memory access (DMA) operations of up to 16 Kbytes each between the local store and system memory. Unlike traditional hardware caches, this organization lets the system prefetch large memory operands into on-chip memory in parallel with program execution, avoiding the performance-degrading effects of frequent cache misses commonly associated with hardware caches. This local memory hierarchy is very important, especially in conditions when execution rate of a SPE is significantly higher than the rate at which data can be fetched into the SPE.

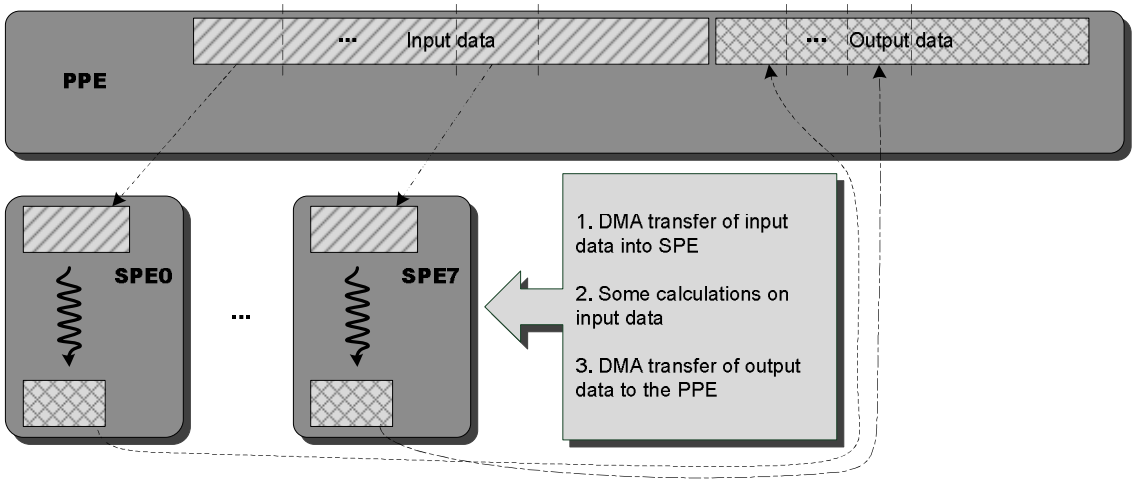

Figure 9: Each SPE independently computes a part on the input dataset. The results are assembled on the main processing unit, PPE

In the Cell/B.E., the peak SPE-to-main-memory bandwidth is 25.6 Gb/s, which is one of the highest memory-to-CPU bandwidth designs. With double buffering to overlap communication and computation, the Cell/B.E. has a computeï toï memory bandwidth ratio of 8 ops per byte of data fetched from the system memory (204.8 Gflops/25.6 Gbytes).

To compile code for the Cell/B.E., one can use either the IBM xlc or the GNU gcc (ppu-gcc and spu-gcc) compilers. In addition to the original thread based programming model for developing Cell/B.E. applications, developers have ported several other programming models to this architecture, including OpenMP, message-passing interface, Google's MapReduce, the data-parallel RapidMind model, and the data driven Cell Superscalar (CellSs) model [5]. Cell/B.E.-based systems are also the top energy-efficient systems on the green500 list (www.green500.org).

III.B.4 ClearSpeed

The ClearSpeed Technology focuses on alleviating power, heat, and density challenges of high performance systems. Their latest processor CSX700 is high performance solution with very low power consumption, mainly due to low clock frequency and advanced clock gating [7]. It is at the heart of ClearSpeed's e720 accelerator card that brings the power of CleerSpeed's processor to the HP blade server. Thanks to compact size of card and low power consumption of processor it can fit into type II mezzanine slot without requiring any additional cooling or power supply [8]. Low power consumption means lower heating and has positive effects on the reliability of system. Further reliability improvement is done by inclusion of ECC bits with each memory.

