JOURNAL OF ENGINEERING RESEARCH AND APPLIED SCIENCES (JERAS)
5th Edition - May 2018, (Hoon – Libya), **ISSN 2414-6515**

ON OPENCL AND CHAOTIC PHENOMENA

Mohamed Khalifa, Ali Alsaadi , and Ali Alkhathlan
Department of Computer Science & Engineering
University of Colorado Denver, USA
Email : moh.khalifa@sebhau.edu.ly

Abstract
GPGPU (General Programming on Graphics Processing Units) vastly improves over traditional methods of computation by utilizing the massive parallelism available through graphics programming units. This project aims to utilize GPGPU through the use of OpenCL (Open Computing Language), an open standard for computing on heterogeneous platforms, including CPUs, GPUs, and even embedded devices, for the computation involved in Chaotic Phenomena, and in particular, the Mandelbrot Set and the bifurcation diagram of the Logistic Map. The performance provided by the GPU through OpenCL will be compared to CPU performance through OpenCL, and a naïve serial implementation as control.

1 INTRODUCTION

GRAPHICS processing units (GPUs) (colloquially termed video cards) are conventionally used in the computation and rendering of graphics in modern computers. These computations, such as the application of shaders on a set of graphics primitives, involve the repeated computation of a fixed set of procedures on each element of a large dataset – an easy target for parallelization. In accordance with this perception, modern GPUs consist of dozens of cores, each containing multiple ALUs, which each contain multiple instruction streams, and which again each contain multiple concurrent fragments, leading to speeds on the order 1 TFLOPs (for comparison, a high-end CPU achieve speeds on the order of 100 GFLOPs, an order of magnitude difference) [11].

To make use of this large amount of computing power, a recent trend called General Programming on Graphics Processing Units (GPGPU) emerged. GPGPU involves the usage of graphics processing units for nongraphics programming (general programming). Graphics routines such as shaders and fragments are subverted for use in general programming, allowing for easily-accessible massively parallel computation at relatively low-cost. Recent successes in GPGPU include fast computation of k-nearest neighbor search for machine learning [4], sorting in databases [7], AES encryption [5], regular expression matching [14], and the implementation of an antivirus engine [13].

However, traditionally, GPGPU requires a deep and low-level understanding of the workings of a GPU, rendering it inaccessible to the vast majority of computer programmers. To combat this, new technologies such as Nvidia CUDA [16] and OpenCL [10] have emerged to simplify the development of GPGPU programs. These technologies work by designing a kernel and handing it to the GPU to execute concurrently on a large set of data. The kernel is a set of instructions to be performed on each datum, but is usually limited to non-branching code, although modern GPUs have added branching (although at a large performance penalty). CUDA and OpenCL simplify this by

allowing programmers to code in a C-based language.

We seek to make use of the processing power made available by these recent developments in the computations involved in investigating chaotic phenomena. As a benchmark, we specifically target the computation of the Mandelbrot set [12], as well as the bifurcation diagram of the logistic map [8]. Both of these chaotic phenomena (as well as many other forms of chaotic phenomena) involve parallel computation over a large space – an ideal target for speeding up via GPGPU.

This research paper will discuss the background required for this, as well as any foreseen merits and difficulties to this approach based on previous work in the literature, as well as the fundamental characteristics of the problem domain. Actual implementation and experimentation is deferred to the project paper.

2 ANALYZING THE PROBLEM

General Programming on Graphics Processing Units (GPGPU) provides massive parallelism through the use of graphics processing units (colloquially, video cards) for non-graphics related computation. This has traditionally been done by treating graphics primitives (such as vertices) as data, and then applying shaders on the graphics primitives to simulate general programming. Recently, however, frameworks have emerged to aid in GPGPU. One of the first to emerge is Nvidia CUDA [16], which is a GPGPU framework developed by Nvidia for exclusive use with Nvidia GPUs. Soon after the release of CUDA, OpenCL (Open Computing Language) was released, which is an open standard for heterogeneous computing in general, which includes CPUs, GPUs, and even embedded devices [10]. Whereas prior to the release of the frameworks, a programmer needed to have a low-level understanding of GPU architecture, the frameworks enabled programmers with little knowledge of GPU architecture to perform GPGPU.

