Maxwell – a 64 FPGA Supercomputer

Rob Baxter[1], Stephen Booth, Mark Bull, Geoff Cawood, James Perry, Mark Parsons, Alan Simpson, Arthur Trew
EPCC and FHPCA

Andrew McCormick, Graham Smart, Ronnie Smart
Alpha Data ltd and FPHCA

Allan Cantle, Richard Chamberlain, Gildas Genest
Nallatech ltd 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

We present the initial results from the FHPCA Supercomputer project at the University of Edinburgh. The project has successfully built a general-purpose 64 FPGA computer and ported to it three demonstration applications from the oil, medical and finance sectors. This paper describes in brief the machine itself – Maxwell – its hardware and software environment and presents very early benchmark results from runs of the demonstrators.

1. Introduction

Against the background of possibilities in the emerging area of high-performance reconfigurable computing [1]the FHPCA was founded in early 2005 to take forward the ideas of an FPGA-based supercomputer.

The FPGA High Performance Computing Alliance (FHPCA [2]) is focused on the development of computing solutions using FPGAs to deliver new levels of performance. It will enable a range of applications to be able to exploit the benefits that such an approach will deliver.

This will be achieved through the implementation of a large scale demonstrator or 'supercomputer' – Maxwell – and supported by a structured programme of knowledge diffusion designed to raise industry awareness, interest and create commercial advantage.

The Alliance builds on Scotland's world-class reputation in the field of reconfigurable computing. The first commercial reconfigurable computer was created by a Scottish company in the early 1990s and Scottish companies and scientists have since been instrumental in developing and advancing the technology.

The alliance partners are Algotronix, Alpha Data, EPCC, the Institute for System Level Integration, Nallatech and Xilinx.

The project has been facilitated and part funded by the Scottish Enterprise Industries team.

This paper is structured as follows. Section 2 describes the basic design goals behind Maxwell. Section 3 delves into the details of the machine's hardware while Section 4 describes the software environment and programming methodology used in porting a number of demonstration applications. Section 5 discusses three key demonstration applications from the fields of financial services, medical imaging and oil and gas exploration., and Section 6 presents early performance results from these applications on Maxwell. Finally Section 7 offers thoughts for the future.

2. Design goals

Maxwell is designed as a proof-of-concept general-purpose FPGA supercomputer. Given the specialized nature of hardware acceleration the very concept of 'general-purpose' for HPRC is worth investigating in its own right. Can a machine built to be as broadly applicable as possible deliver enough FPGA performance to be worth the cost?

2.1 Target application requirements

FPGA computers generally, and Maxwell specifically, will not be good at everything. The fairly extreme nature of hardware acceleration means that certain applications will be more suited to it than others. We note here a number of these features which the Maxwell (and similar) FPGA-based system places on applications.

Significant runtime in computational kernels. The use of FPGA acceleration is an extreme optimisation technique similar to the use of hand-coded assembly language. It requires the replacement of sections of a program (computational kernels) with new optimised versions. For this to be economic the effort required to develop the replacement versions must be justified by the reduction in overall runtime of the application. As only the performance of the optimised routines is improved this in turn requires that a significant amount of the application runtime should take place in a small number of computational kernels that are amenable to FPGA optimisation.

Computational kernels are small. FPGA acceleration builds accelerated versions of key computational kernels out of FPGA resources. These FPGA resources are limited which in turn limits the complexity of the kernels that can be implemented. Though the size of FPGAs will increase over time it will always be advantageous to have small computational kernels, as this will increase the scope for parallel processing within the accelerated kernel.

Computational kernel data can be made private. Maxwell has dedicated memory banks connected to the FPGAs. To achieve reasonable performance the kernels need to access the majority of their data from these dedicated memory banks. Before an application can be ported to the FPGA prototype it must be refactored[1] so that the majority of the data accesses made by the kernel routines are to data structures that are only accessed by the kernel routines and explicit copy-in copy-out operations. These 'private' data structures can then be implemented using the FPGA memory banks as part of the optimisation process.

This effectively requires that application porters utilise an *object-oriented style* of programming at the bottom layers of the application.

