Understanding the Efficient Parallelisation of Cellular Automata on CPU and GPGPU Hardware

Mike J Gibson
University of Exeter
North Park Road, Exeter,
EX4 4QF.
(+44)1392 724075
Mg278@exeter.ac.uk

Ed C Keedwell
University of Exeter
North Park Road, Exeter,
EX4 4QF.
(+44)1392 724014
E.C.Keedwell@exeter.ac.uk

Dragan Savić
University of Exeter
North Park Road, Exeter,
EX4 4QF.
(+44)1392 723637
D.Savic@exeter.ac.uk

ABSTRACT
Cellular automata, represented by a discrete set of elements are ideal candidates for parallelisation, particularly on graphics cards using GPGPU technology. This paper shows that the speedups of 50 times over CPU are possible but that the hardware is only partially responsible and the memory model is vital to exploiting the additional computational power of the GPU.

Categories and Subject Descriptors
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) – *Single-instruction-stream, multiple-data-stream processors (SIMD).*

General Terms
Algorithms, Measurement, Performance, Experimentation, Languages, and Verification.

Keywords
Cellular Automata, CA, General Purpose Graphical Processing Unit, GPGPU, OpenCL.

1. INTRODUCTION
Modern hardware is becoming increasingly parallel as shown by the fact that off-the-shelf CPUs are now equipped with 2 to 8 processing cores and modern graphical processing units (GPUs) now have thousands of cores. Through the recent introduction of General Purpose-Graphic Processing Units (GPGPUs), programmers may take advantage of the expanding hardware parallelism of Graphics Processing Units (GPU) for general computation. The latest Nvidia GPGPUs have up to 3072 processing cores [1], and as such GPGPUs demonstrate even greater parallelism than their CPU counterparts. The multi-core nature of most modern CPUs is allowing Moore's law to continue to hold true despite stagnation in individual core speed [2]. Several sources suggest that co-processors like GPGPUs, with their inherently Massively Multi-Core (MMC) nature, are increasing in performance faster than their CPU counterparts [3].

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
ACM 978-1-4503-1964-5/13/07.

2. CELLULAR AUTOMATA
Cellular Automata (CA) were first conceived by John Von Neumann in the 1940's [4] as a model of the universe, following a set of natural laws. The automata consist of a lattice (grid) of cells commonly arranged in a 2D regular grid, with each cell being in one of a number of states. Each cell is updated using state transition rules (a rule set), that uses the current state of that cell and its adjacent neighbours in a particular pattern to determine the state of that cell in the next iteration of the algorithm. Each cell is updated in an idealised parallel sense to form a new lattice of cell states, and the process is repeated for a number of iterations.

3. RELATED WORKS
There appears to be few attempts in the literature to develop parallel CA algorithms and to investigate how exactly they will interact with MMC technologies. There are a number of examples of implementations presented, however few of these investigate the spread of possible speed-ups, or how the variation of the CA's base parameters (e.g. lattice size, number of generation, number of states/data types, neighbourhood sizes, rule complexity) affects these speed ups. A representative approach to the use of CPU and GPU computing to speed up CA execution is described in [5]; and indeed the literature presents a wide variety of speed up factors, with little explanation of the features that affect this, other than the levels of hardware parallelism.

4. METHOD
Tests are performed using the well-studied 'game of life' rule set developed by John Conway in the late 1970s [4], which has 2 states and a Moore neighbourhood. Since memory look-up is relatively expensive on the GPGPU; a programmatic function is used here (a series of C if-statements) which forms the basis of a decision tree. The GPGPU present three memory types including 'global memory' which is essentially the RAM on the GPU card; 'local memory' which is on-chip and therefore much faster but limited in space and scope to a single compute core; and finally 'image memory' (sometimes referred to as texture memory) which is memory specific to the native task of the GPGPU as a graphics processor and is cache-lined even in older models as well as having special hardware for dealing with border conditions. As image data commonly has four channels (Red, Green, Blue, Alpha), the GPGPU hardware is specially set up to deal with and transfer vectors of values, most commonly with a length of four. This is used to create a novel hybrid of texture memory and vectorisation, in which the lattice is folded into four quarters, much as one would imagine folding a square of paper. A combination of the texture memories' ability to deal with border conditions, and swizzling operations (which re-order vectors efficiently in hardware) are used to make near-optimal use of the

hardware features present in the GPU. In this way the wider memory lanes of the GPGPGU are used to effectively allow the transfer of four values in parallel, saving additional memory accessing time; also reducing the number of threads required. Finally, tests are also performed with vectorisation (folding) and global memory.

5. EXPERIMENTAL SETUP

Table 1. Full specifications of GPGPU's tested [6] [7].

