

High-Performance Reconfigurable Computing – the View from Edinburgh

Rob Baxter¹, Stephen Booth, Mark Bull, Geoff Cawood, Kenton D’Mellow,
Xu Guo, Mark Parsons, James Perry, Alan Simpson, Arthur Trew
EPCC and FHPCA

¹ communicating author: r.baxter@epcc.ed.ac.uk; 0131 651 3579;
University of Edinburgh, James Clerk Maxwell Building, King’s Buildings, Edinburgh EH9 3JZ

Abstract

This paper reviews the current state of the art in high-performance reconfigurable computing (HPRC) from the perspective of EPCC, the high-performance computing centre at the University of Edinburgh. We look at architectural and programming trends and assess some of the challenges that HPRC needs to address in order to drive itself across the chasm from the optimistic early adopters to the pragmatic early majority.

1. Introduction

High-performance reconfigurable computing (HPRC) is new.

Programmable logic devices and other pieces of reconfigurable hardware have been around for a long time but only recently have they found their way into high-performance computing (HPC).

EPCC at the University of Edinburgh has been around a shorter time – some 17 years – but has spent that time at the leading edge of HPC. As part of the FPGA HPC Alliance [1] EPCC have been involved in building a supercomputer from FPGAs and have embraced HPRC as a natural extension of ‘traditional’ HPC. In this paper we take a step back and offer a perspective on HPRC very much from an HPC angle.

The paper is structured as follows. Section 2 kicks off by asking the question *Why use FPGAs for HPC?* Section 3 takes a look at the foundations of high-performance reconfigurable computing and Section 4 looks at the range of HPRC architectures around today. Section 5 looks at programming models and tools from an HPC perspective, while Section 6 assess the suitability of applications of various types for acceleration using HPRC. We close in Section 7 by assessing some of the current challenges facing HPRC.

2. Why use FPGAs for HPC?

An FPGA is, of course, a field-programmable gate array. The most succinct way to define what this means in the context of HPC is by contrast with a traditional microprocessor or CPU. Traditional microprocessors contain a complex hardwired digital electronic circuit which fetches the instructions of the program and executes them one by one. FPGAs on the other hand contain a flexible array of simple interconnected ‘logic cells’, and programming the FPGA configures these logic cells and interconnections to create an arbitrary digital circuit.

FPGAs have been used over the years in a number of different application areas, including:

- prototyping and testing integrated circuits. FPGAs allow changes to be made and bugs in the circuit to be fixed easily before costly dies are made. When testing ends the same hardware description code can be used to fabricate an ASIC for mass production.
- custom hardware. For small production runs, FPGAs are cheaper than ASICs.
- accelerating software. FPGAs are especially suited to e.g. signal and image processing.
- high-performance computing.

This last is a relatively new area, but has attracted a lot of interest in recent years, so much so that an entire new phrase has been coined to describe it: high-performance reconfigurable computing, or HPRC.

Today large HPC vendors are already supplying machines with FPGAs ready-fitted, or have FPGAs in their product roadmaps. But why? What are the benefits of using FPGAs (and their attendant arcane skills) in HPC?

The first and most obvious answer is performance. HPC is renowned as that area of computing where current machine performance is never enough. The potential speed gains of using FPGAs are huge – two or three orders of magnitude for some applications kernels – and massive parallelism within an FPGA is possible. FPGAs also give performance junkies full control over hardware

at the lowest level, allowing the hand-crafting of caching strategies, pipelining, etc. which can be optimised for each application. Unnecessary components can be eliminated – don't need a square root? don't implement one! – and there is no instruction fetching/decoding overhead.

Secondly, and no less important these days, is low power consumption. FPGAs are low-power devices compared to traditional CPUs. Xilinx's latest Virtex-5 devices fabricated on a 65nm process boast power consumption figures for 'typical' designs of a little over 3 Watts [1] – contrast this with a typical 60-70 Watts for latest-generation Intel Core 2 Extreme [3]. FPGAs have lower clock speeds than CPUs, typically by a factor of 10, and coupled with their higher efficiency and less unnecessary circuitry this gives them a distinct edge. In an age where new HPC datacenters are being refused permission to expand by local power companies, and where power costs dominate new service contracts, this is a significant advantage.