Data transfers between kernel-private and other data must be minimal. In the prototype system the FPGA memory banks are accessed from the host system via a PCI bus. This has relatively low bandwidth compared to normal memory speeds and large amounts of data transfers between the FPGA memory banks and the host memory could be detrimental to performance.

To keep these transfers to a minimum we require that either (a) the work implemented by a single kernel invocation is sufficiently large to amortize the cost of copying the data to and from the FPGA memory banks, or (b) that data can be left resident on the FPGA memory banks between kernel calls without the host program needing to access it.

Restricted workspace is required for computational kernels. Only limited memory is available in the memory banks directly attached to the FPGAs (between 0.5 and 1 GB per FPGA). This gives a total space for FPGA memory of 48GB. The inner working set of the application therefore needs to fit in this space.

[1] *Refactoring* refers to the practice of restructuring and rewriting lines of program code while not changing external program functionality.

Parallel computational kernels should be compute bound rather than memory bound. Though the directly connected memory banks on the FPGA modules provide reasonable memory bandwidth the total memory bandwidth available in the prototype system is not significantly different from that available in more conventional compute clusters. As a result of this any application that is predominantly memory bound will not perform significantly better on the prototype system than on a conventional cluster.

Parallel decomposition must be compatible with a 2D torus. Maxwell is constructed as a two-dimensional torus of FPGA nodes connected by Rocket-IO channels. This provides very good communication performance for this particular network topology. However we do not have any general purpose communication IP available in this project so any communication patterns that are not sub-sets of the 2D torus must be either constructed explicitly using virtual links overlaid on top of the physical 2D torus or routed through the host processors and their attached network.

3. Hardware overview

3.1 Physical view

Physically, Maxwell comprises two 19-inch racks and five IBM BladeCentre chassis. Four of the BladeCentres have seven IBM Intel Xeon blades and the fifth has four.

Each blade, a diskless 2.8 GHz Intel Xeon with 1 GB main memory, hosts two FPGAs through a PCI-X expansion module. The FPGAs are mounted on two different PCI-X card types – Nallatech HR101s and Alpha Data ADM-XRC-4FXs. This provides an interesting mixed architecture and an environment in which to experiment with vendor-neutral programming models.

The FPGAs have between 512 and 1,024 MB external memory and four MGT Rocket IO connectors, each of which is capable of shunting data at 3.125 Gb/s. All 64 FPGAs are wired together directly in a two-dimensional torus. This direct connection is what makes Maxwell such an interesting system to program, allowing for full distributed-memory parallel programming purely on the FPGAs. The CPUs are also connected over gigabit Ethernet through a single 32-way switch.

There is also a single login-node (a simple Dell server) with a host-bus adapter connection to the University's research computing SAN.

3.2 Logical view

Logically we regard Maxwell as a collection of 64 nodes. A node is defined as a software process running on a host machine, together with some FPGA acceleration hardware, as shown in Figure 1. In full operation each blade CPU thus hosts two software processes, each of which 'manages' one FPGA during runtime.

The combination of CPU-to-CPU Ethernet connections and FPGA-to-FPGA Rocket IO channels gives Maxwell two independent comms networks. The general approach is to use the Ethernet network purely as a control network and to perform parallel communications over Rocket IO. The Ethernet is also used for any explicit MPI calls that remain in the application – for instance for start-up data distribution or finalizing data marshalling on completion.

The FPGA connections are purely point-to-point – we do not implement routing logic in the FPGA devices. This makes the Rocket IO network highly suitable for nearest-neighbour communication patterns but less than ideal for reduction operations such as global sums. For these, applications call back to the host CPUs for MPI reduction operations to be performed over Ethernet.

