Price-Performance Aspects of Accelerating the FDTD Method Using the Vector Processing Programming Paradigm on GPU and Multi-Core Clusters

Robert G. Ilgner and David B. Davidson

Department of Electrical and Electronic Engineering
University of Stellenbosch, Matieland, Private Bag X1, Western Cape, South Africa
bobilgner@gmail.com, davidson@sun.ac.za

Abstract ― The parallelization of the FDTD on GPUs has become popular due to the low cost, low power and high compute performance achieved with these devices. In recent years, manufacturers of multi-core processors have enhanced the vector processing capability inherent in conventional processing cores, to the extent that these are now contributing considerably to the acceleration of the FDTD and competing with GPUs. This paper will compare the power consumption and purchase cost versus the performance benefits of several parallel FDTD implementations, in order to quantify the effect of parallelizing the FDTD using various processing paradigms. The purchase cost of hardware, computational performance and power consumption are used to compare the parallel FDTD deployments on the BlueGene/P, GPU clusters and the multi-core clusters using SSE. It is shown that the deployment of the parallel FDTD using a hybrid programming paradigm achieves the best computational performance for the lowest purchase cost and power consumption.

Index Terms – AVX, cluster, FDTD, GPU, multi-core, performance, SSE and vector processing.

I. INTRODUCTION

The Finite Difference Time Domain (FDTD) method is inherently highly parallel and has been accelerated by coding the FDTD in parallel form on a variety of platforms. Contemporary examples included the BlueGene/P, multi-core clusters and clusters using Graphical Processing Units (GPU). These systems can all achieve similar computational throughput, depending on the scale of the system. Upfront purchase cost and power consumption are crucial considerations when evaluating high performance computing hardware. Recent years have seen a sustained effort by manufacturers to reduce both, but nonetheless maintain an increase in computational performance [1].

This paper will compare the performance related cost of purchase and power consumed by the FDTD implementations on three different high performance computing architectures. To avoid the effect of scaling on the comparison, computational performance, price and power consumption will be normalized on a per core basis. The FDTD will be implemented using a task parallel method on the BlueGene/P and the multi-core clusters and by a combination of data parallel and task parallel methods; i.e., hybrid methods on the GPU cluster and multi-core clusters. The multi-core clusters make use of their Vector Arithmetic Logic Units (VALU), such as the Streaming SIMD Extensions (SSE) or the Advanced Vector Extensions (AVX), to achieve good performance for the FDTD. The computational performance of these parallelised FDTD implementations will be used to compare the cost and power consumption for the FDTD on high performance computers on a per core basis. These results are the main contribution of this paper, as most results in the literature for parallel FDTD deployments do not compare across entirely different architectures. With the exception of results for an i7-3960x and the GPU cluster (which are taken from the literature), results to be compared were obtained from parallel code

Submitted On: June 18, 2013
Accepted On: April 21, 2014

developed and deployed by the present authors in C on the different platforms. As such, another contribution made by this work is in removing the bias, which can be associated with different programming styles, programming languages and differing test data.

II. PARALLEL FDTD ON HIGH PERFORMANCE COMPUTERS

A. The FDTD method as a computational load

The FDTD method is based on a finite differences approximation of Maxwell's equations in both the time and spatial domains [2]. The computation is performed on a lattice of electric grid values offset by one half a grid spacing from the magnetic grid values. The FDTD remains the most widely-used time domain based Computational Electromagnetic (CEM) solver and overall is one of the most widely-used CEM methods

In the serial form of the FDTD process, program profiling shows that typically more than 98% of the process load is dominated by the repeated calculation of the updated FDTD grid. These calculations are also referred to as iterations, or sometimes leap-frogging and comprise several sets of three level deep computational loops (in three dimensions) [2].

The FDTD process is readily decomposed for parallelisation in either a task parallel or data parallel sense, as described by the methods that follow next.

In this paper, a compute node usually consists of several processors, each processor having several cores, following widely-used terminology in the high performance computing field.