The investigation of chaotic phenomena (the phenomena studied in Chaos Theory) requires heavy computation. However, much chaotic phenomena exhibit large amounts of data parallelism, in that the same computation is performed over and over on differing inputs. Prime examples of these are the Mandelbrot Set [11] and bifurcation diagram of the logistic map [1]; computation of the Mandelbrot Set involves repetitive application of the map

$$z = z^2 + c$$

On the points of the complex plane, while computation of the bifurcation diagram of the logistic map involves applying the logistic map

$$x = rx(1-x)$$

Repetitively until the logistic map converges to its long term iterates. The problem to be tackled by this project is the usage of GPGPU, through OpenCL, to perform the computations required to investigate chaotic phenomena, and in particular, the Mandelbrot Set and the bifurcation diagram of the logistic map.

2.1 Objectives:

a) Successful implementation of algorithms for the computation of the Mandelbrot Set and the bifurcation diagram of the logistic map, both in OpenCL and in plain C++.
b) Comparison of performance between OpenCL running on a GPU, OpenCL running on a CPU, and as control, a plain serial C++ implementation of the previously mentioned chaotic phenomena.
c) Analysis of the benefits and complications derived from the usage of OpenCL for the computation of chaotic phenomena.

2.2 Approach:

The investigation of the effectivity of OpenCL for the computation of chaotic phenomena will be performed by comparing implementations of the same algorithm for the computation of the Mandelbrot Set and the bifurcation diagram of the logistic map, first in C++, then in OpenCL on the GPU and on the CPU.

3 BACKGROUND (KEY CONCEPTS)

1) GPGPU and GPU Architecture:

GPGPU, or General Programming on Graphics Processing Units, is the usage of graphics processing units (video cards) for non-graphics-related computation. This is advantageous because graphics processing units allow for massive parallelism due to their architecture: since graphics processing units have traditionally been used for graphics-related computations, such as applying a Gaussian mask.

Graphics-related computations usually involve applying the same computation over a large set of graphical primitives, such as the convolution of the Gaussian kernel on an image. Thus, the architecture of graphics processing units are oriented towards performing tasks that involve data parallelism. Figure 1 depicts a sample GPU architecture.

Whereas an average CPU contains 2 to 8 cores, a GPU, such as the chip below from Nvidia [17], which is especially oriented towards GPGPU computation, contains 512 cores, each capable of SIMD (single instruction, multiple data) – in other words, a GPU has up to hundreds of cores (as compared to CPUs which have 8-16 at the most), and Each of those cores is capable of executing dozens of instruction streams as the same time. This, of course, comes at a cost: GPUs have few control structures, and no current GPUs contain advanced control structures such as branch prediction; even branching is a computationally expensive task. The general flow of computation on a GPU starts with the loading of data onto the GPU. This is often one of the most expensive operations, since the data has to travel through the bus. In normal GPU processing, this step has the transfer of graphics primitives, such as an

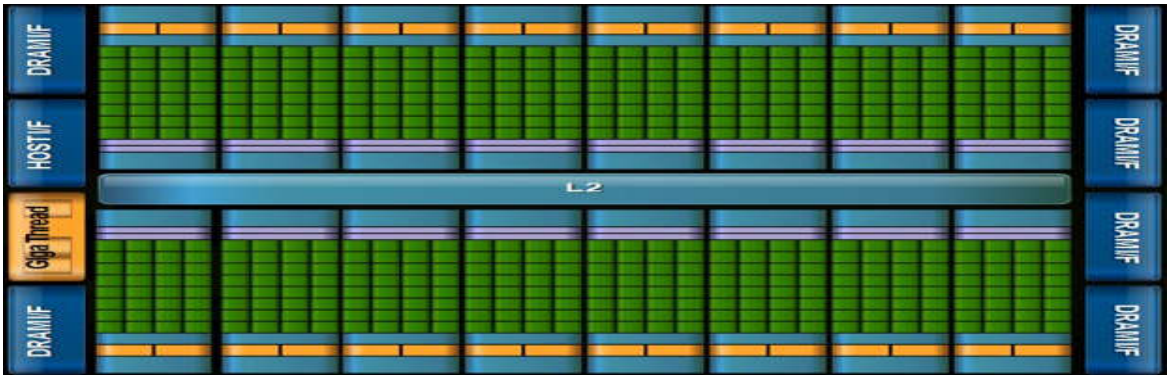

Fig. 1: Nvidia Fermi GPU architecture [17]