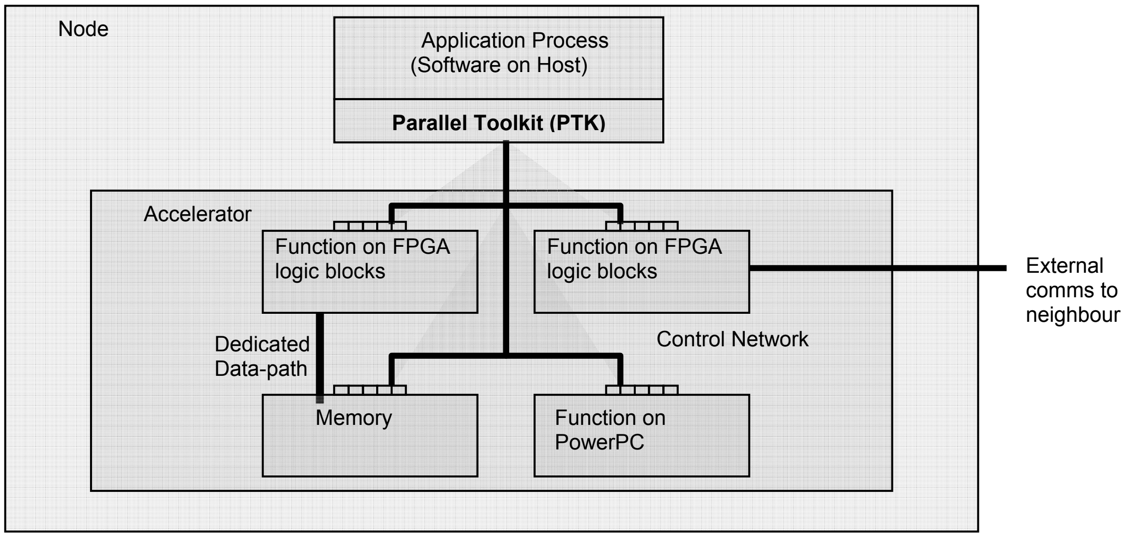
Node
Application Process (Software on Host)
Parallel Toolkit (PTK)
Accelerator
Function on FPGA logic blocks
Function on FPGA logic blocks
External comms to neighbour
Dedicated Data-path
Control Network
Memory
Function on PowerPC

Figure 1. Abstraction of a node on Maxwell.

3.3 Public/private collaborative success

Maxwell has been an excellent example of university/ business collaboration here in Scotland. While development support was provided by Scottish Enterprise, the hardware for the machine itself has all been donated by the partners.

The IBM BladeCentres, bladeservers and cables were provided by EPCC. The 64 Virtex-4 devices were donated by Xilinx; the 32 HR101 cards were provided by Nallatech and the 32 ADM-XRC-4FX cards by Alpha Data. Without such genuine commitment from all the project partners this machine would not have been built.

4. Software environment

Our aim with the environment on Maxwell was to make it as 'HPC-system-like' as possible. Uptake of FPGA and related technologies in HPC will be hindered if the machines require a whole different approach to that of 'mainstream' HPC.

In the early days of parallel computing every vendor had their own way of doing things and progress for application developers was slow. One vendor's communication protocols did not map onto another's and machine environments were very different. Today, parallel environments are much more standardized and there is a lot of 'legacy' software built using libraries like MPI, BLAS and SCALAPACK. Machines requiring significantly different approaches will not find much traction.

Thus Maxwell looks very much like any other parallel cluster. It runs the Linux variant CentOS and all standard Gnu/Linux tools. It offers Sun Grid Engine as a batch scheduling system and MPI for inter-process communication.

The one novel innovation is the Parallel Toolkit (PTK). The PTK has been developed as part of the overall design of Maxwell and provides an attempt to enforce top-down standardization on application codes across the different 'flavours' of hardware. The PTK is described in detail elsewhere, but suffice to say that it is a set of practices and infrastructure intended to address key issues in hardware acceleration e.g.

- associating processes with FPGA resources;
- associating FPGAs with bitstreams;
- managing contention for FPGA resources within a process;
- managing code dependencies to facilitate re-use.

5. Demonstration applications

There is little point in building any kind of computer unless it has something to do. As part of the overall project we also ported three demonstration applications to run on Maxwell.

Two criteria were used to select these applications. Firstly they were chosen from the application areas of financial engineering, medical imaging and oil and gas, three areas judged generally to have most to gain from hardware acceleration solutions of one form or another. Secondly they were chosen to illustrate progressively more complex parallel application features, from trivial parallelism and simple data requirements to full-scale distributed-memory parallelism.