Finally, when contrasted with other hardware acceleration strategies used in HPC FPGAs are highly flexible. Most hardware accelerators to date have been special purpose – QCDOC [4], for example, or the GRAPE gravity pipeline engine [5] – but an FPGA accelerator module can be configured and reconfigured to accelerate multiple applications.

FPGAs, then, have – at least on paper – a lot to offer HPC, and certainly this has been borne out by the buzz at recent Supercomputing conferences. A key question still to be answered, though, is: can HPRC manage the most common HPC performance killer – memory bandwidth?

Let's take a closer look at the ins and outs of HPRC.

3. Foundations of HPRC

Perhaps surprisingly, HPRC isn't new – it's been around since the 1960's. The original concept of reconfigurable computing is credited to Gerald Estrin [6] who described a hybrid computer structure consisting of an array of reconfigurable processing elements. His aim was to attempt to combine the flexibility of software with the speed of hardware, with a main processor controlling the behaviour of the reconfigurable hardware.

Estrin's concept was not realisable at the time – electronics technology needed to catch up a little – and HPRC didn't revive until the late 1980s. At that time silicon technology had advanced to a point that allowed complex designs to be implemented on a single chip – large and very large-scale integration which paved the way for today's system-on-a-chip designs and FPGA programmable logic devices.

The beginning of the 1990s saw the world's first commercial reconfigurable computer, the Algotronix

CHS2x4. This was an array of CAL1024 processors in ISA format with up to 8 FPGAs, each with 1024 programmable cells. It was funded by the Scottish Development Agency (now Scottish Enterprise) and shipped to participating UK universities.

Both Algotronix and Scottish Enterprise are founding partners of the FHPCA.

Since then there have been a plethora of reconfigurable architectures spawned from both industry and academia. These include: Garp from UC Berkley, combined RISC and FPGA processors on a single chip [7]; PACT-XPP, an industrial signal/media processing application [8]; FIPSOC, Field-Programmable System-On-Chip [9].

The common approach has been to combine reconfigurable elements with dedicated CPU cores or ASICs.

4. Common HPRC architectures

HPRC comes in quite a variety of flavours but at the basic level they tend to be defined as *fine-grained* or *coarse-grained*. Granularity here is defined as the smallest functional unit placed upon a reconfigurable element.

Raw FPGAs are fine-grained and can be manipulated at the bit level. This gives greater flexibility in design, but of course is more difficult to compile (place and route) for.

Coarser grained processing elements such as *reconfigurable Data Path Arrays* (rDPAs) are simpler to compile for and good for standard data tasks but lack the full flexibility of FPGAs. Which is more useful depends, of course, on the problem you're trying to solve.

As an attempt to classify the variety of HPRC architecture types (or at least those from reasonably mainstream vendors), consider the following list:

- plug-in pc boards as standard accelerators (eg ClearSpeed);
- FPGA daughterboards across slow bus;
- FPGA daughterboards with additional dedicated interconnect;
- tight integration into network fabric, including direct connection to NIC;
- direct connection to memory fabric;
- SRC's IMPLICIT+EXPLICIT architecture and MAP processor.

Below we look at each of these in turn, focusing primarily on FPGA-based systems.

4.1 Loosely-coupled coprocessors

Encompassing both FPGA daughterboards and other 'plug in' hardware accelerators the model here is a classic

coprocessor one. Acceleration hardware is plugged into the CPU's serial bus – usually PCI or PCI Express – and the CPUs are connected together over a standard (Ethernet) network. This model is commonly found in smaller systems of a few CPUs.