image to be rendered; in GPGPU, this is the transfer of the data to be operated on. After the data is loaded, the data is operated on using a set of operations – a shader, in traditional GPU terms. Whereas before, shaders were constrained to a fixed set, nowadays shaders are programmable – which is what enables GPGPU. In GPGPU, a shader is called a kernel instead, to reflect the more general scope. Afterwards, data is then transferred back to main memory, where the CPU can operate on it once again.

1) OpenCL:

OpenCL (Open Computing Language) is an open language and framework standard initially developed by Apple and now owned by the non-profit conglomeration Khronos Group [10]. OpenCL was designed for use in heterogeneous computing: a common framework for computation on CPUs, GPUs, and embedded systems such as FPGAs. The basic computation unit in OpenCL is the "kernel", which is a procedure meant to be performed repetitively on a dataset that is split into work units. Kernels in OpenCL are written in the OpenCL language, which is based on a slightly modified C99 for heterogeneous computing; namely, recursion and function pointers are absent from the OpenCL language, and the standard C library is replaced by the OpenCL library; special primitives are also present, such as multidimensional vectors [11]. OpenCL is an advancement over Nvidia CUDA, another framework [16]; Nvidia CUDA is a framework solely for Nvidia GPUs, and is thus limited in scope. All major hardware manufacturers support OpenCL, including Nvidia , Intel [6], and AMD/ATI [2].

2) Chaotic Phenomena:

Chaos Theory, the study of Chaotic Phenomena, was popularized among scientific circles by Yorke's 1975 paper, "Period Three Implies Chaos," [26] Mandelbrot's work on fractals [11], and Lorenz's work on weather simulation [3].
Chaotic Phenomena, such as the soon to be discussed Mandelbrot Set and the logistic map, are characterized by three properties: sensitivity to local conditions, topological mixing, and dense periodic orbits [1]. Briefly explained, sensitivity to local conditions (also known as the "Butterfly Effect") is the property that two points that are arbitrarily close to each other can have massively different trajectories, such as depicted in figure 2,

where nearly identical points produce massively different trajectories. Topological mixing, on the other hand, is the notion that any region in the phase space of a phenomenon will eventually overlap (mix) with other regions as the phenomenon evolves, while density of periodic orbits means that any point in the space of a phenomenon is arbitrarily lose to a point in a periodic orbit

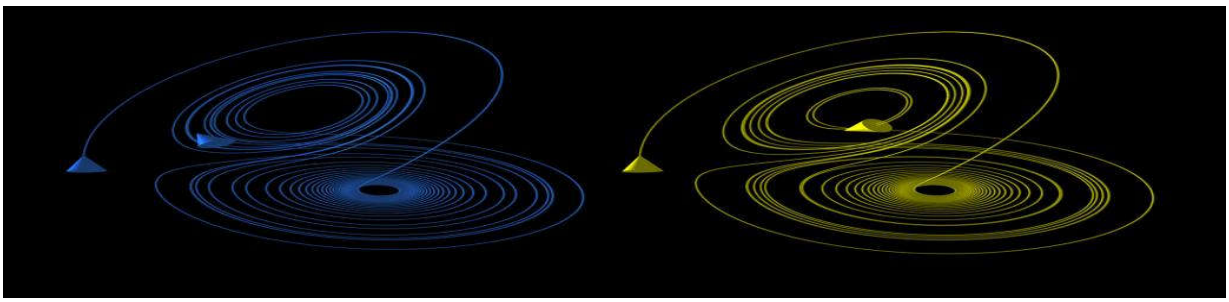

Fig. 2 Two Lorenz orbits starting from close initial points (the arrows on the left) [31]

2.3 Mandelbrot Set:

The Mandelbrot Set is the set of points in the complex plane that remain bounded after infinitely iterations of the map

$$z_{n+1} = z_n^2 + c,$$

Where $z_0 = c$ is the original point [12]. Note that once a point leaves the circle of radius 2

Centered on the origin of the complex plane (i.e., $|z_n| > 2$), the point will eventually escape to infinity. Thus, the Mandelbrot Set is often visualized by coloring the complex plane according to the number of iterations it takes for a point to escape the circle of radius 2 (a sample visualization is in figure 3).