In all three cases we adopted the methodology of the PTK: identify the application hotspot; define an abstract object-oriented interface that encapsulates the hotspot; refactor the code against this interface; generate accelerated versions of the hotspot code underneath the interface. In each case we also produced a 'pure software' version of the hotspot against the same PTK interface, providing a direct point of comparison for both testing and benchmarking purposes.

5.1 Monte Carlo option pricing

Financial engineering is a mathematical branch of economics that deals with the modeling of asset prices and their associated derivatives. One of the cornerstones of financial engineering is the Black-Schole model of prices [3], essentially a recasting of the equations of physical heat diffusion and Brownian motion.

The assumptions of the Black-Scholes model imply that for a given stock price at time t, simulated changes in the stock price at a future time t + dt can be generated by the following formula:

$$dS = S\, r_c\, dt + S\, \sigma\, \varepsilon \sqrt{dt}$$

where S is the current stock price, dS is the change in the stock price, r_c is the continuously compounded risk-free interest rate, σ is the volatility of the stock, dt is the length of the time interval over which the stock price change occurs and ε is a random number generated from a standard Gaussian probability distribution.

The pricing of stock options follows the Black-Scholes model, and simple stock options (so-called 'European' options) can be priced with a simple closed-form formula called, unsurprisingly, the Black-Scholes formula. More complex options such as those whose final price depends on a time-average or other path-dependent price calculations have no closed form and are typically priced using stochastic or Monte Carlo modeling.

Our first demonstration applications supposes you wanted to price an 'Asian' option, an option in which the final stock price is replaced with the average price of the asset over a period of time, computed by collecting the daily closing price over the life of the option. The price can be modeled as a series of dSs over the option's lifetime (say $N_{timesteps}$). The formula for each dS is based on the previous day's closing price, and the average of the $N_{timesteps}$ stock prices would determine the value of the option at expiration.

The above gives you one possible future for the stock price; repeating the model N_{runs} times allows the process to converge on the 'right' option price. N_{runs} here is of order 10,000 – 50,000.

Based on the above model, a serial code would be as follows:

```
for i = 1, $N_{runs}$
  for n = 1, $N_{timesteps}$
    $\varepsilon$ = gaussianRandomNumber()
    S[n][i] = S[n-1][i] (1 + $r_c$ dt + $\sigma\ \varepsilon\ \sqrt{dt}$)
  endfor n
  $S_{av}$[i] = 1/$N_{timesteps}$ $\Sigma_n$ S[n][i]
  c[i] = max($S_{av}$[i] - K, 0)
  p[i] = max(K - $S_{av}$[i], 0)
endfor i
$S_{bar}$ = 1/$N_{runs}$ $\Sigma_i$ $S_{av}$[i]
$c_{final}$ = 1/$N_{runs}$ $\Sigma_i$ c[i]
$p_{final}$ = 1/$N_{runs}$ $\Sigma_i$ p[i]
```

K here is the strike price of the option, the price defined in the option contract; r_c and σ are as defined above.

Pricing a single Asian option thus requires $N_{runs} \times N_{timesteps}$ Gaussian random numbers (plus four multiplies and three adds each).

Our demonstration captures this whole core, parameterized, on FPGA, and batches similar pricing calculations for different stocks/assets across the whole 64 FPGAs. In fact the demonstrator core is so compact that it can be replicated 10 times or more across a single FPGA device, providing an additional order of magnitude in possible speedup.

This demonstrator – MCopt – is the simplest of the three applications, having a simple, compact computational core and very limited data requirements.

5.2 Three and four-D facial imaging

The second demonstration application was produced in collaboration with Dimensional Imaging 3D ltd, a firm specializing in three and four-dimensional facial imaging for medical applications such as maxilo-facial surgery [3].

The principle here is that a digital camera rig is used to capture pairs of still (for 3D) or video (for 4D) images. Each stereo pair is then combined to produce a depth map which contains full three-dimensional information and is used to create a 3D software model, or a 3D video in the case of 4D capture.