Suitable applications tend to be limited; getting efficiency out of such systems requires a highly optimised data flow model, ideally small volumes of input and output data and a high compute to i/o ratio. Applications that are suited can see excellent serial performance.

Parallel performance is generally rather poor, making this model somewhat unattractive for HPRC. The serial bus is easily overloaded (even with one FPGA) and transferring data to and from the FPGA can take longer than implementing algorithm on the host CPU! For a parallel algorithm, all data are forced to pass through host network, making scalability very hard to achieve.

This approach is, however, relatively cheap.

4.2 Tightly-coupled coprocessors

By 'tightly-coupled' here we really mean 'coprocessors with direct connections'. The FHPCA's Maxwell supercomputer falls into this model: a plug-in coprocessor-based system with additional dedicated interconnect between FPGAs.

Basically an extension of the previous architecture this model allows direct communication between FPGAs over a point-to-point network, typically some form of Infiniband or other fast interconnect. This model allows much greater potential for parallelization and the possibility of good scaling performance – it's well suited to a wide number of domain decomposition problems that only need nearest-neighbour communications, for instance. Another feature of this dual network model is that both can be used simultaneously; CPU and FPGA network uses are not necessarily in contention.

On the downside this model requires a reasonably complex network topology – usually some form of n -dimensional torus – which apart from the physical challenge of the wiring can cause unexpected bottlenecks. Additionally this model still has an on/off board bottleneck for FPGA-to-CPU communications which needs to be factored in to application development – only using the CPU network for global communications, for instance.

The high performance network makes this more expensive than the simple co-processor model, but still relatively inexpensive.

4.3 Integration with network fabric

This model is a further refinement, keeping the separate CPU and FPGA networks but cross-wiring them

by (for example) wiring the FPGAs directly into a network interface chip (NIC). The FPGAs still maintain a dedicated high-speed point-to-point network over, eg, Rocket IO, but they can now talk over the host network too.

The big difference here is that by connecting FPGAs directly to the host network via NICs the concept of one particular CPU 'owning' one or more FPGAs is broken – any FPGA can communicate directly with any CPU. This provides for arbitrary data and control communication patterns, giving tremendous flexibility to the programmer, albeit with increased complexity.

On the downside, FPGA performance can be impacted by CPU contention for the host network which can cause timing problems in FPGA datastreams unless some form of packetisation protocol is implemented on the devices. This is where FPGAs can usefully revert to the Rocket IO or other dedicated network.

Cray's XD1 is a good example of this type of architecture, combining AMD Opteron processors and Xilinx Virtex-4 LX160 FPGA devices over Cray's RapidArray interconnect.

4.4 Direct memory connection

After the fashion of the network-integration architecture, this variant goes a little deeper and connects acceleration hardware directly to a system-wide shared-memory bus. This yields a true any-to-any network topology – indeed an anything-to-anything else one – and requires a strong degree of controller hardware to maintain coherency across the connected memories and devices.

The prime example of this tightly-coupled model is SGI's RASC (Reconfigurable Application Specific Computing) architecture and their NUMalink interconnect fabric. SGI's cache-coherent NUMalink provides a global shared memory bus into which a wide variety of devices can be plugged; SGI provide variants based on Xilinx Virtex-II and Virtex-4 FPGA families.

The major advantage of this approach is that the FPGAs can be treated as features in the global processing landscape engendered by the cc-NUMA interconnect; programming layers can be put on top which hide the complexity of FPGA programming without compromising efficiency too far.

The main downside of this approach is expense.

4.5 SRC's IMPLICIT+EXPLICIT

SRC's IMPLICIT+EXPLICIT architecture is essentially a hybrid of the other types. The concept is based on the ideas of implicit versus explicit programmability: the CPU (or ASIC for that matter) is

regarded as being implicitly programmed, via the operating system and other software instructions, while the reconfigurable component (either FPGA or rDPA etc) is explicitly controlled in that the programmer has direct control over the logic and data-stream.