B. Techniques used to parallelise the FDTD method.

Accelerating the FDTD on different architectures requires that the method of parallelisation matches the architecture of the High Performance Computing (HPC) system. There are currently three principal methods used to parallelise the FDTD on specific architectures. These are:

1) Message Passing Interface (MPI) - appropriate for a multi-core cluster or BlueGene [3]. The FDTD is parallelised as independent processing threads that will exchange FDTD grid data between iteration cycles. The MPI interface provides the communication protocol for the exchange of FDTD grid data fringes between successive iterations of the FDTD process [3].
2) openMP - appropriate for multi-core processors or shared memory processors [4,5]. The FDTD is parallelised on the basis of loop parallelism. openMP is a parametric language construct that is embedded in the program code before compilation. The openMP directive spawns FDTD threads that operate in the same memory space.
3) Vector processors or vector processor like architectures - i.e., GPU or vector registers, such as SSE or AVX [6,7,8,9,10,11]. The FDTD grid data is presented to the GPU's streaming multi-core processor as a data array for processing in an SIMD-like manner. Good performance can be achieved by optimizing the SIMD processing on the vector devices by adhering to best practice rules for data alignment and coalescence of the processing cores with the data being computed. Although the processing of the FDTD with a SIMD processor has gained great popularity over recent years with the emergence of the GPU, SIMD processing of the FDTD has been well documented in the literature dating back nearly 20 years [12].

In terms of programming complexity and effort required to code the FDTD with these methods, the rating in Table 1 is based on the subjective experience of programming the FDTD on the respective CHPC systems. A rating of 1 is "best," i.e., implies least effort to parallelise.

Table 1: Coding effort required to parallelise the FDTD

Technique	Implementation Effort
MPI	2
openMP	1
GPU	4
VALU	3

In practice, contemporary HPC systems are a combination of several of these basic architectures. The method with which the FDTD is implemented on these architectures reflects this. As an example, consider the FDTD implemented on a GPU cluster. The GPU cluster hardware

architecture consists of a collection of multi-core nodes using GPUs as acceleration devices. The corresponding parallel FDTD program consists of a data parallel program processing the FDTD on the GPU nodes, with an MPI process providing the communication between the multi-core compute nodes.

III. PARALLEL FDTD IMPLEMENTATIONS

The parallel FDTD implementations on the BlueGene/P, Xeon 5670 cluster and the discrete GPUs described here have been deployed by the present authors on various computing systems of the national South African Centre for High Performance Computing (CHPC), located in Cape Town, South Africa, using code developed by the authors. The results of these implementations are compared to those from a variety of publications as referenced.

The size of the models processed in this paper are limited by the memory available to each node and the extent to which the physical hardware has been scaled. A 32 bit system is obviously at a disadvantage in this respect, in that it will require many more nodes to process the same data volume as a 64 bit system. None the less, all of the multi node systems examined here, process models in the order of several billion grid points.

A. Parallel FDTD on the BlueGene/P

The BlueGene/P is a collective of compute nodes that communicate via a sophisticated interconnect system, as is shown schematically in Fig. 1. The communication interconnect between the compute nodes can be configured to reflect different processing strategies and topologies. The four core PowerPC 450 processors available on this machine each have access to two GB of memory. A schematic of this assembly is shown in Fig. 1.

The BlueGene/P allows the interconnect to be configured to map the topology of the hardware to the requirements of the FDTD. The Torus memory interconnect is particularly suitable for the processing of the 3D FDTD, as the physical topology reflects the structure of the FDTD in a 3D cubic mesh [13,14].

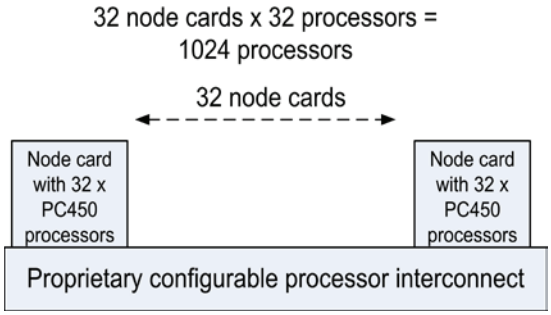

Fig. 1. BlueGene/P architecture as seen by the FDTD application.

The parallel FDTD is built using the MPI application interface. Equal FDTD grid allocations are computed on each processor, one MPI thread per core. Some implementations make use of the shared memory nature of the BlueGene's processing nodes, to process the FDTD grids local to these nodes with the openMP method [14,15]. The efficiency of the FDTD implemented on the BlueGene/P, as described in this paper, is compared to an implementation undertaken for the BlueGene/L (the earlier model of the system) in Fig. 2. The efficiency is calculated as (noting that "ideal" implies linear speed-up):

$$Efficiency(\%) = \frac{Measured\ FDTD\ cell\ throughput}{Ideal\ FDTD\ cell\ throughput} \times 100 \quad (1)$$

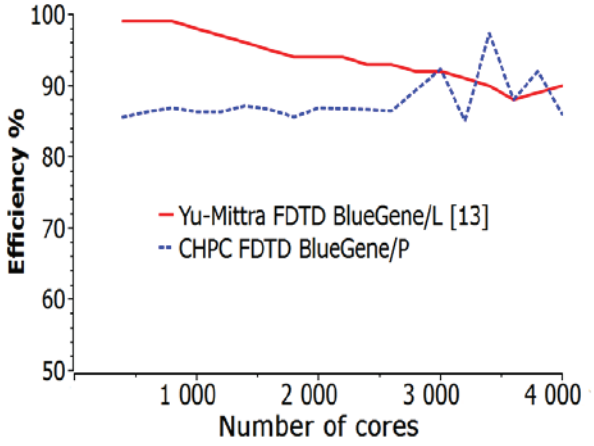

Fig. 2. Efficiency of the FDTD on two BlueGene models with a FDTD grid of two billion cells.