Image combination is an expensive business and is an ideal application for FPGA acceleration, playing well to the devices' strengths in image processing. Our demonstrator thus takes a key part of Dimensional Imaging's own software and accelerates it using FPGA hardware. Two versions of the demonstrator were produced – an embedded version which can connect to a live camera rig and provide on-the-fly image combination, and a batch version designed to process large numbers of image pairs from video frames.

This latter version is designed to run across all 64 FPGAs of Maxwell and provides the next step-up in complexity. While as trivially parallel as the MCopt application this demonstrator has real data requirements – large digital images must be managed and streamed through the FPGAs in an efficient manner to ensure overall performance.

5.3 CSEM modeling

Our final demonstrator is another commercial code, this time in the area of oil and gas exploration. OHM plc are an Aberdeen-based consultancy offering services to the oil and gas industry [4]. OHM specializes in a form of simulation called controlled source electromagnetic modeling, a technique which uses the conductive properties of materials to analyse pieces of the seabed in the search for oil or gas reserves [6].

OHM's three-dimensional CSEM code provides the basis for our final demonstrator. Already parallelized using MPI, this is a 'classic' HPC application involving large data sets representing physical spaces and fields, double-precision arithmetic and iterative numerical methods for performing linear algebra operations on large matrices and vectors.

6. Initial performance results

This section presents our preliminary results from early tests of the three demonstrator applications on Maxwell. Unfortunately at the time of writing this section is only partially complete since the final versions of the demonstrators are still undergoing test.

All results quoted in this section were run on Maxwell. The quoted CPU results are thus for the 2.8 GHz Intel Xeon processors in the IBM blades. In caption legends we use the label 'AD' to refer to the Alpha Data hosted FPGAs, and 'NT' to refer to the Nallatech hosted devices.

6.1 MCopt

MCopt is the simplest of the demonstrators, a trivially-parallel engine to explore the parameter space of a typical option pricing calculation.

Our tests for MCopt aim to explore the five-dimensional parameter space defined by the variable input parameters to the Monte Carlo version of the Black-Scholes model (S, K, r_c, σ, $N_{timesteps}$). Our test draws 100,000 data samples from this parameter space; we fix the number of Monte Carlo iterations, N_{runs}, to 10,000 for each sample.

The single-node performance is shown in Figure 2, and Figure 3 shows MCopt run across 1 to 16 nodes, both CPU and FPGA

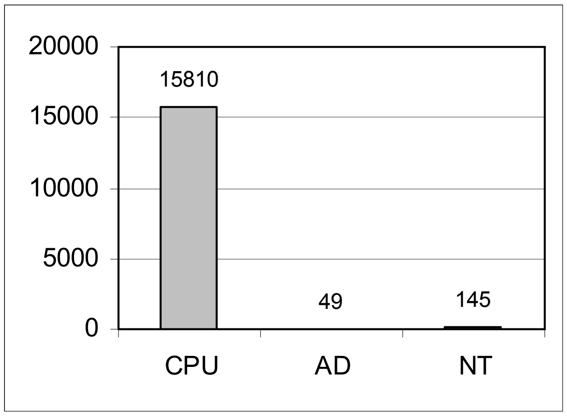

Figure 2. Single-node MCopt performance (s).

In Figure 3 the logarithmic scale belies the extreme scalability of the FPGA versions here: a batch of 100,000 parameters (prices, rates or some combination of these) that would take over 4 ½ hours on a Xeon blade runs in less than a minute on one FPGA, or less than 3 seconds on 16. As might be expected with such a simple calculation the FPGAs outperform the CPU by over two orders of magnitude – a factor of 300 in the Alpha Data case.

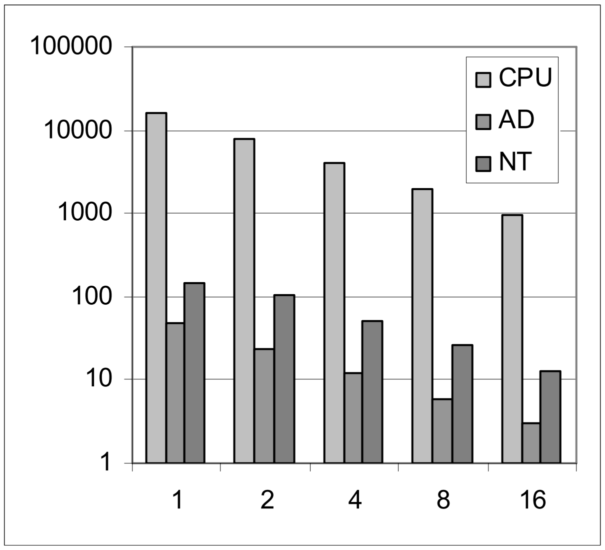

Figure 3. MCopt scaling performance on Maxwell (times in s). Note the scale is logarithmic.