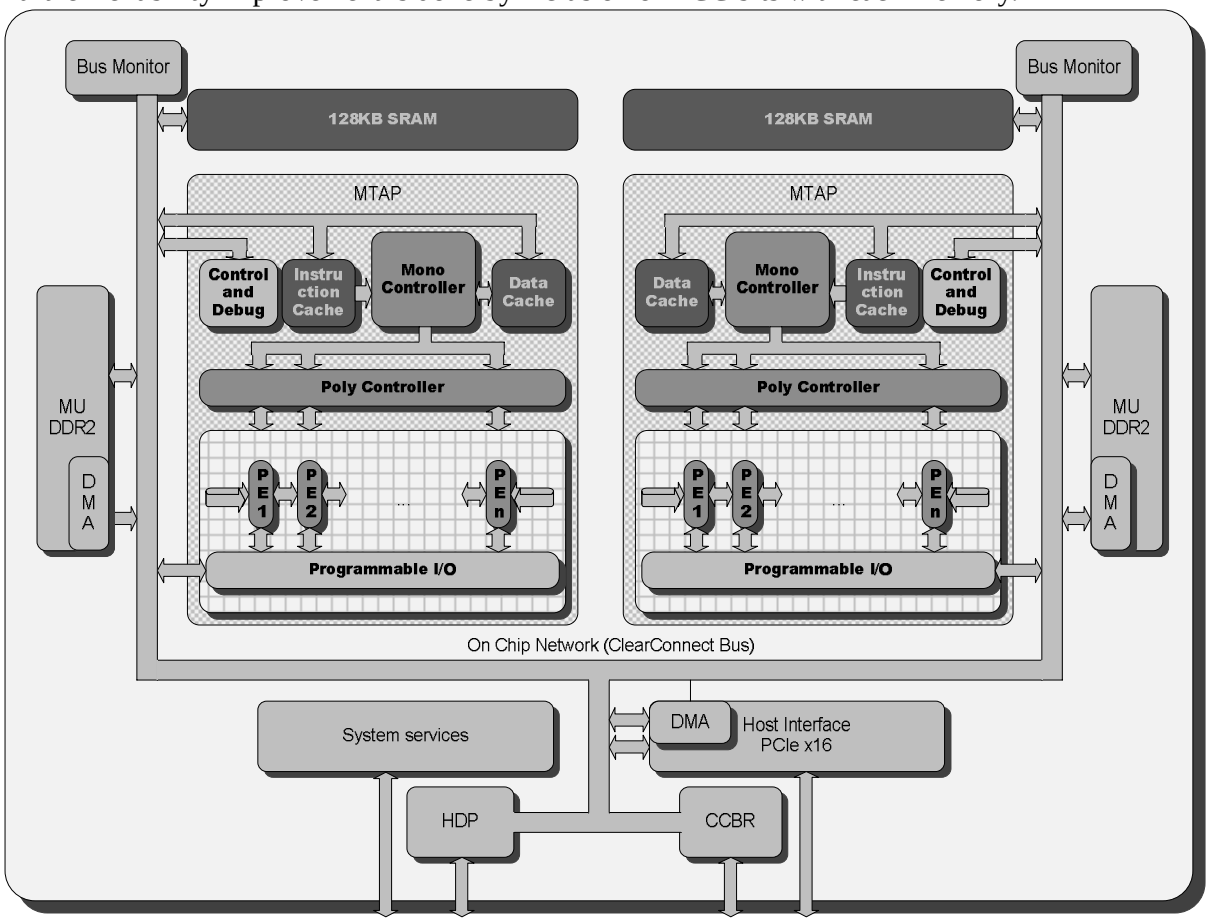

Figure 10: Structure of the ClearSpeed's CSX700 processor

Figure 10 shows structure of the processor. It is a dual core SIMD architecture, where each core is multithreaded array processor (MTAP). Along with cores, there are two DDR2 DRAM interfaces that supports 8GB of RAM each, two on-chip 128KB SRAM memory, PCIe x16 host interface, and

system services block. Inside of core, there are: One main multithreaded processor, and one SIMD processing unit. Main processor supports 8 hardware threads and for synchronization of threads has 128 8bit semaphores. Processor has two caches, instruction and data cache. Both caches are

organized as 4-way set-associative caches. Instruction cache has 512 lines of 4 instructions, while data cache has 256 lines of 16B each. Execution unit has ALU, double precision FPU, and several register files, 128B each.

The SIMD processing unit has 96 processing elements (PE) that give a lot of processing power to the SlearSpeedôs processor. Poly controller accepts instructions from the main processor, and organizes execution on PEs. Each PE is equipped with: 128B register file, one FPU for single and double precision operations with dual issue pipelined add and multiply, a 6KB of PEôs SRAM, 16-bit MAC with 64 bit accumulator, and support for integer and floating point divide and square root operations. The PEôs SRAMs give high memory bandwidth for the purpose of parallel execution on PEs. The PEs have a possibility to communicate directly to the memory or IO through configurable programmed IO channel. Chanel is 128bit wide, can transfer 8, 16, 32 or 64 bit per PE, and can use hardware semaphore for synchronization.

The code is fetched by the main processor, and depending on the instruction type, executed by main processor or by the SIMD processing unit.

Accelerator can be used in two ways. The first one is plug and play where ClearSpeed supported library intercepts call and determine whether that call can be accelerated or not. If the answer to the previous question is positive, library makes further steps in order to utilize accelerator without any programmerôs help. The other one is native code development where programmer recognizes parts of code that can be speeded up and using SlearSpeed SDK translate them into the code for accelerator. The SDK includes: (a) A suite of tools including an industry standard source-level debugger based on gdb and an ANSI C-based cross compiler for ClearSpeedôs CSX family of processors, (b) An extensive set of standard C libraries based on the newlib open source library together with a set of libraries to support architecture-specific features, (c) The ClearSpeed Vector Math Library and ClearSpeed Random Number Generator Library, and (d) Documentation for the software development tools and languages.

III.B.5. Maxeler Dataflow Engines

Maxeler Technologies in Palo Alto and London is a fast expanding HPC company with roots at Stanford, Bell Labs, US, and Imperial College, London, UK.

Maxeler Technologies' core competence is in delivering substantially increased performance for HPC applications through the rewriting of applications for Dataflow Engines. By mapping compute-intensive algorithms directly into parallel hardware, tightly coupled to a conventional CPU through a high-speed I/O bus, complete applications can be accelerated by orders of magnitude over conventional CPU implementations. By exploiting massive parallelism at the bit-level, Maxeler MaxNode solutions deliver performance far in excess of CPUs at approximately a tenth of the clock frequency and power consumption.

Figure 11 sketches the architecture of a Maxeler hardware acceleration system which equips one or more Dataflow Engines attached to a set of memories and connected to a host CPU via PCI Express channels [15 and 25]. MaxRing interconnects (not shown in Figure 9) establish high bandwidth communication channels between the Dataflow Engines on the accelerator. Accelerating an application involves looking at the entire application, adapting program structure and partitioning the execution flow and data layout between CPUs and accelerators. The program of the Dataflow Engines comprises arithmetic data-paths for the computations (the kernels) and modules orchestrating the data I/O for these kernels (the manager).

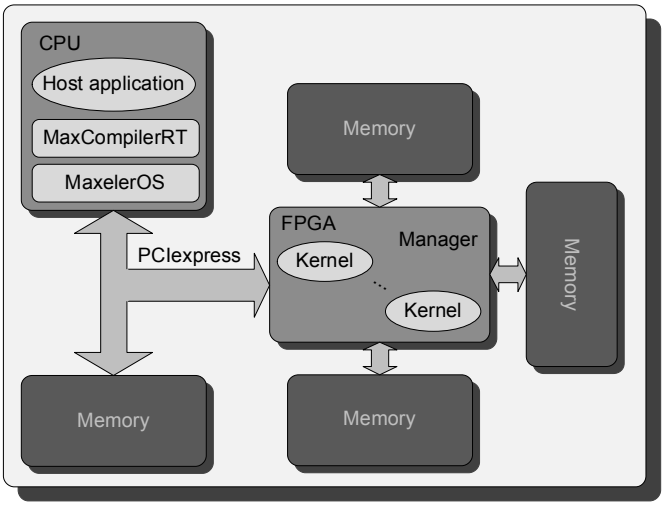

Figure 11: The Maxeler accelerator architecture

The Maxeler Kernel Compiler generates the dataflow engines, and as such the program describes computations structurally (computing in space) rather than specifying a sequence of processor instructions (computing in time). A kernel is a streaming core with a data flow described by a unidirectional graph without cycles.

