Computer Science • Vol. 10 • 2009

Dawid Kuna[*], Ernest Jamro[*,**], Paweł Russek[*,**], Kazimierz Wiatr[*,**]

USING STANDARD HARDWARE ACCELERATORS TO DECREASE COMPUTATION TIMES IN SCIENTIFIC APPLICATIONS

Nowadays, general-purpose processors are being used in scientific computing. However, when high computational throughput is needed, it's worth to think it over if dedicated hardware solutions would be more efficient, either in terms of performance (or performance to price ratio), or in terms of power efficiency, or both. This paper describes them briefly and compares to contemporary general-purpose processors' architecture.

Keywords: *general-purpose processors, standard accelerators, computation accelerators, dedicated architectures, custom computing, GPGPU, Cell, ClearSpeed*

UŻYCIE STANDARDOWYCH AKCELERATORÓW SPRZĘTOWYCH DO SKRÓCENIA CZASU OBLICZEŃ NAUKOWYCH

Współcześnie w obliczeniach naukowych stosuje się procesory ogólnego przeznaczenia. Gdy potrzebna jest duża przepustowość obliczeniowa, warto zastanowić się, czy dedykowane rozwiązania sprzętowe nie okazałyby się efektywniejsze po względem wydajności (lub stosunku wydajności do ceny), zużycia energii bądź obu czynników jednocześnie. Artykuł opisuje pobieżnie dedykowane rozwiązania sprzętowe i porównuje ze współczesnymi architekturami procesorów ogólnego przeznaczenia.

Słowa kluczowe: *procesory ogólnego przeznaczenia, standardowe akceleratory, akceleratory obliczeń, architektury dedykowane, GPGPU, Cell, ClearSpeed*

1. Introduction

High-performance computing (HPC) has migrated over time from expensive and dedicated devices to cheap general-purpose components. The former were efficient and designed specially with high overall performance in mind. Narrow application space and low-volume production as well as expensive components made their price a significant disadvantage. In contrary, commodity hardware components, due to their

[*] Academic Computer Centre CYFRONET AGH, Krakow, Poland, dawid.kuna@cyf-kr.edu.pl
[**] Department of Electronics AGH, University of Science and Technology, Krakow, Poland, jamro@agh.edu.pl, russek@agh.edu.pl, wiatr@uci.agh.edu.pl

high-volume production, were both generally available and affordable. As they slowly adapted some ideas from other computers (also HPC machines), they got even more interesting. As a downside, their generalness makes them still not well suited for some tasks.

This paper attempts to show that by omitting or radically extending some well-known and now-obvious design solutions present in desktop computers' CPUs, one can get a computing device better suited for scientific computations' needs. Theoretical peak performance is presented to give some basic comparison of devices. Behaviour and performance for integer, single- and double-precision floating-point computations varies greatly, but the paper focuses strictly on the last data type.

2. Common mechanisms used in CPUs

This section discusses in general chosen few mechanisms that are rarely absent in common desktop CPUs and, at the same time, are widely known and understood. More detailed introduction can be found in [8].

2.1. Out-of-order execution

When instructions are executed in order they appear in instruction sequence, some CPU execution units (multipliers, adders, etc.) that could be used by next instructions, may remain idle waiting for any work to do. Out-of-order execution is about looking for nearby instructions that don't rely on their unfinished predecessors and have all necessary CPU resources free. If they exist, they get executed, thus improving CPU utilisation and increasing performance.

To perform out-of-order execution correctly, it must be seen from the outside of the CPU as if the instruction order was not perturbed, so additional logic to hide instruction reordering is necessary. Control logic needs to be augmented with a dependency-tracking circuitry to skip operations not ready to be performed and presence of interrupts require cancellation mechanism to invalidate instructions executed in advance.

2.2. Speculative execution

Out-of-order execution skips instructions not ready to be executed and waiting for other instructions to deliver them operands. Speculative execution performs those skipped operations by guessing what the value of missing operands may be. If the guess is correct, computations will finish sooner. Otherwise, the CPU will have to invalidate operations that were based on false assumptions and do them once more, this time with proper operands. Prediction mechanisms affect the guess rate.