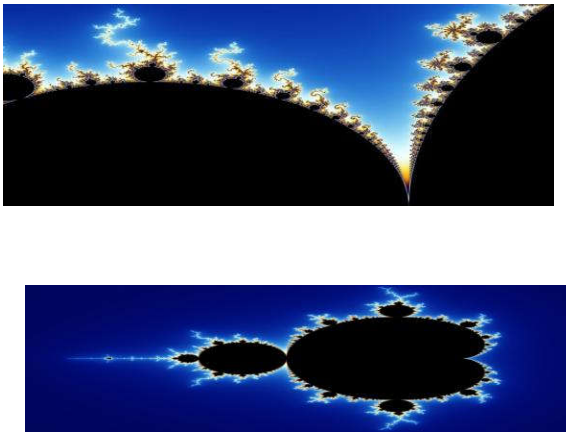

Fig. 3 The Mandelbrot Set [30]

This visualization gives insight to the structure of the Mandelbrot Set; namely, its infinite self-similarity, as can be seen in figures above.

The border of the Mandelbrot Set contains infinitely many replicas of the original Mandelbrot Set, and each replicas itself contains infinitely replicas.

The detail most essential to this project, however, is that the computation of the Mandelbrot Set is highly amenable to parallelization: the same map is performed on each point in the complex plane: a perfect example of data parallelism, and an excellent candidate for GPGPU.

2.4 Logistic Map:

The logistic map is the real map:

$$x = rx(1-x)$$

Where x is taken from 0 to 1, and r is a positive real number. The logistic map is a useful model of population growth, but it is now better known for its chaotic properties [1]. In particular, the bifurcation diagram of the logistic map is a plot of the long term iterates of the logistic map, with r varying. It can be shown that the long term iterates of the logistic map either oscillate among a fixed set of values, based on r, or are chaotic, as can be seen in figure below

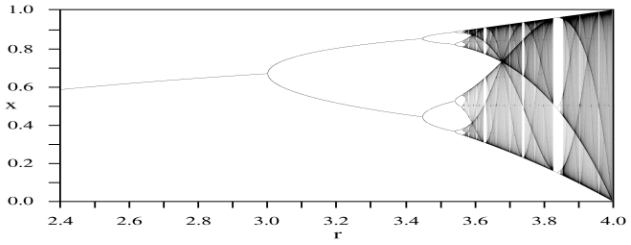

Fig. 4 Bifurcation diagram of the logistic map [32]

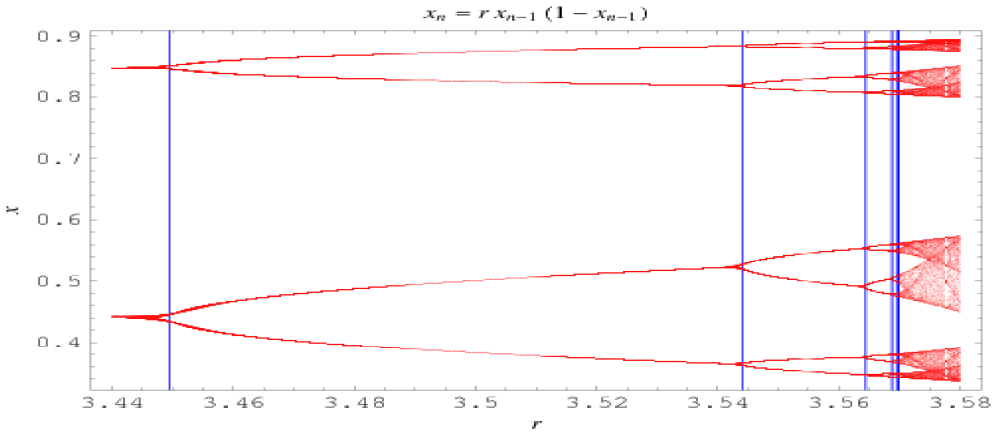

Fig. 5 Self-similarity of the bifurcation diagram of the logistic map [28]