B. Parallel FDTD on the multi-core cluster

The Xeon 5670 cluster computer has the simplest architecture of all the HPC platforms compared here. This cluster is a collection of processors in a blade configuration, as shown schematically in Fig. 3. The blades communicate via a local area network system, the physical network structure comprised by a system of Quad Data Rate (QDR) Infiniband switches (note that some clusters use Ethernet as the interconnect fabric, but proprietary systems such as Infiniband are usually much faster.) The blade configurations are comprised of multi-core chips, which are in themselves tightly coupled using the Quick Path Interconnect (QPI) fabric [16]. It is worthwhile noting that the 6275 Blade configuration allows the memory to be accessed via three physical channels, a feature which greatly improves the access to memory [17] and reduces the data bottleneck experienced by the parallel FDTD on multi-core processors. The QPI fabric will provide cache coherency between the processing cores on all of the CPUs [16] connected with QPI.

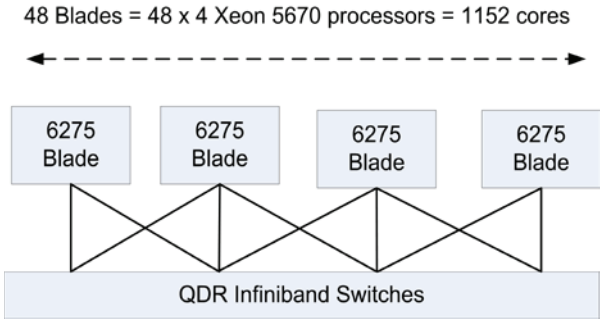

Fig. 3. Schematic architecture of the CHPC multi-core cluster.

All of the blades on the network will contend for network communication bandwidth, as required by the process running on the system. The Infiniband connection does not possess any switching logic to provide features such as parallel memory access to one thread, as is found in interconnects such as the one used by the BlueGene/P.

For this paper, the FDTD is implemented on the clusters using the Message Passing Interface (MPI) threading. The shared memory architecture of the multi-core processors has also been exploited by some [4] to produce hybrid implementations of the FDTD using both the openMP and MPI threading methods.

Figure 4 shows a comparison of the efficiency achieved by the parallel FDTD on a Xeon 5670 multi-core cluster at the CHPC, parallelized using the MPI method.

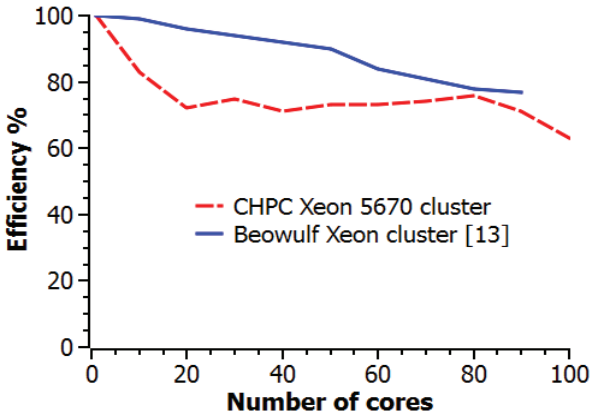

Fig. 4. A comparison of the efficiency of the parallel FDTD with MPI on a cluster at the CHPC with results from [13].

C. Parallelisation using Vector registers on multi-core processors

The SSE and Advanced Vector Extensions (AVX) are also described as Vector Arithmetic Logical Units (VALU) in some publications. Their functionality is based on a collection of vector registers resident on the die of contemporary microprocessor cores. The SSE and AVX registers allow the parallelisation of the FDTD in a data parallel manner [10], by using a Single Instruction Multiple Data (SIMD) approach, as is shown schematically in Fig. 5; i.e., the SSE and AVX can respectively process four or eight single precision floating point values simultaneously [18]. Figure 6 illustrates the benefit in terms of FDTD throughput when processing with AVX on a contemporary four core processor. Figure 7 demonstrates the performance improvement when using SSE on the CHPC's Xeon 5670 clusters. Also shown in Figure 7 is an implementation of the FDTD on a cluster of i7-3960x multi-core processors with the AVX functionality [8]. One particular feature of the i7-3960x's architecture is the presence of four dedicated memory channels supplying data to processors using the processing cores, thereby mitigating the effect of memory

bandwidth bottlenecking associated with processing of large FDTD data sets.