For example, kernel code for a 3-point moving average over N values with 2-point averages at the boundaries:

$$y_i = \begin{cases} (x_i + x_{i+1})/2 & if i = 0 \\ (x_{i-1} + x_i)/2 & if i = N-1 \\ (x_{i-1} + x_i + x_{i+1})/2 & otherwise \end{cases}$$

Figure 12: Code for moving average over N values

Figure 13 depicts the kernel graph for the moving average which splits into a data part (right-hand side) and a control part (left-hand side). The Kernel Compiler is effectively a Java software library and as such, kernel graphs are created by writing a Java program and executing it.

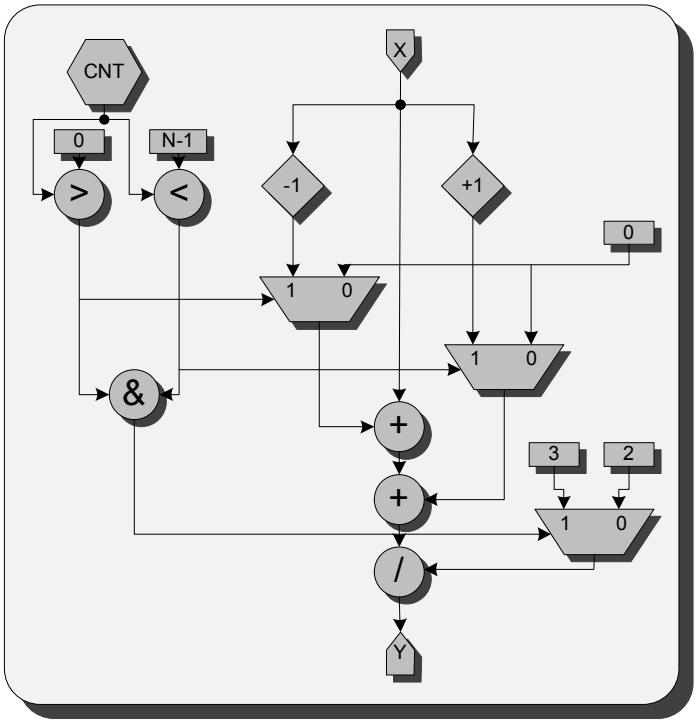

Figure 13: Kernel graph for the moving average example

Kernel graphs are directly mapped to hardware and then data streams through the arithmetic nodes. Efficient streaming kernels strongly emphasize the regularity of the data flow, making the actual computations look like a side-effect of streaming. Such streaming kernels lend themselves to deeply pipelined dataflow implementations which are key to achieving high performance in custom hardware.

III.B.6. SGI's Reconfigurable Application Specific Computing (RASC)

SGI was founded as a spinoff of the Stanford University's research in accelerating a specific application, three dimensional graphics. The SGI pioneered acceleration of graphics through hardware setting records and providing technological capabilities that were impossible without specialized computational elements [33].

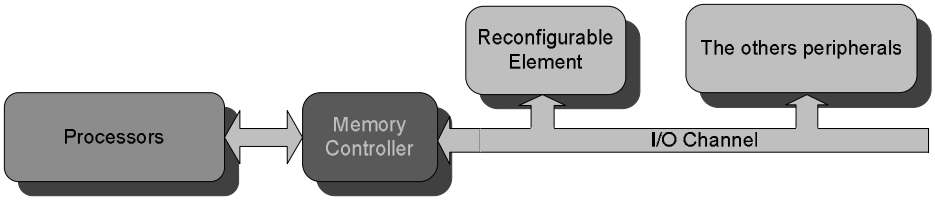

Figure 14: RASC Architecture

Altix is a family of SGIôs SMP (Symmetric Multi Processing) solutions. Its distinguishing feature is that each processor has both fast access local memory and slower access to local memories of other processors in the system. Data exchange between processors is achieved thanks to NUMALink bus. The NUMALink bus enables data exchange between processors to be relatively fast. The NUMALink interconnect is a hierarchical system bus. It allows for global addressing and scalability of SMP systems. Maximum NUMALink data transfer is 6.4 GB/s. The integral component of the Altix system can be the Reconfigurable Application Specific Computing (RASC) module. It is the SGIôs technology that is enabling users to develop application specific hardware using reconfigurable logic elements.