2.5 Design

The Mandelbrot Set and the bifurcation diagram of the logistic map will each be implemented in both plain C++ and OpenCL. The implementations will be straightforward adaptations of the algorithms, and are detailed in the following section.

2.6 Implementation

The implementation of the Mandelbrot Set will be as follows: the 2-dimensional region of the Mandelbrot Set to be visualized will be divided into a 1024 x 1024 grid, each corresponding to a pixel in the visualization. Each pixel will then be iterated upon for up to 1024 iterations, or until it escapes the circle of radius 2. In the C++ implementation, this will be done using a simple for-loop, while this will be performed in an OpenCL kernel for the OpenCL implementation. The region will then be colored according to the number of iterations performed.The implementation of the bifurcation diagram of the logistic map will be similar: the 1-dimensional region of the logistic map (i.e., the interval of r values to take the long term iterates of) will be divided into 1024, each corresponding to a column of pixels in the final image. Each value of r will then be iterated upon with an initial value of x**=0.4** (as the initial value has no effect on the long term iterates) for 2^{20} iterations as warmup, then the values of the next 1024 iterations will then be recorded, and subsequently plotted onto the image. Once again, the iterations will be done as an ordinary for-loop in the C++ implementation, and as an OpenCL kernel for the OpenCL implementation.

3 METHODOLOGY

The performance of the implementations of the Mandelbrot Set and the bifurcation diagram of the logistic map will be measured by timing the implementations as they compute the Mandelbrot Set and the bifurcation diagram of the logistic map over a various set of fixed regions, and using the CPU and GPU drivers for the OpenCL implementation. The resulting diagrams will then be compared visually and numerically for any discrepancies (with the C++ implementation as the gold standard). The computation was performed on an Intel Core i5 Dual-Core M460 at 2.53 GHz on each processor, and an AMD Radeon Mobility HD 5145 (GPU version ATI RV710). The software used to compile and run the implementation are the Intel OpenCL driver [6], and the AMD OpenCL driver (branded as AMD Accelerated Parallel Processing) [2]. This was compiled on Visual C++ using Visual Studio 2010.

4 RESULTS AND DISCUSSION

i. Mandelbrot Set:

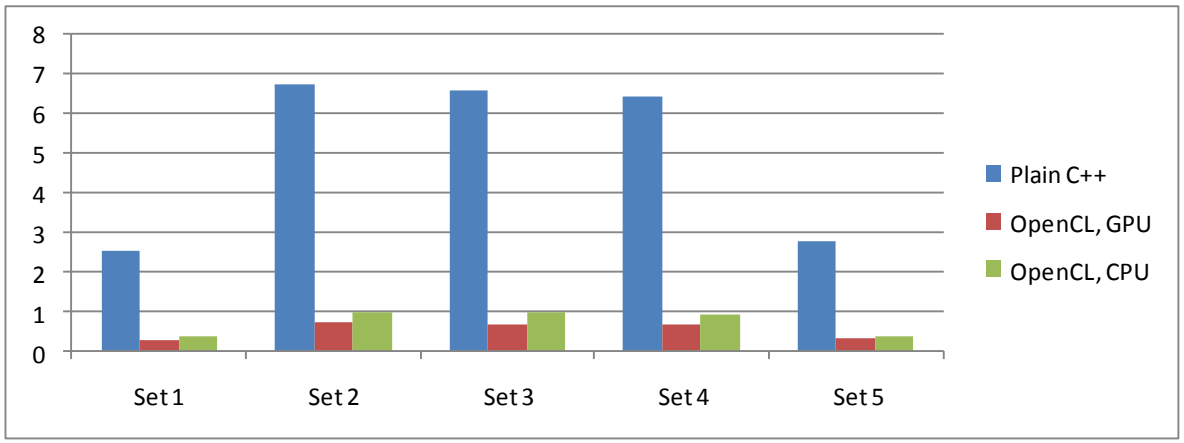

Fig. 6 Performance timings for the Mandelbrot Set implementations

Charted above in Figure 6 are the performance results for the different implementations of the Mandelbrot Set. We see that in all cases, the OpenCL implementation is faster than the plain C++ implementation by roughly 10 times (an order of magnitude). We also see that in all cases, the OpenCL implementation on the GPU driver runs faster than the OpenCL implementation on the CPU driver by roughly ¾ (with the notable exception of the last set, which may be considered an outlier).

Displayed below in figures 7 through 9 are the generated visualizations of the Mandelbrot Set. By inspection, they seem to be exactly the same, and the numerical comparison of the number of iterations for each visualized point confirms that

Fig. 7 Mandelbrot Set visualizations from C++

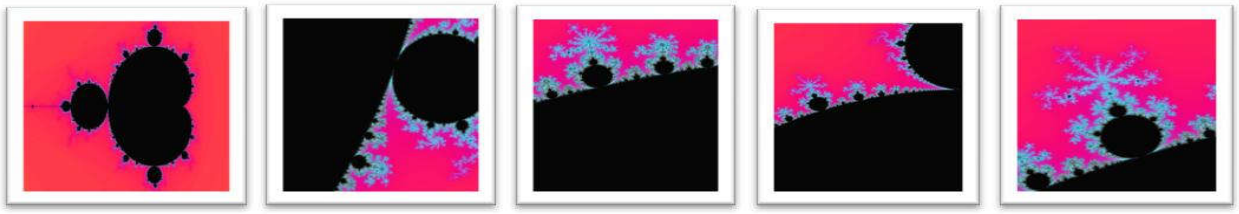

Fig. 8 Mandelbrot Set visualizations from OpenCL, GPU driver

Fig. 9 Mandelbrot Set Visualizations from OpenCL, CPU driver

Ii Logistic Map:

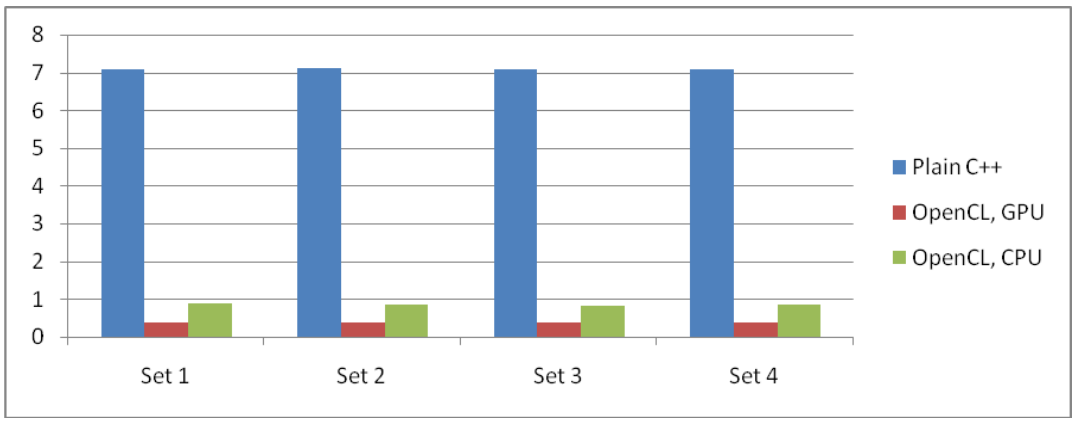

Fig. 10 Performance timings for the logistic map implementations

Similar results appear in the computation of the bifurcation diagram of the logistic map: in all cases, the OpenCL implementation outpaces the C++ implementation by an order of magnitude. We see, however, a more marked difference between the OpenCL implementation backed by the GPU driver and that backed by the CPU driver: the GPU driver takes half the time to perform the necessary computations. Plotted below in figures 11 through 13 are the generated diagrams by each implementation. Visual comparison shows that they are the same, and once again, numerical comparison confirms.