The main advantage speculative execution brings is performance gain and greater resource utilisation at the cost of increased complexity and higher power consumption caused by additional work to be done when guess fails. The most known type of speculation is a branch prediction (a control-flow speculation), but value speculation is currently actively researched. [6] gives some insight on the subject from power-usage standpoint.

2.3. Vector processing

Modern general-purpose CPUs are inherently scalar, but they are commonly enhanced with some additional vector instructions and corresponding circuitry. Vector instructions operate on fixed-sized groups of scalar values of the same type and often perform exactly the same scalar action on each pair of corresponding vectors' elements. Usually vector processing provides higher computational performance for problems expressible, at least partially, in terms of vector computations.

Vector size is a trade-off between CPU area, power usage and available performance. Nowadays double-precision floating-point vectors are usually only two elements long, but CPUs are expected to have it doubled (or even quadrupled) in following generations. General-purpose CPUs' vector capabilities constitute a specific flavour of vector processing, as its design is usually very scalar-centric, as opposed to e.g. video cards. Requirement of data conversions and manipulations at junction of vector and scalar processing is expensive, in terms of either number of instructions or execution time; it is discussed further in [7].

Vector processing is well suited for algorithms from the domain of image processing, vector and matrix algebra, and so on. For inherently scalar and strongly data-dependent algorithms, vector instructions are of little or no use.

2.4. Multiple instruction streams

Internal CPU resources' usage may be improved further by handling more than one instruction stream simultaneously, sharing most of the logic, as seen in Hyper-threading technology from Intel. It's not about raising theoretical peak performance, but rather about reducing a gap between achievable and theoretical CPU parameters. Hyper-threading adds control logic that makes single processor appear as if there were two. In fact, additional control logic acts as a gate between CPU's environment and its internals. No execution units have to be added, the same pool is being used as in CPU without Hyper-threading. Yet, performance improves when each running thread's demands for execution units are disjoint. Conversely, when some instruction sequences require the same type of CPU resources, the application needs more time to finish. Much simpler approach is to employ number of independent CPUs and a communication subsystem to let them exchange data. Resulting system will inherently be able to handle multiple instruction streams. Notwithstanding its resource usage factor won't improve that way.

3. Different choices for acceleration

Computation accelerators are designed with different goals than CPUs are. Therefore they are constrained by other factors and different set of choices may be more valuable for them.

3.1. Backward compatibility

When the accelerator is an add-on device to the CPU, its design doesn't have to be constrained by backward compatibility. CPU designers are often bound to make new generations of processors accepting software from older ones. This is caused by large amounts of already written applications, that would need to be ported or replaced by new software, which is costly. For add-on accelerators working as co-processors the amount of code executed directly on them is not so large: usually it's limited to a group of small computational kernels. Therefore accelerators' designers may introduce major architectural changes in new chips, and retreat from decisions which proved to be wrong in practise.

3.2. Exceptions, interrupts

CPU is not a purely computational device, it also controls other pieces of computer system. Whatever the internal behaviour is, this makes CPU behave as if each instruction was executed in-order, one after another, so both external devices and internal subsystems may correctly react to current application state by signalling interrupts. Interrupt requires the CPU to stop its current work immediately and take an adequate action.

Presence of interrupts complicates circuitry implementing all kinds of ahead-of-time execution. Add-on accelerators are not concerned so much by interrupts. However, having strict interrupt handling in the hardware is useful for debugging the software.

In [2], authors discuss issues concerning interrupt handling schemes for out-of-order processors and propose their own solution; [1] explains that, under some circumstances, exceptions may be useful when speculative execution is involved.

3.3. Memory caching

As processors become faster, memory accesses become slower and slower with respect to processors' speed. It's common to use small auxiliary memories (caches) and let them keep the copies of data expected to be used soon or frequently. When memory subsystem notices the CPU wants to access the data kept in the cache, it redirects the CPU to the fast cache instead of slow main memory. As in speculative execution, caching efficiency partially depends on guessing which memory locations will be accessed next, and which data to keep in cache.