6.2 Facial imaging

The facial imaging demonstrator, while still trivially parallel in execution, is more challenging than the MCopt application because of its data requirements.

Our tests here involve the batch processing of 32 pairs of video still images – 64 in all – each around 150 kB in size. This represents a little over a second of three-dimensional video.

The images are read in from networked disk, so this is also an interesting test of the network and i/o overheads of the parallel system.

Figure 4 shows the single node performance for these tests. Note that the FPGA times are at this stage estimates based on earlier versions of the hardware bitstreams. These figures suggest a factor of 6 or more speedup per node for the FPGAs.

Figure 5 plots the runtime for the same test against increasing numbers of nodes. Initially we show only the CPU times; hardware timings are not yet available.

The final plot in Figure 6 shows the speedup curve from the scaling results. Note the falloff towards 16 and 32 processors for the CPU (software-only version) – we suspect that this is a feature of i/o bandwidth saturation across the machine.

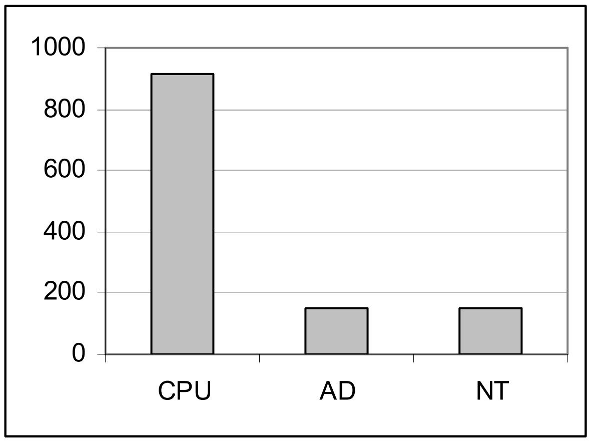

Figure 4. Single node facial imaging performance (times in s). The AD and NT times are projected from earlier hardware versions and are not final.

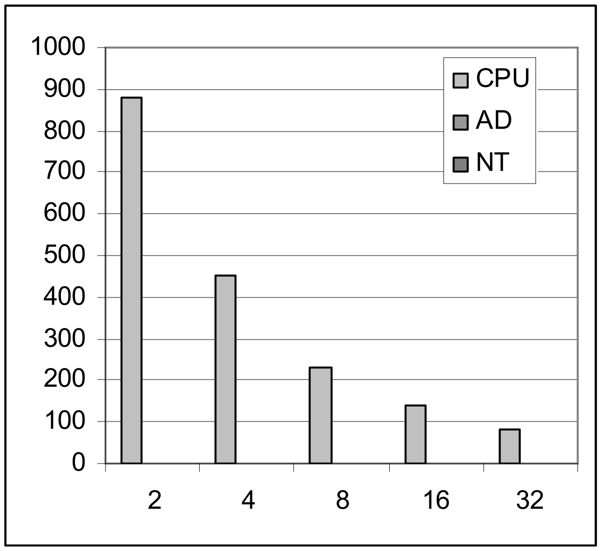

Figure 5. Facial imaging scaling performance on Maxwell (times in s). Note the AD are NT times are not yet available.