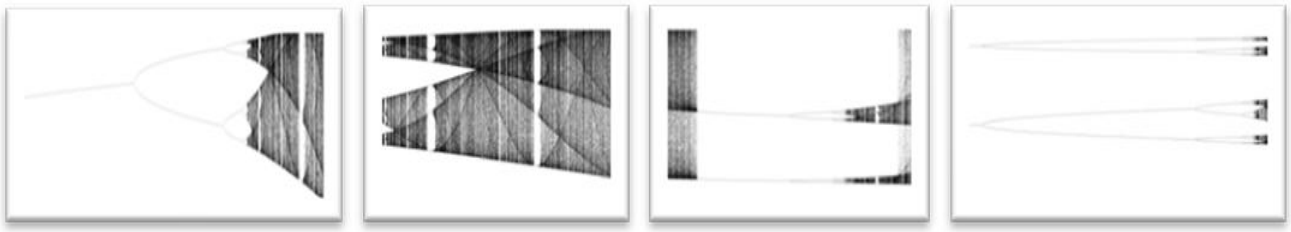

Fig. 11 Bifurcation diagram by plain C++ implementation

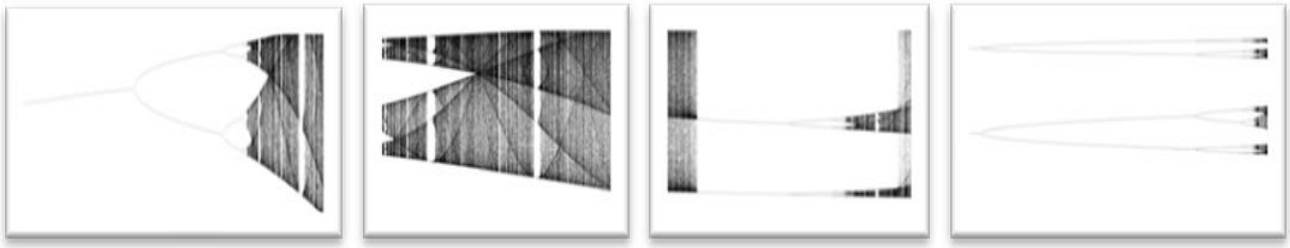

Fig. 12 Bifurcation diagram by OpenCL implementation, GPU driver

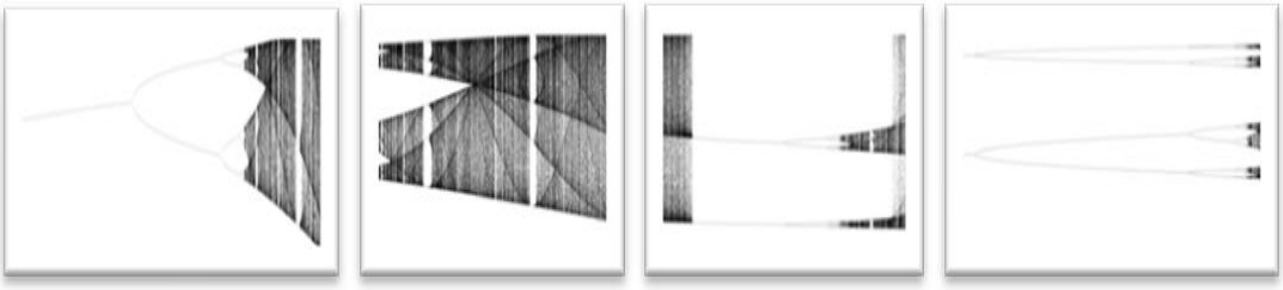

Fig. 13 Bifurcation diagram by OpenCL implementation, CPU driver

5 ANALYSIS

For both chaotic phenomena tested, the OpenCL implementation outperformed the plain (single-core) C++ implementation by an order of magnitude. Furthermore, the OpenCL implementations were direct adaptations of the plain C++ implementations, and are nearly identical with the exception of the OpenCL boilerplate. In particular, the same code was used for the OpenCL kernel and the loop body of the for-loops in the plain C++ implementation. Also, the same code was used for the OpenCL implementations on the GPU and the CPU. These together show the potential for large performance improvements with little effort by simply porting code from C++ to OpenCL. The GPU driver outperformed the CPU driver, but only by a factor of ¾ to ½, which is less than expected considering the parallelism of the problems. This may be due to the amount of time required to transfer data between the GPU and the CPU, coupled with the relatively small workload involved. This affects greatly the performance of the GPU relative to the CPU, and may perhaps be improved by increasing the workload of each work unit (thus reducing the number of times data has to be transferred).