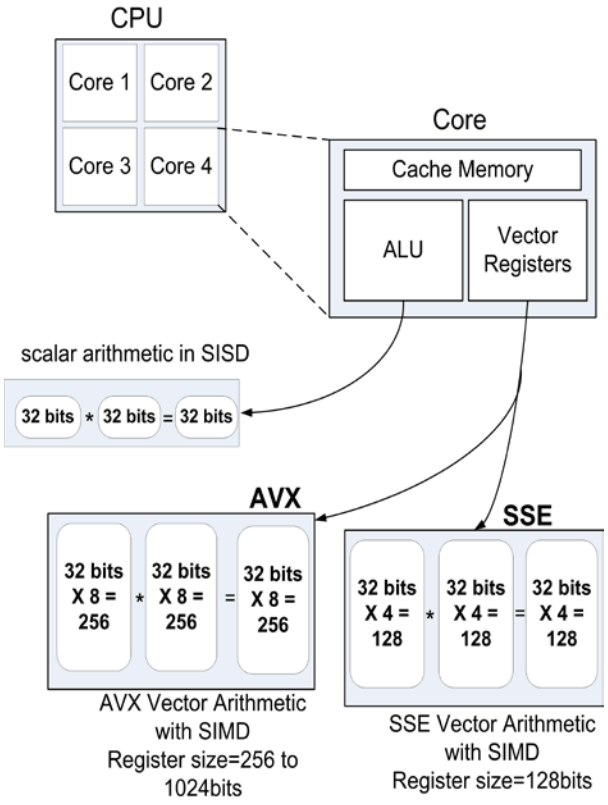

Fig. 5. A comparison of AVX and SSE on a multi-core processor.

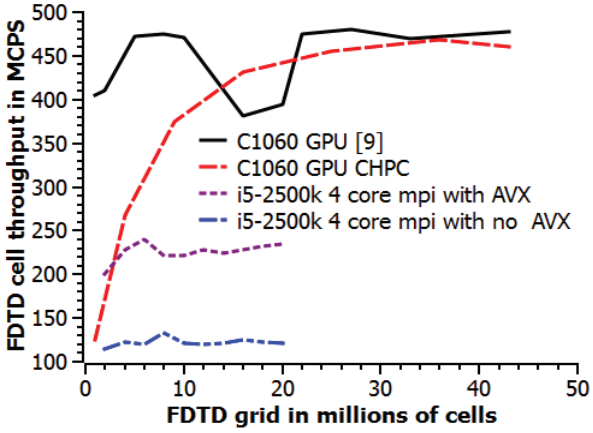

Fig. 6. A comparison of the acceleration of the FDTD on a GPU and on a four core processor using the AVX functionality.

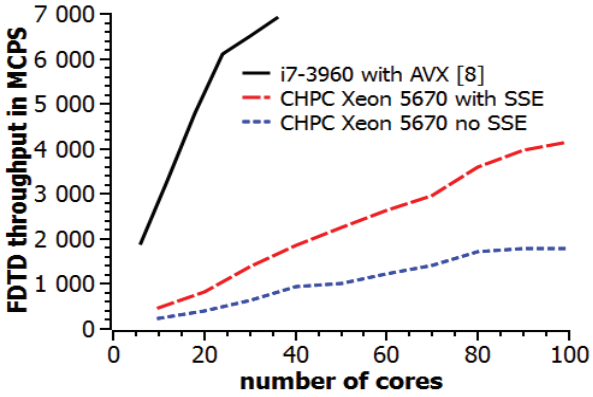

Fig. 7. Comparing AVX and SSE on a cluster.

D. Parallel FDTD on the GPU cluster

The discrete GPU is typically configured to a host computer via a Peripheral Component Interconnect express (PCIe) bus, as is shown in Fig. 8. The cluster GPU architecture shown in Fig. 10 is in effect, still a multi-core cluster architecture and uses the S870 GPU boards as accelerators connected to the multi-core worker nodes.