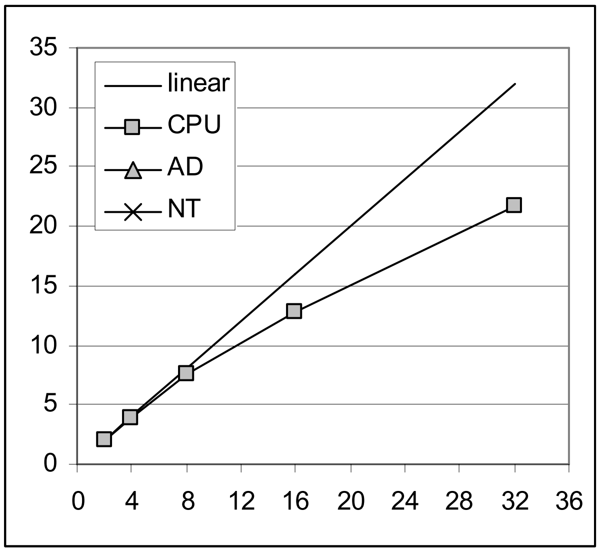

Figure 6. Parallel speedup of the facial imaging code on Maxwell. AD and NT results not yet available.

6.3 CSEM modeling

The most challenging of the demonstrators is the CSEM modeling application, a fully-parallelised classic HPC simulation. Here we used a sample dataset of size 50×45×200 (450,000 points) and ran the solver until it converged on a solution.

We estimate the single-node performance in Figure 7. Note that this is an estimate based on component timings – such are the memory requirements of this code that it is unable to run on fewer than 4 CPUs.

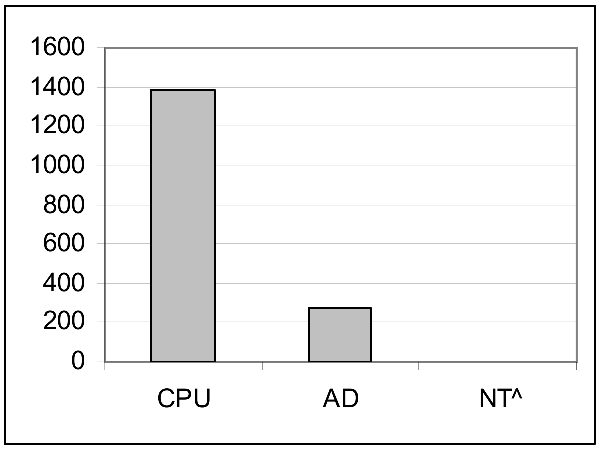

Figure 7. CSEM single-node performance in seconds (estimated). ^ Note that Nallatech figures are unavailable at time of writing.

Preliminary results suggest the FPGA versions run at least 5 times faster per node than the pure software version.

Figure 8 shows a scaling plot against number of nodes. Here the CPU times are accurately measured but the FPGA times are either projected (in the case of Alpha Data, based on our analysis of the single node performance, and assuming the same scaling as the software), or not yet available (in the Nallatech case).

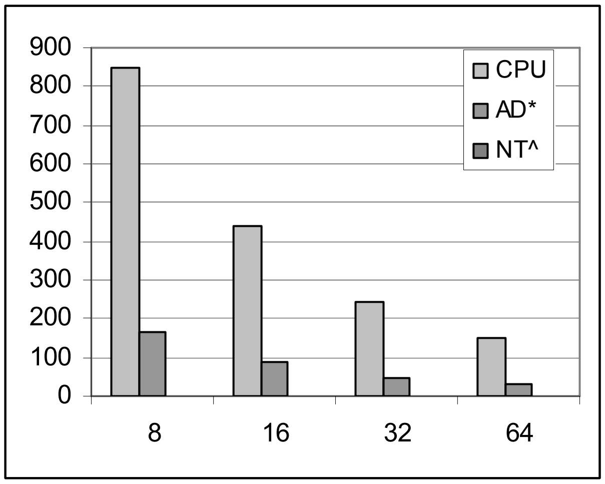

Figure 8. CSEM scaling performance on Maxwell (times in s). *AD times are estimated; ^NT times are not yet available.

Figure 9 plots parallel speedup (time on one node divided by time on N nodes) where the theoretical linear best case is shown.