SRC's architecture defines a tightly integrated combination of microprocessor and reconfigurable elements, sharing the same memory, I/O and network systems. This allows very fast communications to and from the FPGAs, and direct network access to the FPGAs through the unified NIC and memory controllers.

The aim here is to make the FPGA optionally invisible, providing for ease of portability: the application *can* employ the FPGA, but doesn't have to. SRC's compiler tools allow users to build a *unified executable* which controls both implicit and explicit devices in a relatively seamless way – a single binary image capable of instructing both processors.

One consequence of this is that the available FPGA algorithms are coarse-grained; much of the flexibility of FPGAs is hidden by the tight integration of FPGA, CPU and memory.

5. HPRC programming models

Reconfigurable devices are reconfigurable at the level of logic gates. Rewriting an algorithm really means rewiring a circuit and for typical HPC users this is a *major* challenge.

HPC is sufficiently mature after 15 years of mainstream parallel computing that there now exists a large body of finely-tuned parallel software based on the 'cache-based RISC processor plus MPI' architectural model. For HPRC to gain traction in this community it must bear this in mind – the community are unlikely to rewrite their software unless it is *really* worth the investment.

That said, there are a number of approaches to managing the undoubted complexities of programming reconfigurable devices.

5.1 Standardised libraries

The simplest approach to realising hardware acceleration from the application developer's perspective is to make use of a vendor-supplied accelerated library. Standard maths or HPC libraries (libm, fftw, BLAS etc) can be supplied ready configured and can include automatic parallelisation across many FPGAs.

For such 'out of the box' solutions parallel performance can, in principle, be good, although not optimal, or perhaps even suited to FPGAs! ClearSpeed's CSX600 outperforms any FPGA for standard library acceleration, and such canned solutions require no code

modifications or even recompilation – in some cases, re-linking is all that is necessary.

Such library approaches suffer the major drawback of data transfer overhead between CPU and reconfigurable hardware for each call. For example, a single code repeatedly employing two alternate library calls will suffer from potentially huge and unnecessary bus or network traffic.

5.2 Customised hotspots

One way around the 'two library calls' problem noted above is for a user to develop their own customised library for compute-intensive 'hotspots'. This approach maps the 'meat' of the algorithm onto the FPGA, and calls it via subroutine. This can suffer from the same network overheads associated with standard libraries, and parallelisation across many FPGAs is possible but difficult – it is essentially left to the user.

The key advantage is that an approach more tailored to the particular numerics of the algorithm in question can result in both serial and parallel performance being excellent, but at fairly high development expense. Hand crafting accelerated routines on FPGAs is non-trivial for a typical computational scientist – the place-and-route build process for an FPGA bitstream is significantly more complex than the usual Fortran edit-compile-run cycle.

5.3 FHPCA's Parallel Toolkit

A natural extension of the customised hotspot approach is that taken by the FHPCA in the Parallel Toolkit (PTK) model. Rather than work on small kernels or library calls the PTK tries to incorporate as much of whole application kernel onto the reconfigurable hardware as possible. This way the PTK provides a uniform multi-vendor interface between application code and underlying hardware resources and aims to minimise traffic to and from host CPU.

Serial and parallel performance are potentially excellent, but still with a significant development expense – the PTK approach relies on both generic *and* application-specific accelerator cores and the more complex the application the more VHDL code needs to be generated to ensure the bulk of the algorithm runs hardware-side.

5.4 C-to-gates tools

Rather than force application developers to write VHDL or Verilog code directly a number of independent software vendors supply compiler tools which will take high-level languages and 'compile' them down to VHDL and FPGA network descriptions which can be fed directly

into hardware synthesis tools. This way, high-level language code – C for instance – can be compiled directly down to reconfigurable hardware.

Such tools include Mentor Graphics' CatapultC, Celoxica's Agility compiler and DK Design Suite, Nallatech's DIMETalk, Impulse Accelerated Technology's Impulse-C and others, and while sounding like Nirvana for HPRC application developers the reality is not quite as rosy.