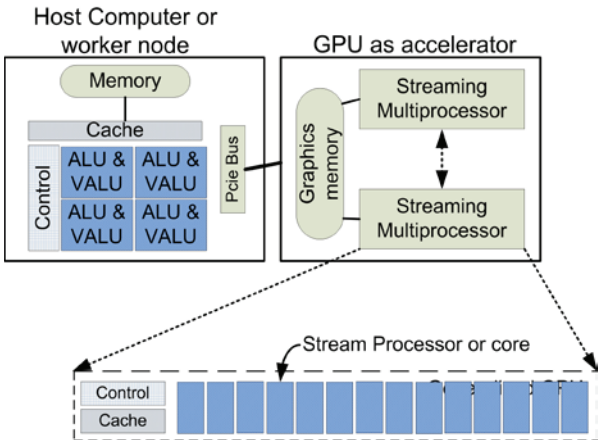

Fig. 8. Discrete GPU attached to host computer consisting of a four core multi-core processor.

The programming of the GPU allows some programming flexibility over and above that provided by the VALU technology [11,19], in that each GPU core or Stream Processor (SP) in Nvidia terminology, is allocated a dedicated program thread [20,21]. All cores in a Streaming Multiprocessor (SM) will execute the same thread.

Figure 8 illustrates the physical relationship between the SP and the SM. Nvidia has coined the term Single Instruction Multiple Threads (SIMT) to differentiate this style of programming from the purist SIMD. The difference between SIMT and SIMD, is that with SIMT one can use conditional (IF) logic within the SIMT program flow. The discrete GPU on its own can achieve good FDTD throughput, as is shown in Figs. 6 and 9. The two GPUs compared in Fig. 9 are NVIDIA's C1060 and the C2070 GPUs. The C2070 has 448 stream processors and 6 GB of global memory, whereas the C1060 has 240 stream processors, as shown in the architectural sketch of Fig. 8. The GPU is programmed in a data parallel sense and optimal performance is gained by the alignment and coalescence of data with the processing cores [7,9,22,23,24,25,26].

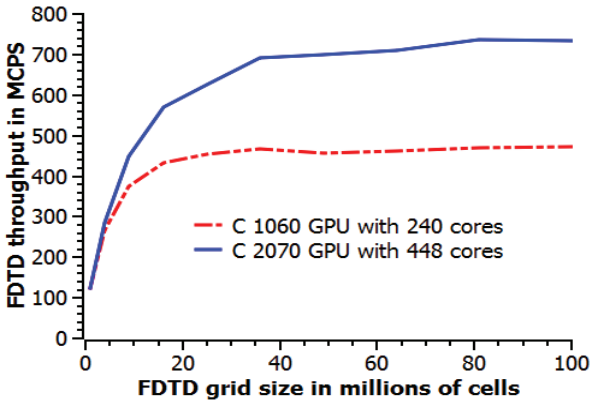

Fig. 9. Performance comparison of the FDTD on two different GPUs.

The architecture in Fig. 10 illustrates the configuration of multiple GPUs attached to a worker node or host PC. The multiple GPUs are attached to the worker node via a dedicated switch, which uses a single PCIe form connection. The CHPC achieves a similar architectural configuration, although individual GPUs are attached to the worker node by dedicated PCIe form factors; i.e., the worker node uses multiple PCIe channels to accommodate several GPUs directly. For these results the GPU used is Nvidia's S870 node, a collection of four low power GPUs, each with 128 stream processors. Power consumption per S870 is a low 800 watt.

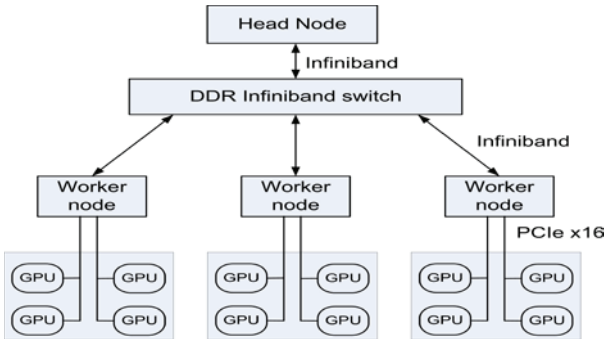

Fig. 10. Schematic of a cluster using GPUs as FDTD accelerators.

The processing of the FDTD on a cluster of GPUs overcomes the memory limitation of processing the FDTD on a single discrete GPU. All of the FDTD data processed by a discrete GPU needs to be transferred on to the GPU's physically dedicated memory before the computation commences. A comparison of the data sizes that the systems described in this paper are capable of processing, is shown in Fig. 11. It must also be born in mind that the data processing capability of a cluster will depend on the scale of a specific system.