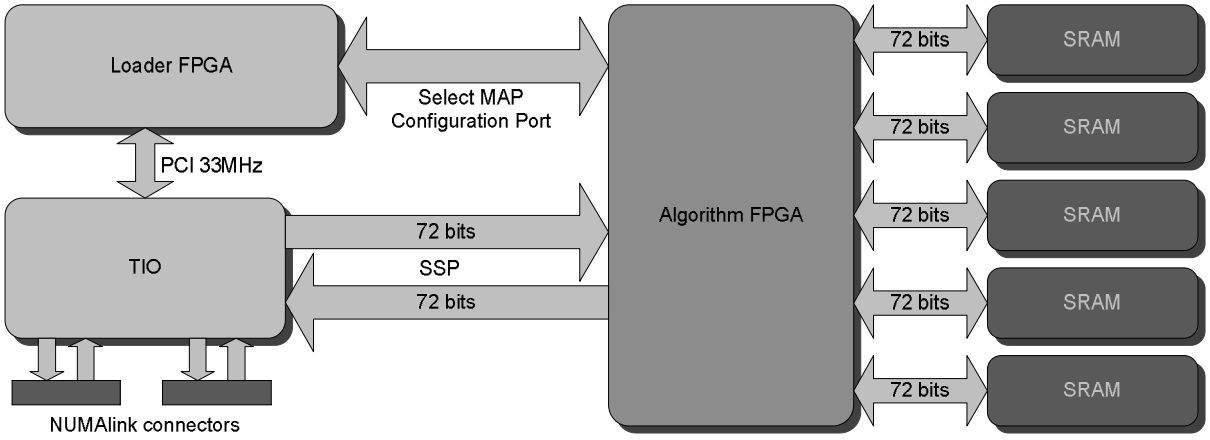

Figure 15: The RASC module structure

The SGI RASC is tightly integrated with NUMALink. From the hardware perspective, FPGA is no longer a co-processor mode in this model. As shown in Figure 14, with the NUMALink FPGA, it has access to global shared memory and there is no need to load and unload data. The RASC is coupled with two Virtex4LX200 FPGA chips. Each one offers 200k of reconfigurable logic cells. Additionally there are two blocks of 64 MB QDR RAM memory. This memory acts like a second level cache for FPGA. The first level cache is implemented inside the Virtex4LX200 structure and is called BlockRAM. Bidirectional data interface implemented for FPGA has an 128-bit width and is clocked with frequency of 200 MHz [22].

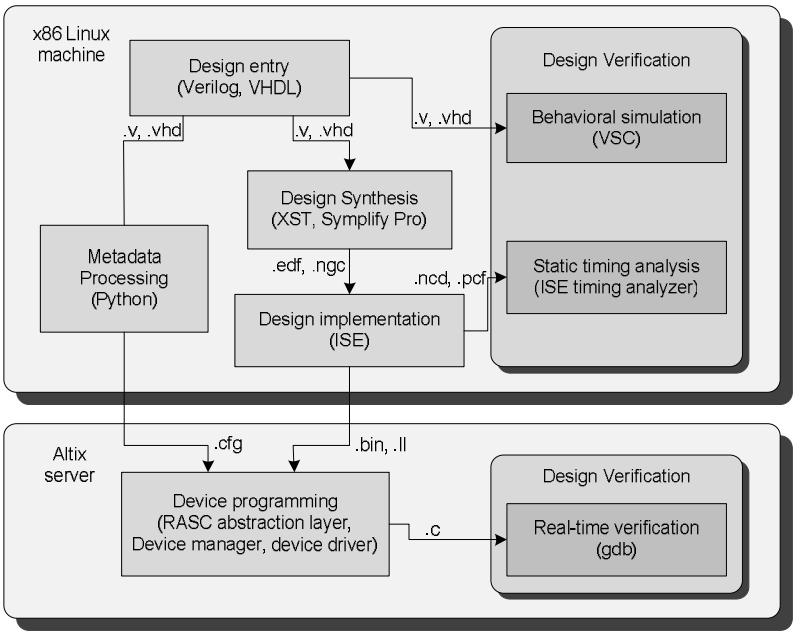

Figure 16: The RASC FPGA programming process

The RASC hardware module is based on an application-specific integrated circuit (ASIC) called TIO. The TIO attaches to the Altix system NUMAlink interconnect directly, instead of being driven from a compute node using the XIO channel. One TIO supports two PCI-X busses, an AGP-8X bus,

and the Scalable System Port (SSP) port that is used to connect the Field Programmable Gate Array (FPGA) to the rest of the Altix system for the RASC program. The RASC module contains a card with the co-processor (COP) FPGA device, as shown in Figure 15.

Summary of the SGI approach to FPGA programming is as follows (see Figure 16): Write an application in the C programming language for system microprocessor; Identify computation intense routine(s); Generate a bitstream using Core Services and language of choice; Replace routines with RASC abstraction layer (rasclib) calls that support both a C and Fortran90 interface; Run your application and debug with GDB.

III.B.7. Convey's Reconfigurable Coprocessor

The Convey hybrid-core is the FPGA based accelerator board coupled to the processor as a coprocessor [10]. This system, appeared in 2008, brings a new programming model, where personalities are used as an extension of the instruction set of the processor. There are predefined sets of personalities for problems in several industries, including: Oil and gas, computer-aided engineering (CAE), bioinformatics, and financial services.

A structure of the system is shown on Figure 17. The main component is a board with four Xilinx