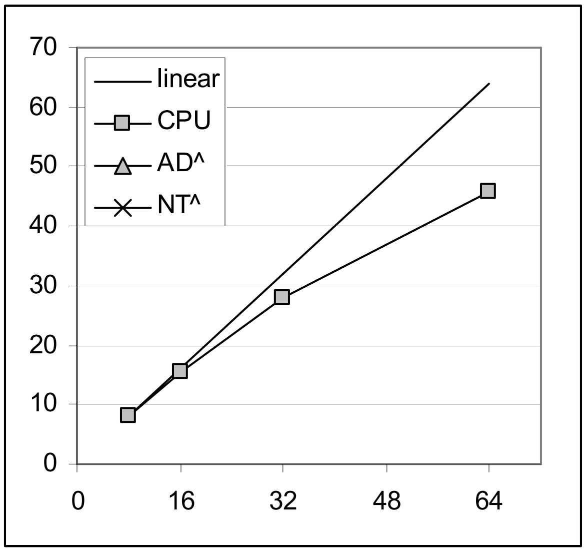

Figure 9. Parallel speedup of the CSEM code on Maxwell, rebased to 8 on 8 nodes. ^ AD and NT results not yet available.

Unfortunately the lack of data at this stage leaves us conjecturing about the possible parallel performance of Maxwell. In terms of single-node performance for CSEM we see around a factor of 5 improvement in the FPGA version over the software. We believe that with further memory optimizations this could rise to a factor of 8.

In terms of scaling, we anticipate the FPGA versions will certainly be no worse than the CPU versions, and could in fact scale better. Our reasoning here is that the communication patterns of FPGA-to-FPGA data transfer are very different to the MPI send/receive patterns of CPU-to-CPU transfer. In software terms, overlapping useful computation with communication overheads is the goal of efficient parallel computing but is, in practice, hard to achieve. It is rare that HPC codes do not follow the pattern compute–communicate–compute. The CSEM code is no exception – this is visible in the fall away from linear scaling as communications overheads begin to show.

In the FPGA core algorithms communication between devices occurs through the Rocket IO MGTs. The MGTs are represented at VHDL level as interfaces in just the same way as external memory banks are. As far as the core is concerned, it sees no difference in talking to an MGT interface as it does to a memory interface. Thus, provided the FPGA code can be optimized to ensure that data flow through the interfaces and cores when they are needed, there is no longer an explicit communications step. Data flow uninterrupted and the algorithm does not suffer – at least in theory – from the same communications overheads as the software version.

Thus, while we are sadly short of hard data at time of writing, we are optimistic in our expectations of parallel performance.

7. The future

Maxwell now exists, a 32-way IBM Bladecentre containing 64 Xilinx Virtex-4s configured in two flavours. Initial performance results show that it has not suffered too badly from its 'general purpose' nature, and indeed suggest that 'general purpose FPGA computer' is at least not an oxymoron!

The goals of the FHPC Alliance are now to promote the machine as a computing resource to both business and academia. The FHPCA Technology Translator is an activity coordinated by EPCC to disseminate the results of the supercomputer project and attract potential new users to FPGA-based HPRC.

In addition we intend to explore ways to improve the programmability of Maxwell and FPGA-based systems in general. While the three demonstrators here show that FPGAs *can* deliver genuine performance benefits even for memory-bound HPC simulation codes, the hardware accelerations did not write themselves.

Realising significant benefit does still require collaboration between software and hardware engineers. The PTK has been a useful development in providing high-level vendor-neutral application interfaces and common methods of configuration and job launching but its standardizing approach does not go deep enough into the software stack.

The HPRC community needs now to turn its attention to standards at all levels to help cement FPGAs into the new fabric of high-performance computing.

8. References

[1] R. Baxter *et al*, "High-Performance Reconfigurable Computing – the View from Edinburgh", *Proc. AHS2007 Conf.,* Second NASA/ESA Conference on Adaptive Hardware and Systems, Edinburgh, 2007.
[2] The FHPCA, www.fhpca.org
[3] F. Black & M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, Vol. 81, pp. 637-654.
[4] Dimensional Imaging, http://www.di3d.com/
[5] OHM plc, http://www.ohmsurveys.com/
[6] L. MacGregor, D. Andreis, J. Tomlinson & N. Barker, "Controlled-source electromagnetic imaging on the Nuggets-1 reservoir", *The Leading Edge*; August 2006; v. 25; no. 8; pp. 984-992