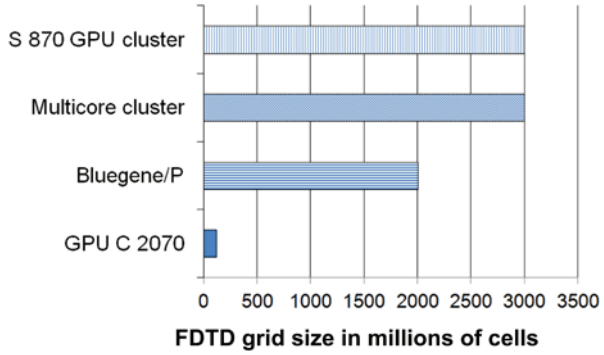

Fig. 11. A comparison of the processing capability of various systems described in the article.

The GPU and multi-core clusters compared in Fig. 12 show very similar performance characteristics for a low number of cores. Both systems compared here show a reduction in efficiency as the number of cores deployed on increases. Of note is the performance of the i7-3960x cluster, which achieves the greatest FDTD throughput per core, as is listed in Table 3.The reduction in efficiency as the size of a system

increases is very probably due to the communication overhead between a larger number of processors.

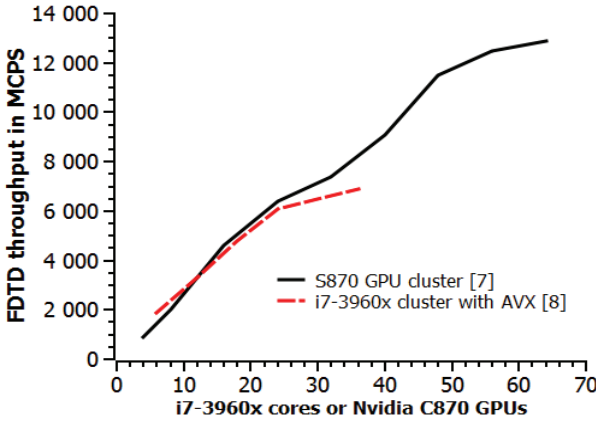

Fig. 12. FDTD throughput compared for a GPU cluster and a cluster using AVX optimised multi-cores.

IV. PERFORMANCE COMPARISON

A. Performance summary

The performance summary in Table 2 is an indication of the computational performance of the parallel FDTD implementations discussed in this paper. The rating used for the performance is the FDTD throughput on the respective platforms, in Millions of Cells Per Second (MCPS). In order to compare the performance on these platforms, the peak MCPS has been normalized on a per core basis, as is shown in Table 3.

Table 2: Computational throughput of the FDTD method on various computing platforms

Platform	Cores	Peak MCPS	Peak MCPS Per Core	Release Date
BlueGene/P	4096	8900	2.2	June 2007
S870 node cluster	8192	12 900	1.6	Mar. 2008
x5670 SSE cluster	100	4155	41.6	Mar. 2010
i7-3960x AVX cluster	36	6900	192	Nov 2011

B. Price and performance

Typical cost of purchase, pricing of HPC systems are shown in Table 3. These pricings were obtained from the Centre of High Performance Computing in Cape Town. The prices given in Table 3 are approximates for the year 2012/2013. Prices have been normalized to the peak FDTD throughput in MCPS per core.

Table 3: Performance/purchase cost comparison. CPS=FDTD grid cells per second.

Platform	Cost In USD 2012	Price Per Core In USD	Peak MCPS	Peak CPS Per USD
BlueGene/P	750000	183	8900	11900
S870 node cluster	46000	6	12900	262000
x5670 cluster	37000	370	4155	112000
i7-3960x cluster	15000	416	6900	462000

The normalized prices show that best computational performance per dollar is achieved by the optimized FDTD using the AVX on the i7-3960x multi-core cluster [8]. The GPU cluster lies in second position and is a better proposition than the BlueGene/P, when considering processing performance for the FDTD. It is not surprising that these performance ratings are all roughly ordered with the age of the hardware, as this accords with Koomey's law.

Two pricing considerations that have not been factored into the study are the maintenance costs for the HPC systems and for the system life expectancy before it is overtaken by newer technology. According to Koomey's law [1], the life cycle of a HPC system may currently be only one to two years before it is overtaken by newer technology.

Large computing systems require sophisticated servicing and sub systems, such as cooling racks and clean rooms. The costs attached to a maintenance plan for a HPC system in terms of capital expenditure and in terms of dependence on a specific vendor are not included in this paper.

C. Power consumption per core

Table 4 shows that for the systems compared in this paper, the multi-core cluster at the CHPC consumes the least power for the performance achieved when processing the parallel FDTD. The variation in the normalized power consumption between the lowest and highest consumers is not large and is a reflection of the low power design criterion for all of these systems.