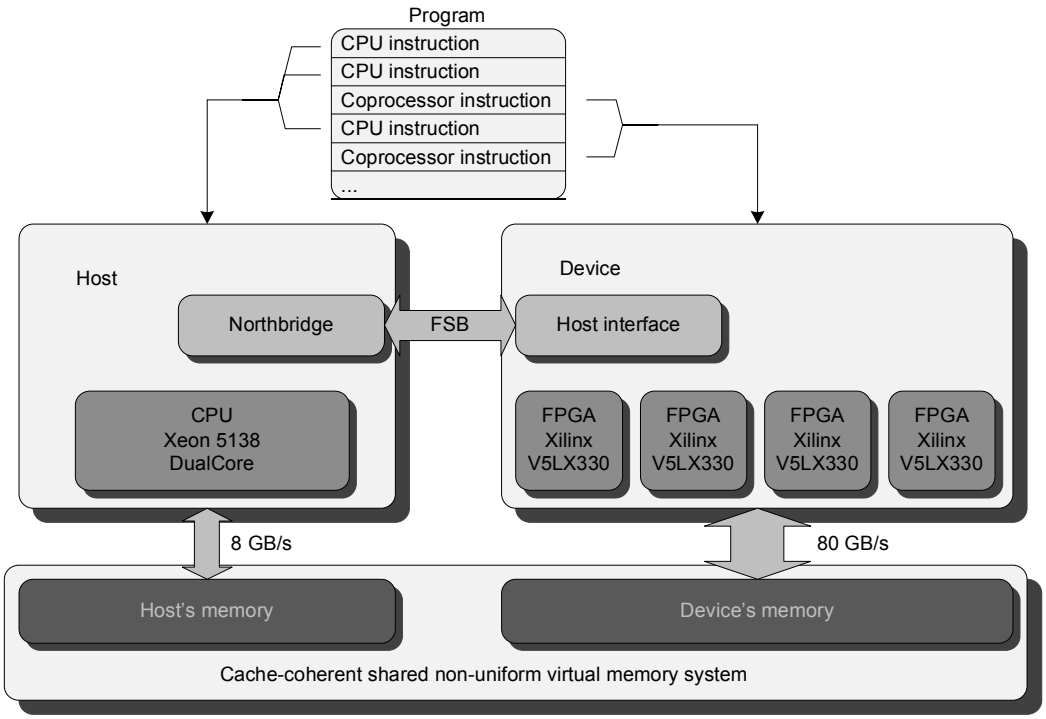

Figure 17: Structure of system with Convey accelerator card.

Virtex 5 LX330 chips intended for calculations. There are two additional FPGAs that implements interface to the host. The board is tightly coupled with processor, and has access to the processorôs RAM memory. The board also has its local memory, with significantly greater throughput to it. Across the whole system a cache-coherent shared virtual memory system is provided that allows to access data in the CPUôs memory and in the accelerator device memory. Though, logically, memory is seen as uniform, because different memories in use, system is implemented as ccNUMA[2]. For the best performance, programmer have to make appropriate data placement. FPGAs have on chip memory, known as BlockRAM, but this memory is not directly visible to the programmer.

The programming model of this system offers solution to overcome complexity of programming FPGA based accelerators [11]. The solution is in a form of the predesigned hardware, as Convey calls it, personality. Operations that are often in the domain of interest, and that are time-consuming, are implemented in the hardware. According to a particular program that will be executed, FPGAs are configured with personalities. Which personalities will be used is decided in compile time by setting compiler flags. Compiler is in some degree able to automatically recognize parts of code that can be accelerated with specified personalities, and to compile those parts, so they can be also executed on FPGAs (generated code can be executed on machine without FPGAs). Because of limited abilities of compiler, it is possible that programmer annotates source code written in C, C++, or Fortran with pragma directives. All of this is possible because compiler gives to the programmer unique view of all memory, systemôs one and acceleratorôs one. From programmerôs point of view, this significantly simplifies programming, but because of NUMA nature of the system it can significantly reduce performance. To deal with this drawback, programmer has to think where to put which data, and to specify that in a form of pragma directives. Pragma directives are also used for compiler optimizations.

Code intended for execution on FPGA has some other limitations. It cannot call non-coprocessor code including system calls, and it cannot do input/output operations.

Compiler comes with some predefined personalities that are single precision and double precision floating point operations, defining a soft-core for vector coprocessor. For someone who needs more, there is Conveyôs personality development kit for developing new customized personalities, but that requires significant additional knowledge and effort.

III.B.8. Conclusion About Existing Solutions

We have presented several commercially successful solutions for high performance computing. For each one we have given an overview of hardware and software. This section gives a summary of selected architectures, future trends from our point of view, and a conclusion.

III.B.8.a. Architectures

A performance comparison is given in the table below [1, 2, 5, 6, 7, 16, 21, 36, and 37]. When interpreting the data presented in Table 1, keep in mind that not all solutions belong to the same generation of technology. Year of issue can be used as an orientation to which generation a particular solution belongs to. This is important because the generation has a significant impact on the speed of a system, hiding the true relationship between two different architectures. Further on, most of the quoted sources present peak performances as the major (often times the only one) performance data, while the reality says the only useful performance measures are those related to sustained performances. Some studies show that the sustained performance is usually several times lower than the peak performance. One such study shows that presented GPUs with peak performance measured in TFLOPS are able to sustain only several hundreds of GFLOPS [37]. Also, the shown differences between CPU and GPU performances are in question [24]. The later one shows that Convey HC-1 performs less than 10 GFLOPS (80 GFLOPs in peak performance) on most of the used kernels (parts of bigger real program that are time consuming, e.g. dense matrix-vector multiplication), and that real bandwidth to the memory is less than half of the presented peak bandwidth [2]. One respected exception is the Maxeler machine, and its data related to the sustained performance in the context of oil drilling.

Also, the important parameters for assessing performance potential for parallel algorithms, besides peak performance are also a bandwidth of shared memory access or interconnection network speed.

Table 1: Performance comparison

Intel Nehalem E5520 Quad-core CPU

Computation Capacity				Memory Capacity			
				L1 cache		Memory	
# cores	Clock frequency	Peak performance	Power	Size	Bandwidth	Size	Bandwidth
4	2.27 GHz	36 GFLOPs	80W	128 kB	291 GB/s	not limited	25 GB/s

Cell/B.E

Computation Capacity				Memory Capacity					
				CPU cache		Local store			
# cores	Clock frequency	Peak performance	Power	Size	Bandwidth	Size	Bandwidth	Memory size	Bandwidth
1+8 hetero	3.2 GHz	230.4 GFLOPs	135W	512kB	44 GB/s	8*256KB	204.8 GB/s	16 GB	25 GB/s

ClearSpeed CSX700

Computation Capacity				Memory Capacity					
				CPU cache	Local store		Memory		Bandwidth to host
# cores	Clock frequency	Peak performance	Power	Size	Size	Bandwidth	Size	Bandwidth	
2+192 hetero	250 MHz	96 GFLOPS	11.4W	24KB	2*128KB	192 GB/s	2*8 GB	2*4 GB/s	4 GB/s