Most of these tools – with the notable exception of Mentor's CatapultC – support non-ANSI compliant dialects of C. The IEEE standard SystemC (essentially a C++ extension for describing hardware systems) is supported by Mentor and Celoxica; other tools tend to have their own C dialects – Handel-C for Celoxica, DIME-C for Nallatech, Impulse-C for Impulse.

A second caveat is that tuning the application dialect-C code to achieve reasonable performance moves the application further and further away from standard C; essentially the application suffers almost immediately from the flipside of any extreme optimization technique – broken portability and disappearing comprehensibility.

Current C-to-gates tools, then, suffer from a lack of standardization on a single dialect, although SystemC is probably a fair bet to take on this role. And sharp-eyed Fortran or Java programmers will have noticed that these tools are C compilers – no other high-level language is currently supported by FPGA compiler vendors.

5.5 Cray's Software Integration

The programming models noted so far are all generic; more advanced models exist with more tool support but currently they are all proprietary. In this HPRC mirrors parallel computing in the early 1990s – lots of ways to do inter-process communication on particular vendor systems but no standardisation, and thus little portability.

Cray, for instance, have developed a model called Software Integration for the XD1 line of accelerated systems. Software Integration makes use of operating system-level and communications management software to integrate the FPGAs into the system. A C-based API allows the host CPU to interface to, and even re-program, the FPGA during execution.

This way large algorithms can be broken up into bite-sized FPGA 'personalities', and host CPUs can download the personality required by the current operation at runtime. Cray's any-to-any architecture allows any CPU 'host' to program any FPGA.

Additionally standardised libraries provide a 'quickstart' ability to get programs running with little development effort.

5.6 SGI's RASCAL

SGI provide a programming model called RASCAL (RASC Abstraction Layer) for their Altix NUMALink-based servers. RASCAL enables serial or parallel FPGA scaling through a combination of API and core services libraries, and also provides direct support for third-party tools and compilers such as Celoxica-C and Mitron-C. Xilinx Synthesis Technology provides support for incremental and modular designs at the coarse-grained level.

5.7 SRC's CARTE

SRC provide the CARTE Programming Environment as an integrated part of their MAP-based systems. CARTE provides fundamental support for the IMPLICIT+EXPLICIT architecture, allowing the single compilation of high level language (C or Fortran) into a unified executable. The CARTE compiler analyses the application code and assigns subsections to either CPU or FPGA. The compiler also creates the logic design and all required interfacing using again a coarse-grained model, essentially collating a series of pre-defined computational units.

CARTE offers a very user-friendly approach and short development times, but lacks the efficiency of hand-tuning or fine-grained approaches that give the user full control of FPGA use.

Such is the Catch-22 of FPGA development – make it easy for non-hardware engineers to program them and you can lose out on the performance advantage; leave the users with full control over every gate and they may give up.

5.8 The Mitron Virtual Processor

Mitronics have attempted to solve the HPRC Catch-22 with a novel approach that deploys a 'virtual processor' architecture onto a base FPGA. The Mitron Virtual Processor can be thought of as a 'compute cluster on a chip', with pipeline connections between functional units acting in some ways as network connections between parallel processors.

The Mitron platform offers a C dialect, Mitron-C, in which applications can be written or rewritten from standard C. The Mitron compiler then converts the C code into a configuration file for the virtual processor – a half-way house between application code and hardware reconfiguration. This makes the whole process of accelerating an application much easier while still providing a finer-grained model than traditional library approaches.

Mittrion has the additional advantage of being a horizontal platform solution, running on SGI, Nallatech and Cray hardware – all based, of course, on Xilinx FPGAs.

5.9 Xilinx Labs' CHiMPS

Recently Xilinx Research Labs have announced an ongoing project [10] which aims to address the programmability of Xilinx devices with a particular focus on high-performance computing applications. Xilinx's aim with CHiMPS is to provide in essence a virtual machine layer (the CHiMPS Target Language) which can be targeted in a straightforward way by traditional procedural or object-oriented languages and yet can also be easily converted into a data-flow graph for mapping to FPGAs.