Table 4: Performance/power comparison

Platform	System Power (kW)	Cores	Power Per Core (W)	MCPS Per W
BlueGene/P	31.5	4096	7.7	0.285
S870 node cluster	16	8192	2.0	0.860
x5670 cluster	4	100	40	1.004
i7-3960x cluster	2	36	56	0.571

The power consumption is a limiting factor for the size of the computing system from the aspect of being able to dissipate enough heat without physically destroying the computing hardware. Larger clusters will need dedicated cooling systems.

V. CONCLUSION

The performance of four parallel FDTD implementations, different in hardware and programming technique, have been compared to show that FDTD deployments accelerated with a combination of vector processing paradigm and the MPI threading interface, presently provide the best performance for cost and best performance for power consumed. However, the comparisons are of course subject to Koomey's law - it would have been instructive to repeat the comparison with a BlueGene/Q, had one been available to the authors (the very high cost of these systems means they are often not readily available, of course). Even with this note, the price/performance and power/performance results are not conclusive because whilst the acceleration of the FDTD on the i7-3960x cluster provides the best performance, it does so at a cost of consuming 50% more power per core than the Xeon 5670 cluster. This work also provides a framework for the comparison of the FDTD on nascent low power technology, such as the Intel Phi.

It should also be noted that memory is another major constraint for the FDTD and one which GPUs have been slow to overcome, due to the relatively small amount of RAM usually associated with a GPU.

ACKNOWLEDGMENT

DBD and RGI acknowledge the support of SKA South Africa, the South African Research Chairs Initiative of the Department of Science and Technology (DST), National Research Foundation (NRF) and the Centre of High Performance Computing.

REFERENCES

[1] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, pp. 46-54, 2011.

[2] A. Taflove and S. C. Hagness, "Computational electrodynamics: the finite-difference time-domain method," Third Edition*, Artech House*, chapters 3-7, 2005.

[3] C. Guiffaut and K. Mahdjoubi, "A parallel FDTD algorithm using the MPI library," *IEEE Antennas and Propagation Magazine*, vol. 43, no 2, pp. 94-103, April 2001.

[4] D. Luebke, White Paper, "Nvidia® GPU architecture and implications," 2007.

[5] openMP website: available at *http://www.openMP.org*.

[6] W. Yu, X. Yang, Y. Liu, and R. Mittra, "A novel hardware acceleration technique for high performance parallel conformal FDTD method," *27th Annual Review of Progress in Applied Computational Electromagnetics (ACES)*, Virginia, USA, pp. 903-908, March 2011.

[7] S. E. Krakiwsky, L. E. Tumer, and M. M. Okoniewski, "Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU)," *IEEE MTT-S International Microwave Symposium Digest*, vol. 2, pp. 1033-1036, June 2004.

[8] W. Simon, A. Lauer, and A. Wien, "FDTD simulations with 10^{11} unknowns using AVX and SSD on a consumer PC," *IEEE Antennas and Propagation Society International Symposium (APSURSI),* Chicago, IL, USA, pp. 1-2, July 2012.

[9] V. Demir and A. Z. Elsherbeni, "Programming finite-difference time-domain for graphics processor units using compute unified device

architecture," *IEEE Antennas and P ropagation Society International Symposium*, Toronto, Ontario, Canada, July 2010.

[10] L. Zhang, X. Yang, and W. Yu, "Enhanced parallel FDTD method using SSE instruction sets," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 27, no. 1, pp. 1-8, January 2012.

[11] W. Yu and W. Li, "An enhanced hardware acceleration FDTD technique for parallel signal line simulation," *28th Annual Review of Progress in Applied Computational Electromagnetics (ACES)*, Ohio, USA, pp. 411-416, April 2012.

[12] D. B. Davidson and R. W. Ziolkowski, "A connection machine implementation of a three dimensional parallel finite difference time-domain code for electromagnetic field simulation," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 8, pp. 221-232, 1995.

[13] W. Yu, X. Yang, Y. Liu, L. Ma, T. Su, N. Huang, R. Mittra, R. Maaskant, Y. Lu, Q. Che, R. Lu, and Z. Su, "A new direction in computational electromagnetics: solving large problems using the parallel FDTD on the bluegene/l supercomputer providing teraflop-level performance," *IEEE Antennas and Propagation Magazine*, vol. 50, no. 2, pp. 26-41, April 2008.

[14] W. Yu, M. Hashemi, R. Mittra, D. de Araujo, M. Cases, N. Pham, E. Matoglu, P. Patel, and B. Herrman, "Massively parallel conformal FDTD on a bluegene supercomputer," *IEEE Int. Conf. on Systems, Man and Cybernetic*, San Antonia, Texas, 2009.