Machine	Machine A	Machine B
Type	Dell XPS M1530 laptop	PC workstation
CPU	Intel Core2 Duo T8100 @ 2.1GHz	Intel Core-I7-2600 @ 3.4Ghz
GPGPU	Nvidia GeForce 8600M	Nvidia GTX 560 Ti
CPU / GPU cores	2 / 32	8 / 384

The new open-standard language/API OpenCL is used, designed specifically for parallel hardware such as the GPGPU, as well as multi-core CPUs. For detailed information the reader is referred to the OpenCL specification [8].

6. EXPERIMENTAL RESULTS

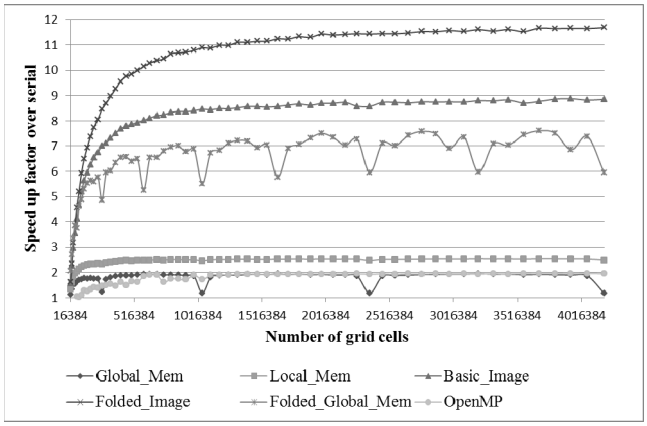

Figure 1 - Machine A, Speed ups over CPU serial implementation for parallel CPU (OpenMP), and the OpenCL memory algorithms on the GPGPU.

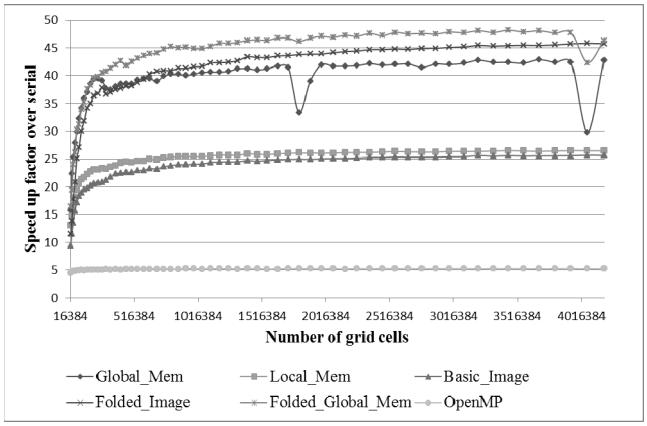

Figure 2 - Machine B, Speed ups over CPU serial implementation for parallel CPU (OpenMP), and OpenCL memory algorithms on the GPGPU.

7. CONCLUSION

Firstly it can be seen that a large enough grid size is required in order to overcome the overheads of parallelisation which on the GPGPU included small amount of compilation time and transfer down the known bottleneck of the PCI connection. Also seen are periodic reductions in performance which are due to loading balancing by the hardware between the number of available cores and the specific number and processing times of individual threads/cells.

Figures 1 and 2 show a marked difference in performance between the two tested machines, due to the introduction of cache-lined global memory in the Fermi (Machine B) generation of GPGPU's. Local and image memory gain greater speed-ups than global memory alone on Machine A, whereas on Machine B local and image memories are less efficient than global memories due to the more efficient caching and the need to explicitly copy data to the local and image memories. When vectorisation (folding) is applied both global and image memories show an increase in performance. For Machine A, the vectorised image/texture memory performance best, but for Machine B, it is the vectorised global memory that is the top performer. This is due to the more efficient cache-line global memory of the Fermi chip with Machine B.

8. REFERENCES

[1] Nvidia, "GeForce GTX 690 specification," Nvidia, 2012. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-690/specifications. [Accessed September 2012].

[2] P. Richmond, D. Walker, S. Coakley and D. Romano, "High performance cellular level agent-based simulation with FLAME for the GPU.," *Briefings in Bioinformatics,* vol. 11, no. 3, pp. 334 - 347, 2010.

[3] J. Ning, H. Xu, B. Wu, L. Zeng, S. Li and Y. Xiong, "MCA-based Animation of Fracturing Heterogeneous Objects," in *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on*, Jinan, 2012.

[4] B. Chopard and M. Droz, Cellular Automata Modeling of Physical Systems, Cambrige University Press UK, 1998.

[5] M. R. Lopez-Torres, J. L. Guisado, F. Jimenez-Morales and F. Diaz-del-Rio, "GPU-based cellular automata simulations of laser dynamics," jornadassarteco.org, Seville, Spain, 2012.

[6] Nvidia, "GeForce 8600M specification," Nvidia, 2012. [Online]. Available: http://www.nvidia.com/object/geforce_8600M.html. [Accessed 2012].

[7] Nvidia, "GeForce GTX 560 Ti," Nvidia, 2012. [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-560ti/specifications. [Accessed 2012].

[8] "OpenCL 1.2 Specification," Khronos group, 2012. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf. [Accessed March 2012].