Xilinx report good early results with CHiMPS, even seeing cost/performance figures that beat 3 GHz Pentium 4 processors for certain benchmarks, and early HPC users of CHiMPS also report a much more straightforward code development process [11]. CHiMPS could turn out to be a key development in HPRC programming.

6. Application suitability

Despite the variety of hardware and software platforms available for HPRC, reconfigurable hardware is clearly not a silver bullet for HPC performance. By no means all HPC applications can benefit from hardware acceleration under current architectures. We look at some of the key challenges yet to be overcome in Section 7, but firstly let us consider types of applications that *are* well suited to HPRC.

As we have remarked, the major stumbling block for a lot of HPC codes is not raw algorithm performance but memory bandwidth – keeping the algorithms fed with data. There is an argument that accelerating algorithms in hardware exacerbates the problem; the flipside is that applications *without* large memory bandwidth requirements stand to gain the most.

Applications like search and sort work well – witness Mittrion's success with the BLAST genome searching tool [12] – as does encryption where the extensive use of XOR operation makes this very well suited to FPGAs.

Fast Fourier transforms and random number generation (eg. for Monte-Carlo simulations) work very well – indeed FPGAs can be made to generate truly random numbers (rather than pseudo-random) using a non-algorithmic configuration of the FPGA as an array of ring-oscillators, from which one can harvest the electronic noise, or 'jitter'.

Other highly-suited applications include real-time data visualisation and image processing, general purpose

signal processing and data collection, and error correction, coding and decoding. However few of these are 'traditional' HPC areas, where data-bound simulations of physical systems rule the roost. The classic areas of HPC – fluid dynamics, physics, structural engineering, chemical process engineering – give rise to the main challenges to HPRC.

7. Current challenges to HPRC

So, is HPRC the future of HPC? Possibly, although there are, as we see it, three major challenges to overcome:

- programmability;
- memory bandwidth performance;
- price/performance.

7.1 Programmability

Currently, high-level programmability tools for FPGAs are still in the early days of development. Good tools exist, but for non-standard dialects of only one of the 'big two' languages of HPC. Good programming environments exist, but are very platform specific. FPGA standards exist, but only at the core level.

The question HPRC tool providers need to address is the following: I have a parallel Fortran code in which my organisation has invested many staff-years' of effort. It is written as portable Fortran 90 with MPI library calls for message-passing and OpenMP directives inline to take advantage of multi-threading nodes. I don't mind making small changes to the code to take advantage of the latest hardware advances, but portability and maintainability are extremely important to preserve my investment in this code. Now, given these constraints, can I take advantage of HPRC?

There is a way to go yet before HPRC can answer with a definite 'yes', although signs are promising.

7.2 Memory bandwidth issues

As already mentioned one of the typical performance killers for many HPC codes is memory bandwidth, or rather its lack. Here HPRC finds itself in the same boat as traditional HPC – the performance of memory systems is just not keeping pace with the performance of the FLOP unit. Significant work needs to be done to improve memory architectures in general, and especially for the cluster architectures of both HPRC and traditional HPC. Adding more and more processing power into the same basic von Neumann architecture will not work.

As FPGAs get bigger, which they will do, the possibilities for HPRC to work around this issue increase faster than those of CPU architectures. Bigger devices

can support more memory interfaces, more banks of memory can be used simultaneously, bigger block RAMs can be configured into the FPGAs themselves. However, the real challenge is for memory manufacturers rather than HPRC vendors.

7.3 Price/performance

Currently reconfigurable technology big enough for serious computing is still very expensive. Given that none of Intel, AMD, Sun and IBM have yet given up on wringing more performance out of a single microprocessor die, Moore's Law has not yet halted CPU performance. Multicore CPUs in volume have a significant price/performance advantage over FPGAs even before additional reconfigurable development costs are factored in.