Usually, memory caching in CPUs is being done automatically by the hardware. Sometimes programmer is allowed to suggest which cache's behaviour is desirable, but it isn't rare for the hardware to ignore those suggestions. Caching can significantly improve overall performance, especially for memory-bound applications. However, when the hardware doesn't support or can't correctly detect software's memory access pattern, its efficiency gets small. Recently in accelerator devices, manual or hybrid cache management popularity increases.

3.4. Multiple execution units

Execution model with one instruction stream operating on several, possibly independent, data sets, is called *Single Instruction Multiple Data* (SIMD). When more than one simultaneous instruction stream is executed with several data sets, that's *Multiple Instructions Multiple Data* (MIMD). It's more flexible, but more expensive in hardware as well. CPUs support SIMD model when they include vector processing capabilities. However, accelerators often offer much larger vectors and lessen the need for data reorganisation.

Accelerators, being computational devices, involve more circuitry for actual computations than for control. They are designed for handling larger data sets, so for small data sets, for which CPUs suffice, they'll remain underutilised and not as efficient as they might be.

4. Popular computation accelerators

Some dedicated architectures known in High-Performance Computing industry getting most attention last year or two are presented next. FPGA devices are omitted, because their architecture is flexible, not fixed.

4.1. Cell/Broadband Engine

Cell processors, by IBM, Toshiba and Sony, consist of one PowerPC general purpose processor (*Power Processing Element*, PPE) and up to eight specialised *Synergistic Processing Elements* (SPEs) coprocessors controlled by PPE. As each processing element works independently, the whole architecture is MIMD. Support for double-precision floating-point computations appeared in the second generation of Cell processor, of which PowerXCell 8i is the main representative, so that's the model described further.

Both PPE and SPEs support vector instructions, however the vector size is small (just two elements for double-precision floating-point data type). Nevertheless, MIMD architecture is what makes Cell processor worth taking into consideration whenever SIMD is not enough to solve the problem in hand efficiently.

Each SPE has its own small local memory, manually controlled by the programmer; there's no automatic caching. Memory transfers between the main memory and local memories are implemented by a DMA mechanism. Cell provides some instructions dedicated to data reorganisation, so the processor is especially well-suited for algorithms working mostly at byte- or even bit-level. Internally SPEs and PPE communicate through *Element Interconnect Bus*, a ring bus where data transfer latency depends on a distance between the source and the target unit.

Cell processor is not designed as an add-on card. It isn't designed to work as an accelerator, either. Quite the opposite, general purpose PowerPC processor is to run operating system, and SPEs to handle purely computational tasks efficiently. But due to its performance, it is regarded similar to accelerators.

4.2. Video cards

Nowadays, video cards are providing necessary means to use them for computations unrelated to computer graphics. They are implementing a variant of the SIMD model, but different from the CPU's vector units, since there are no scalar–vector conversions needed. Designers decided to improve throughput, even if it leads to latency degradation, and that corresponds to requirements batch jobs processing large amounts of data in scientific computations have.

When programming the video cards, instances of code execution (here called *threads*[1]) are managed by the hardware. When sufficient number of execution resources is available, threads can be executed concurrently, otherwise sequential execution lefts; it's up to the hardware to choose ordering. Programmer only provides a function to be called (so called *kernel*) and specify a data-set over which the hardware will spawn function calls. When synchronisation isn't needed, one gets the software scalability, with respect to the number of execution units, easily.

The key to obtain optimal performance is to employ as many threads as possible, because thread multiplexing can hide long memory access times. It's also necessary to mix memory accesses with pure computations, to allow multiplexing to be effective. The maximum number of threads the hardware can service is limited directly by the hardware, and indirectly by the number of kernel's actively used registers.

Unfortunately, regardless of interesting design and high performance, manufacturers keep some technical information hidden from the users, thus successfully limiting chances of fully exploring video cards' capabilities.

4.2.1. GT200

NVIDIA's most recent graphics core, called GT200, is present in both GeForce GTX280 video card and Tesla C1060 computation accelerator. At the top level, GT200 consists of ([3]) *Thread Scheduler* and a group of *Thread Processing Clusters* (TPC). The Thread Scheduler assigns threads to the TPCs. TPC consists of a number of *Stream Multiprocessors* (SM), which execute the same computations on different operands, and some L1 cache, shared between all SMs. For double-precision

[1] Using the word *thread* in this context is misleading as it denotes something different than in CPUs. Moreover, when double-precision is considered, modern video cards' threads are inherently scalar, when CPUs' threads are not.

floating-point operations, SM can be thought of as a single execution unit that can work on a single operation at time.