[15] M. F. Su, I. El-Kady, D. Bader, and Y. Lin, "A novel FDTD application featuring openMP-MPI hybrid parallelization," *IEEE International Conference on Parallel Processing*, Montreal, QC, Canada, 2004.

[16] R. Maddox, G. Singh, and R. Safranek, "Weaving high performance multiprocessor fabric," *Intel Press*, chapters 1-2, 2009.

[17] U. Drepper, "What every programmer should know about memory," *Swiss Federal Institute of Technology*, 2007, available at: http://www.akkadia.org/drepper/cpumemory.pdf.

[18] W. Yu, X. Yang, Y. Liu, R. Mittra, J. Wang, and W. Yin, "Advanced features to enhance the FDTD method in GEMS simulation software package," *IEEE International Symposium on A ntennas and Propagation (APSURSI)*, Washington, USA, pp. 2728-2731, July 2011.

[19] V. Demir, "An algorithm to improve solution efficiency of FDFD method on GPU," *28th Annual Review of Progress in Applied Computational Electromagnetics (ACES)*, Ohio, USA, pp. 364-369, April 2012.

[20] V. Demir and A. Z. Elsherbeni, "CUDA-openGL interoperability to visualize electromagnetic fields calculated by FDTD," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 27, no. 2, pp. 206-214, February 2012.

[21] V. Demir, "A stacking scheme to improve the efficiency of finite-difference time-domain solutions on graphics processing units," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 25, no. 4, pp. 323-330, April 2010.

[22] "CUDA programming manual," available at: http://www/nvidia.com.

[23] J. Stack and Jr., "Accelerating the finite difference time domain (FDTD) method with CUDA," *27th Annual Review of Progress in Applied Computational Electromagnetics (ACES)*, Virginia, USA, pp. 897-902, March 2011.

[24] M. Weldon, L. Maxwell, D. Cyca, M. Hughes, C. Whelan, and M. Okoniewski, "A practical look at GPU-accelerated FDTD performance," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 25, no. 4, pp. 314-322, April 2010.

[25] M. Ujaldon, "Using GPUs for accelerating electromagnetic simulations," *Applied Computational Electromagnetics Society (ACES) Journal*, vol. 25, no. 4, pp. 294-302, April 2010.

[26] J. Stack, B. Suchoski, and J. Infantolino, "CUDA implementation of moving window finite difference time domain," *28th Annual Review of Progress in Applied Computational Electromagnetics (ACES)*, Ohio, USA, pp. 300-304, April 2012.

Robert G. Ilgner obtained his B.Sc. and B.Sc. (Hons) degrees in Geophysics from the University of Witwatersrand in 1982 and 1983, respectively. As a Geophysicist he conducted geophysical exploration surveys for mining houses. He then worked in London for Siemens in the Information Technology industry building large database systems and received his M.Sc. in 1991 from the University of Surrey in Guildford, UK. He was awarded his Ph.D. from the University of Stellenbosch in 2013. He was employed by Schlumberger in the Seismic division, creating parallel processing applications used for Seismic data reduction and modeling. He built software for the creation of panoramic images and advertising on the internet. He is currently a Postdoctoral Researcher at the University of Stellenbosch.

David B. Davidson received his B.Eng., B.Eng. (Hons) and M.Eng. degrees (all cum laude) from the University of Pretoria, South Africa, in 1982, 1983 and 1986, respectively. He received his Ph.D. degree from the University of Stellenbosch, Stellenbosch, South Africa, in 1991. In 1988, he joined the University of Stellenbosch. As of 2011, he holds the South African Research Chair in Electromagnetic Systems and EMI Mitigation for SKA there. He has held a number of visiting appointments, including at the University of Arizona; Cambridge University, England; Delft University of Technology, The Netherlands and the University of Manchester, England. His main research interest through most of his career has been Computational Electromagnetics (CEM) and he has published extensively on this topic. He is the author of "Computational Electromagnetics for RF and Microwave Engineering" (Cambridge, U.K.: Cambridge Univ. Press, 1st ed., 2005, 2nd ed., 2011). Recently, his interests have expanded to include engineering electromagnetics for radio astronomy. Davidson is a Fellow of the IEEE and a member of the South African Institute of Electrical Engineers and the Applied Computational Electromagnetic Society. He was a recipient of the South African FRD (now NRF) President's Award in 1996. He received the Rector's Award for Excellent Research from the University of Stellenbosch in 2005. He is the editor of the "EM Programmer's Notebook" column of the IEEE Antennas and Propagation Magazine. He was Chair of the local organizing committee of ICEAA'12-IEEE APWC-EEIS'12, held in Cape Town in September 2012.