The big win for FPGA technology here is not so much in comparing MFLOPs/dollar with CPUs but in comparing MFLOPs/Watt and Watts/dollar. The low-power nature of reconfigurable computing when amortised over a large system lifetime may see HPRC become increasingly attractive.

7.4 The rise of multicore CPUs

The recent trends in multicore CPU development may be a mixed blessing for HPRC. While dual and quad core Cores and Opterons have undoubtedly produced a quantum leap in processing power over the last year, the efficient programming of multithreaded CPUs is not trivial. As core numbers increase towards 80 and beyond the complexity of programming performance out of multicore CPUs will start to approach the complexity of programming reconfigurable hardware, and this suggests the door for HPRC is far from closed.

HPRC today does indeed look like parallel computing 15 years ago – and that turned out very nicely indeed, thank you.

8. Trademarks

'Xilinx', 'Virtex' and 'RocketIO' are trademarks or registered trademarks of Xilinx Corporation.

'Intel' and 'Core' are trademarks or registered trademarks of Intel Corporation.

'SRC', 'CARTE', 'MAP' and 'IMPLICIT+EXPLICIT' are trademarks or registered trademarks of SRC Computers Inc.

'SGI', 'Altix', 'RASC', 'RASCAL' and 'NUMalink' are trademarks or registered trademarks of Silicon Graphics, Inc., in the United States and/or other countries worldwide.

'Infiniband' is a trademark of the Infiniband Trade Association.

'PCI', 'PCI-X' and 'PCI Express' are trademarks or registered trademarks of the PCI-SIG.

'Cray', 'XD1' and 'RapidArray' are trademarks or registered trademarks of Cray Inc.

'AMD' and 'Opteron' are trademarks or registered trademarks of Advanced Micro Devices Inc.

'ClearSpeed' is a trademark of ClearSpeed Technology plc.

'Catapult' is a registered trademark of Mentor Graphics Corporation.

'Celoxica' and 'Handel-C' are trademarks of Celxica ltd.

'DIMEtalk' and 'DIME-C' are trademarks of Nallatech ltd.

'Impulse-C' is a trademark of Impulse Accelerated Technologies Inc.

'Mitronics' and 'Mitrion' are trademarks of Mitronics AB.

'Catch-22' is, originally, a novel by Joseph Heller.

9. References

- [1] The FHPCA, www.fhpc.org
- [2] Xilinx Virtex 5 power characteristics, www.xilinx.com/products/silicon_solutions/fpgas/virtex5/virtex5/advantages/power.htm
- [3] Intel Core 2 Extreme datasheets, www.intel.com/products/processor/core2XE/index.htm
- [4] The QCDOC project at Brookhaven National Lab, <http://lqcd.bnl.gov/>
- [5] The GRAPE project, grape.astron.s.u-tokyo.ac.jp/grape/
- [6] G. Estrin, "Organization of Computer Systems – The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, Western Joint Computer Conference, New York, 1960, pp. 33-40.
- [7] Berkeley reconfigurable architectures, systems and software GARP project, <http://brass.cs.berkeley.edu/garp.html>.
- [8] PACT-XPP, <http://www.pactxpp.com/>
- [9] J. Faura, M. A. Aguirre, J. M. Moreno, P. van Duong and J. M. Insenser, "FIPSOC: A Field Programmable System On a Chip", *Proc. XII Design of Circuits and Integrated Systems Conference*, 1997 (DCIS97).
- [10] D. Bennett, E. Dellinger, J. Mason, P. Sundarajan, "An FPGA-oriented target language for HLL compilation", RSSI 2006, http://gladiator.ncsa.uiuc.edu/PDFs/rssi06/presentations/13_Dave_Bennett.pdf
- [11] O. Storaasli, C. Steffen, *private communications*.
- [12] Mitrion BLASTN and Mitrion-C Open Bio Project, <http://www.mitrion.com/default.asp?Id=27>