SGI RASC Accelerator board (2 x Virtex4 LX200) max 120W

Computation Capacity						Memory Capacity				
						Block RAMs		On board memory		Bandwidth to host
# LUTs	# FFs	# DSP48E	Clock frequency	Peak performance	Power	#	Size	Size	Bandwidth	
200448 x2	200448 x2	96 x2	200 MHz	47 GFLOPs	120W	336	0.7 MB	40 MB	16 GB/s	6.4 GB/s

Maxeler Max2 FPGA Acceleration Card

Computation Capacity						Memory Capacity					
						Block RAMs			On board memory		Bandwidth to host
# LUTs	# FFs	# DSP48Es	Clock frequency	Peak performance	Power	#	Size	Bandwidth	Size	Bandwidth	
414720	414720	384	150MHz	116 GFLOPs	55W	648	2.8 MB	1519 GB/s	12GB	28GB/s	4GB/s

Convey coprocessor HC-1

Computation Capacity						Memory Capacity					
						Block RAMs			On board memory		bandwidth to host
# LUTs	# FFs	# DSP48E	clock frequency	peak performance	Power	#	Size	bandwidth	size	bandwidth	
4* 207360	4* 207360	4* 192	n/a	80 GFLOPs	100W	4*28 8	4*1.25 MB	n/a	8 GB	80 GB/s	1066 MT/s

NVidia GTX580

Computation Capacity					Memory Capacity				
					shared memory		on board memory		bandwidth to host
# Multiprocessors	# cores	clock frequency	peak performance	Power	Size	bandwidth	size	bandwidth	
16	512	1.54 GHz	1.58 TFLOPs	244W	768 KB	N/A	6 GB	192.4 GB/s	8 GB/s

AMD ATI HD5870

Computation Capacity					Memory Capacity				
					shared memory		on board memory		bandwidth to host
# Multiprocessors	# cores	clock frequency	peak performance	Power	size	Bandwidth	size	bandwidth	
20	1600	850 MHz	2.72 TFLOPs	188W	640 KB	2176 GB/s	6 GB	153.6 GB/s	8 GB/s

Presented architectures have their own strengths and weaknesses, which make differences between them. The main problem of all these solutions is difficulty of programming them, especially keeping in mind that programs have to be tuned for the target architecture, in order to take the best of it. The level of difficulty of programming these systems, and performance that will be achieved differs from application to application.

Based on data published by manufacturers, the best performing architectures for single precision floating point operations are GPUs. Often, they achieve several times better performance than the others. Prerequisite for this performance benefit is that the same operation can be done on a large amount of data in parallel. Algorithms with a lot of synchronization and communication are not suitable for these architectures, especially if the communication pattern is irregular [24]. GPUs are also a preferred solution, if the decision is based on the price and performance per watt ratio.

ClearSpeed, made exclusively for acceleration of high performance calculations, seems very similar to the GPU, except that it has only two SIMD processors several times wider than the ones in the GPU. It is one order of magnitude slower, mainly due to a low working frequency. From the other side, lower frequency means a significantly lower power consumption and better performance per watt ratio.

Despite the fact that CBEA has appeared five years ago, it is still comparable to the state-of-the art CPUs. It presents a very flexible architecture with eight independent cores. Each core can execute different program. Fast communication and synchronization, with over 200 GB/s bandwidth of the interconnect bus, makes CBEA a very flexible solution. The price paid is more difficult programming, compared to GPUs. Double precision calculations can also be done very efficiently.

The performance of FPGA based solutions cannot be easily measured using the number of floating point operations done per unit of time, as FPGAs do not have predefined floating point units. Implementing floating point requires use of a significant amount of available logic, and generated logic is much slower than that in CPUs and GPUs. The FPGA based solutions are much better suited for algorithms based on fixed point, integer, or logical operations. For those applications, FPGA can significantly outperform competitions. One more benefit of using FPGAs is the ability to implement custom precision floating point, completely independent of the IEEE-754 standard. For Maxeler FPGA solutions in particular, peak performance potential of the chip is not a good guide to overall achieved performance, since the inherent flexibility of the architecture often enables good solutions to be devised for a wide range of applications where more rigid solutions (even with higher peak performance) struggle.

III.B.8.b Programming Considerations

There is a widespread opinion that the major obstacle for use of heterogeneous and reconfigurable architectures is a lack of programming environment on higher levels of abstraction. Such an environment will take some burden away from thinking about every detail of the target architecture from programmer and will provide them with a much faster development cycle. It will also simplify migration of existing software to these new architectures.

The main obstacle in offering a high-level programming (e.g., C and Java), lies in the semantic gap between this model and the explicitly concurrent models used to program hardware devices. Common hardware description languages such as VHDL or Verilog use an execution model based on Communicating Sequential Processes (CSP)[15] and thus far detached from the imperative models.

Approaches to solve a semantic gap problem cover a wide spectrum of solutions, ranging from the easier approach where the input language already offers a concurrency execution model close to the hardware CSP model to the harder approach of automatic uncovering of concurrency from traditional imperative languages [31]. Compilation of programs from the imperative paradigm to hardware has therefore to bridge this semantic gap by automatically extracting as much concurrency as possible. A popular alternative approach is to rely on library implementations where the notions of concurrent

execution have been crystallized by library developers and openly publicized in application programmer interfaces (APIs). Programming systems such as Maxeler MaxCompiler [25] address this by allowing the programmer to describe their application in a high level language (Java) but in an explicitly parallel way.

The popularity of MATLAB, as a domain-specific language for image/signal processing and control applications points to a great potential for adoption of this approach in specific application domains. Current FPGA compilation flows are too rigid as programmers must endure long compilation cycles, for the definition of a suitable reconfigurable computing implementation. A possible avenue for mitigating the issues related to long compilation cycles would include the use of dynamic Just-In-Time (JIT) and incremental compilation techniques. A first, quick translation to possible not very efficient mappings, using only the regions of the input code exercised, would allow the execution to proceed as quickly as possible. As the execution would progress, a run-time system would trigger a recompilation of the more frequently exercised structures for recompilation, this time with the benefit of the knowledge of key specific program values.