In GT200, kernels can access different memory types; in [4] following types are listed: thread's local memory (not cached), shared memory inside each SM, device's global memory (not cached), texture cache and constant cache. So GT200 combines both automatically cached and manually managed memory types. Texture and constant cache refer to read-only regions of device memory, the difference between them lies in caching mechanism.

4.2.2. RV770

ATI's (now AMD) graphics core, RV770, is a major part of Radeon HD 4800-series video cards and FireStream 9200 computation accelerators family. It contains a group of *SIMD Cores* or *SIMD Engines.* Single SE has a *Thread Sequencer*, a *Texture Unit*, a *Local Data Share* and a group of *Thread Processors.* In double-precision floating-point computations, whole TP works as a single unit, but for single-precision and integer operations, TP provides internal units to do the work.

4.3. ClearSpeed accelerators

ClearSpeed's CSX700-based products are add-on floating-point computation accelerators. The CSX700 chip consists of two cores, *Multi-Threaded Array Processors* (MTAPs). Each one has two execution units of different type: *mono* and *poly.* The first one resembles a simple scalar microprocessor, lacking sophisticated mechanisms or logic necessary for modern operating systems' needs. The second one is a SIMD device, grouping *Processing Elements.* PEs contain circuitry for basic arithmetic operations, a set of registers and a small, local memory. They are chained together with a *swazzle path* for fast, internal data transfers.

Mono unit controls poly unit and handles branch instructions differently. To achieve high performance one has to carefully decide, how to split the code into both units. Unlike mono unit, poly unit cannot use data- or instruction cache. Contrary to video cards, the CSX700's threads (that hold the same meaning as CPU's threads) are not completely under the hardware's control: priorities and other means are provided to let the programmer decide how they interact and if they compensate for long main memory access or not.

5. Parameters

Real performance depends on how well solution for a given problem fits particular architecture so it can't be simply described by a single number. But other factors, including theoretical peak computational performance expressed in billions of floating-point operations[2] per second (GFLOP/s), presented in this section, can.

[2] Different operations vary in execution speed, e.g. multiplying is faster than dividing etc. Usually adding is the fastest operation of interest; often accelerators are designed to handle

Table 1

Parameters of general-purpose processors and computation accelerators

Device	Chip	Peak FP64 performance [GFLOP/s]	Power usage [W]	No. FP64 cores[a]	Local per core memory[b][kB]	Max. RAM [GB]
ClearSpeed Advance e710	CSX700	96	25	192 (2×96)	6	2
Radeon HD 4870	RV770	240	160	160	N/A	1
GeForce GTX 280	GT200	78	236	30	16	1
QS22	PowerXCell 8i (x2)	217[c]	250	16 (2×8)	256	32
Intel Xeon X7350	(4 cores)	46.9	130[d]	N/A	N/A	256
AMD Opteron 8350	(4 cores)	32	75[d]	N/A	N/A	128

[a] Concerns processing elements, so CPU cores are not counted
[b] Memory local to computation units, which can be both read and written and is not a cache
[c] 108.5 GFLOP/s each
[d] Only for CPU; memory's power usage is not included

Peak performance is an upper limit of how fast the device can compute and, in practise, assumes that an infinite sequence of multiply-and-add instructions is performed, with all operands and results located in registers. It is presented in table 1 along with performance of the quad-core CPUs (taken from [5]), power usage, per-computational core memory (if a device has a dedicated computational units), and the maximum amount of RAM supported.

5.1. Video cards

Video cards offer high peak performance (especially for single-precision floating-point data) at low cost. The amount of memory supported by video cards is relatively small. Data transfers between the host and the video card memory, not covered here, usually go through PCI Express bus, therefore communication time can be a bottleneck. To fully utilise available computing resources, the application must operate on very long vectors, longer than the number of execution units (to let the scheduler to hide memory latency).

As peak performance is calculated for multiply-add instructions, other computations lead to worse results, e.g. sequence of additions cannot achieve more than half of peak performance. The RV770 core seem to provide more execution units and thus better performance than the GT200, but in the real applications one must take into

multiplication followed by summation in a special way, so it's faster than those operations done separately or even two summations in row. That's why manufacturers usually inform how many additions or multiplications and additions their product can compute per second.

account also a memory system's effectiveness, if it can work fast enough to provide operands as they are needed.

5.2. Cell

Cell's theoretical peak performance assumes all the SPEs and PPE calculate vector multiply-add instructions. Because Cell is a MIMD architecture, and its execution units are more sophisticated than in SIMD-like devices, talking about effective vector size in Cell is artificial. So despite the fact each SPE has a two-element vector, table 1 just shows the number of SPEs.

Each SPE's local memory is greater than in the GT200, the RV770 or the CSX700, and so more data processing can be done between data transfers. Another feature of Cell processor (and the QS22) is the maximum amount of supported physical memory, which is much more than in other accelerator devices described here.

5.3. ClearSpeed modules

The CSX700, as used in the ClearSpeed Advance e710 acceleration board, works at the lowest frequency of 250 MHz, what leads to low power usage. Each MTAP has 96 PEs. Peak performance is achieved when doing a sequence of multiplications interleaved with additions, as those operations are handled by separate logic. Because hardware support for threads is different from that in video cards, there is no need to operate on large virtual vectors to compensate for memory latency.

6. Conclusions

General-purpose CPUs implement several sophisticated mechanisms to achieve high performance. It's fine for mostly scalar operations, but for large matrix or vector computations it may be not sufficient. For those applications, massively fine-grained data-parallel devices have been developed, aiming other goals than general-purpose CPUs. Performance of general-purpose processors constantly increases, so the question may arise: why not to wait for faster CPUs to appear? The answer seems clear: because performance of accelerators grows as well; because it's currently higher than that of CPUs; and they can help solving computational problems at hand now, instead of waiting.

Functional parameters of accelerators were roughly compared against CPUs'. They indicate that peak throughput of purely computational units varies from about 1.66 to 7.5 times the throughput of general-purpose CPUs. This potential for scientific computations' speed-up shouldn't be ignored. There are many ways to loose performance, thus a lot of care should be taken. For optimal software and algorithm design, detailed insight into device architecture is needed.

References

[1] Armstrong D. N., Hyesoon Kim, Mutlu O., Patt Y. N.: *Wrong Path Events: Exploiting Unusual and Illegal Program Behavior for Early Misprediction Detection and Recovery*, in: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, MICRO 37, Portland, Oregon, 2004

[2] Torng H. C., Day M.: *Interrupt Handling for Out-of-Order Execution Processors.* IEEE Transactions on Computers, vol. 42, 1993, 122–127

[3] *NVIDIA GeForce GTX 200 GPU Architectural Overview.* Available on: `http://www.nvidia.com/attach/1539847?type=support&primitive=0`, 2008

[4] *NVIDIA CUDA Compute Unified Device Architecture Programming Guide.* Available on: `http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf`, Version 2.0, 2008

[5] Barker K., Davis K., Hoisie A., Kerbyson D., Lang M., Pakin S., Sancho J. S.: *Experiences in Scaling Scientific Applications on Current-generation Quad-core Processors*, in: Proceedings of the Workshop on Large-Scale Parallel Processing (LSPP'08)/International Parallel and Distributed Processing Symposium (IPDPS), Miami, Florida, 2008

[6] Moreno R., Piñuel L., del Pino S., Tirado F.: *A Power Perspective of Value Speculation for Superscalar Microprocessors*, in: Proceedings of the 2000 IEEE International Conference on Computer Design: "VLSI in Computers & Processors" (ICCD'00), Austin, TX, USA, 2000

[7] Talla D., John L. K., Burger D.: *Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements.* IEEE Transactions on Computers, vol. 52, 2003, 1015–1031

[8] Moshovos A., Sohi G. S.: *Microarchitectural Innovations: Boosting Microprocessor Performance Beyond Semiconductor Technology Scaling.* Proceedings of the IEEE, vol. 89, 2001, 1560–1575