IV. The Trend, Promising Solutions, and Open Questions for Future Research

In our opinion, heterogeneous computing has a great future. As we have seen from the previous text, there are several obstacles that stay on the way to a more widespread use of heterogeneous computing. One of them is migration of existing software to these new architectures. One approach to this problem is to offer libraries that will be used by old software to get benefits from these new architectures.

Looking at the further directions of GPU technology development, it seems that there will be included some new hardware resources, e.g. new high speed reconfigurable interconnections like those in FPGAs. Software environment is also improved: The CUDA 4.0 offers a shared address space, where the whole memory can be accessed by both, CPU and GPU. Earlier versions were limited to the use of a subset of the C language, but the newer ones support full C++.

The future of the CBEA cannot be seen clearly. Although the CBEA has independent cores and use local memory together with DMA, its programming complexity makes this architecture less attractive. However, in our opinion, as power becomes one of the most limiting factors, some features from CBEA will be seen again in new generation of processors [30].

The FPGA based solutions represent a very promising approach. The fast growth rate in the capacity of FPGAs, and at the same time the reduction of price and power consumption are observed during the last several years. It is widespread opinion that FPGAs will be further improved by increasing of the clock frequency and by adding new hardware resources, such as floating point support. Adding new resources will make FPGAs suitable for much wider range of applications. Because of the need for hardware design, these solutions are more difficult to program and to achieve good performance. Programming issue comprises: Methodology of generation bitstream for the FPGA, support for debugging application, and interface between application and system. Performance issue includes optimizations of data movement (partitioning) between microprocessors and FPGAs, and partitioning between multiple CPU+FPGA pairs, driving the scalability of the system topology. There are several approaches that offer solution for these problems. The best performing approach is Maxeler dataflow programming model which offers modeling of dataflow machine on high level of abstraction instead of directly designing hardware. Some other solutions offer a more preferred programming model where programmer sees only the library calls, but with one order of magnitude lower achieved performance.

V. Conclusion

For certain data-intensive applications, more cores do not mean better performance, according to Sandiaôs simulation [26]: òAfter about 8 cores, thereôs no improvement, and further it even decreasesò. Therefore, it is difficult to envision efficient use of hundreds of traditional CPU cores [32]. The use of hundreds of accelerator cores in conjunction with a handful of traditional CPU cores, on the other hand, appears to be a sustainable roadmap.

Currently, there is no one-size-fits-all approach for different computing domains, including general purpose computing. The seven presented architectures are intended for different problem domains, and none of them seem to be universal.

The GPU provides a mass-appeal via highly parallel and accessible accelerator technology where communication and synchronization are avoided. The CBEA is a very flexible architecture which gives to the programmer freedom of eight independent cores with high bandwidth for inter-core synchronization and communication. The last ones, FPGA based solutions are very suitable for applications relying on very large datasets and complex computations. Using appropriate approach, some of them achieve excellent performance.

VI. References

[1] AMD ATI. 2011. ATI RadeonÊ HD 5870 Graphics Specification. Online: http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/hd-5870/Pages/ati-radeon-hd-5870-overview.aspx#2

[2] AUGUSTIN, W., WEISS, J., HEUVELINE, V. 2010. Convey HC-1 Hybrid Core Computer ï The Potential of FPGAs in Numerical Simulation. No. 2010-07. Online: http://www.emcl.kit.edu/preprints/emcl-preprint-2010-07.pdf

[3] BARROSO, L., GHARACHORLOO, K., McNAMARA, R., NOWATZYK, A., QADEER, S., SANO, B., SMITH, S., STETS, R., AND VERGHESE, B. 2000. Piranha: A Scalable Architecture Based on Single-chip Multiprocessing. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, British Columbia, Canada, June 2000, BERENBAUM, A., EMER, J., ACM New York, NY, USA, 282-293.

[4] BOBDA, C. 2007. Introduction to reconfigurable computing - architectures, algorithms, and applications. Springer, Dordrecht, The Netherlands.

[5] BRODTKORB, A., DYKEN, C., HAGEN, T., HJELMERVIK, J., AND STORAASLI, O. 2010. State-of-the-art in Heterogeneous Computing. Scientiýc Programming , 18, 1, 1ï 33.

[6] CHEN, T., RAGHAVAN, R., DALE, J., IWATA, E. 2007. Cell Broadband Engine Architecture and its first implementationï A performance view. IBM Journal of Research and Development. 51, 5, 559ï 572.

[7] CLEARSPEED TECHNOLOGY Ltd. 2011. CSX700 Floating Point Processor Datasheet. Online: http://www.clearspeed.com/products/documents/CSX700_Datasheet_Rev1E.pdf

[8] CLEARSPEED TECHNOLOGY Ltd. 2010. Advance e720 Accelerator Card User Guide. Online: http://support.clearspeed.com/resources/documentation/Advance_e720_User_Guide_Rev1.D.pdf

[9] COMPTON, K., HAUCK, S., 2002. Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys, 34, 2, 171ï 210.

[10] CONVEY COMPUTER CORPORATION. 2010. Conveyôs hybrid-core technology: the HC-1 and the HC-1ex. Online: http://www.conveycomputer.com/Resources/Convey_HC1_Family.pdf

[11] CONVEY COMPUTER CORPORATION. 2009. Compilers for Convey Hybrid-Core Computers. Online: http://www.conveycomputer.com/Resources/Compiler%20Data%20Sheet.pdf

[12] EGGERS, S.J., EMER, J.S., LEVY, H.M., LO, J.L., STAMM, R.L. AND TULLSEN, D.M. 1997. Simultaneous Multithreading: A Platform for Next-generation Processors. IEEE Micro. 17, 5.

[13] EKMECIC, I., TARTALJA, I., MILUTINOVIC, V. 1996. A Survey of Heterogeneous Computing: Concepts and Systems. In Proceedings of the IEEE, 84, 8, 1127-1144.

[14] EKMECIC, I., TARTALJA, I., MILUTINOVIC, V. 1995. A Taxonomy of Heterogeneous Computing. IEEE Computer. 28, 12, 68-70.

[15] FU, H., OSBORNE, W., CLAPP, R., MENCER, O., AND LUK, W. 2009. Accelerating Seismic Computations Using Customized Number Representations on FPGAs. EURASIP Journal on Embedded Systems. 2009, Article ID 382983, 2009.

[16] FU, H. 2010. Accelerating Scientific Computing Through GPUs and FPGAs. Stanford Center for Computational Earth & Environmental Science (CEES) Workshop. Online: http://cees.stanford.edu/docs/CEESWorkshop8-HFu.pdf

[17] GEPPERT, L. 2005. Sunôs Big Splash. IEEE Spectrum Magazine, 42, 1, 56ï 60.

[18] GULATI, M., AND BAGHERZADEH, N. 1996. Performance Study of a Multithreaded Superscalar Microprocessor. In *Proceedings of the 2nd International Symposium on High Performance Computer Architecture*, San Jose, CA, USA, February 1996, IEEE Computer Society. Technical Committee on Computer Architecture, Eds. IEEE Computer Society Press, Washington, DC, USA, 291ï 301.

[19] HAMMOND, L., NAYFEH, B. A., AND OLUKOTUN, K. 2000. The Hydra Chip. IEEE MICRO Magazine. 20, 2, 71ï 83.

[20] HIRATA, H., KIMURA, K., NAGAMINE, S., MOCHIZUKI, Y., NISHIMURA, A., NAKASE, Y., AND NISHIZAWA, T. 1992. An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, Queensland, Australia, May 1992, GOTTLIEB, A., ACM New York, NY, USA, 136ï 145.

[21] ILSCHE, T., JUCKELAND, G. 2008. First experiences with SGI RASC. Technical University Dresden. Online: http://www.juckeland.net/wp-content/uploads/sgirep.pdf

[22] JAMRO, E., JANISZEWSKI, M., MACHACZEK, K., RUSSEK, P., WIATR, K., WIELGOSZ, M. 2008. Computation acceleration on SGI RASC: FPGA based reconfigurable computing hardware. Computer Science. 9, 21-34.

[23] KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. The Hydra Chip. IEEE MICRO Magazine, 25, 2, 21ï 29.

[24] LEE, V.W.,KIM, C., CHHUGANI, J., DEISHER, M., KIM, D., NGUYEN, A.D., SATISH, N. SMELYANSKIY, M., CHENNUPATY, S.,HAMMARLUND, P.,SINGHAL, R., DUBEY, P. 2010. Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. In Proceedings of the 37th annual international symposium on Computer architecture, Saint-Malo, France, June 2010, A, SEZNEC, U. WEISER, and R. RONEN, ACM New York, NY, USA. 38, 3, 451-460.

[25] MAXELER TECHNOLOGIES. 2011. MaxCompiler White Paper. Online: http://www.maxeler.com/content/briefings/MaxelerWhitePaperMaxCompiler.pdf

[26] MOORE, S.K. 2008. Multicore is Bad News for Supercomputers. IEEE Spectrum. 45, 11, 15ï 15.

[27] MUNSHI, A. 2008. OpenCL. Online: http://s08.idav.ucdavis.edu/munshi-opencl.pdf

[28] NVIDIA. 2009. NVIDIAôs Next Generation CUDA Compute Architecture: FERMI. Online: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf

[29] OLUKOTUN, K., AND HAMMOND, L. 2005. The future of microprocessors. ACM Queue. 3, 7, 26ï 29.

[30] OUDA, I. AND SCHLEUPEN, K. 2008. Application Note: FPGA to IBM Power Processor Interface Setup. IBM Research Report RC24596, July 2008. Online: http://domino.watson.ibm.com/library/CyberDig.nsf/1e4115aea78b6e7c85256b360066f0d4/0019083255e3732c8525747a0068a14d?OpenDocument&Highlight=0,Schleupen

[31] PAPAKONSTANTINOU, A., GURURAJ, K., STRATTON, J., CHEN, D., CONG, J., HWU, W. 2009. FCUDA: Enabling Efficient Compilation of CUDA Kernels onto FPGAs. In Proceedings of the IEEE 7th Symposium on Application Specific Processors, San Francisco, California, July 2009, Institute of Electrical and Electronics Engineers. 35-42.

[32] PEDRETTI, K., KELLY, S., LEVENHAGEN, M. 2008. Summary of Multi-Core Hardware and Programming Model Investigations. Sandia Technical Report, SAND2008-3205, May 2008. Online: http://www.cs.sandia.gov/~ktpedre/papers/multicore-tech.pdf

[33] SGI. 2008. Reconfigurable Application-Specific Computing User's Guide. SGI, Inc, doc. No 007-4718-007, published 2008. Online: http://techpubs.sgi.com/library/tpl/cgi-bin/summary.cgi?coll=linux&db=bks&docnumber=007-4718-007

[34] SHI, G., KINDRATENKO, V., PRATAS, F., TRANCOSO, P., AND GSCHWIND, M. 2010. Application Acceleration with the Cell Broadband Engine. Computing in Science & Engineering. 12, 1, 76ï 81.

[35] SONY CORPORATION. 2006. Cell Broadband Engine Architecture. Sony Corporation. Online: http://cell.scei.co.jp/pdf/CBE_Architecture_v101.pdf

[36] STRENSKI, D. 2007. FPGA Floating Point Performance ï a pencil and paper evaluation, HPCwire. Online: http://www.hpcwire.com/hpc/1195762.html

[37] ZHANG, Y., PENGL, L., LI, B., PEIR, J., CHEN, J. 2011. Architecture Comparisons between Nvidia and ATI GPUs: Computation Parallelism and Data Communicationsò. Online: http://www.ece.lsu.edu/lpeng/papers/iiswc11.pdf