The performance of the CPU driver itself is remarkable: the CPU used for this test was a dual-core Intel Core i5, and so one would expect an improvement in the range of ½ (since two core were used with OpenCL, instead of the single core using in the plain C++ implementation), rather than 1/10. This shows the massive improvements possible through OpenCL, even on the CPU, and can be explained by the ability of the Intel OpenCL driver to perform excellent optimizations, and usage of SSE instructions.

6 CONCLUSIONS

GPGPU provides massive parallelism through the unique architecture of GPUs. OpenCL gives easy access to this power, and more, in that OpenCL runs on CPUs and even embedded devices such as FPGAs as well as GPUs. On the other hand, chaotic phenomena require intense computation to investigate, and are often heavily parallelizable, like the Mandelbrot Set and the bifurcation diagram of the logistic map. Given the above, the possibility of using OpenCL for the computation of the two mentioned chaotic phenomena was investigated by creating both C++ and OpenCL implementations for the two. The performance and accuracy of the OpenCL implementations were measured by comparing them with the C++ implementation, and the results showed an order of magnitude improvement over the C++ implementation for both the CPU and GPU drivers. The GPU driver showed a ¾ to ½ time improvement over the CPU driver – smaller than expected, but this may be attributed to the time required to transfer data over the bus. The CPU driver's performance was remarkable in that a dual-core CPU was used, and thus only a ½ time improvement over the original C++ implementation was expected: this shows the potential of OpenCL, and the remarkable optimizations made by the Intel CPU driver.

7 REFERENCES

[1] Alligood, K. T., Sauer, T., and Yorke, J.A. Chaos: an introduction to dynamical systems. New York City, NY: Springer-Verlag, 1997. Print.

[2] "AMD Accelerated Parallel Processing SDK". AMD Developer Central. AMD, n.d. Web. 6 Mar 2012.

[3] Devaney, Robert L. An Introduction to Chaotic Dynamical Systems, 2nd ed,. Boulder, CO: Westview Press, 2003. Print.

[4] Garcia, V., E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. In Proceedings of

the CVPR Workshop on Computer Vision on GPU, 2008. Print.

[5] Harrison, Owen, and John Waldron. "AES on SM3.0 compliant GPUs." In Proceedings of CHES 2007. Print.

[6] "Intel® OpenCL SDK." Intel Visual Computing Source. Intel Corporation, n.d.. Web. 6 Mar 2012.

[7] Mancheril, Naju. "GPU-based Sorting in PostgreSQL." Thesis, School of Computer Science - Carnegie Mellon University. Print.

[8] Milnor, John W. Dynamics in One Complex Variable. 3rd ed. In Annals of Mathematics Studies 160. Princeton, NJ: Princeton University Press, 2006.

[9] "OpenCL." Nvidia Developer Zone. Nvidia, n.d. Web. 6 Mar 2012.

[10] "OpenCL." OpenCL. Khronos Group, n.d. Web. 6 Mar 2012.

[11] Scarpino, Matthew. OpenCL in Action. Greenwich, CT: Manning Publications, 2011. Print.

[12] Strogatz, Steven (2000). Nonlinear Dynamics and Chaos. Perseus Publishing.¬

[13] Vasiliadis, Giorgos, et al. "GrAVity: A Massively Parallel Antivirus Engine." In proceedings of RAID 2010. Print.

[14] Vasiliadis, Giorgos, et al. "Regular Expression Matching on Graphics Hardware for Intrusion Detection." In proceedings of RAID 2009. Print.

[16] "CUDA Zone." Nvidia Developer Zone. Nvidia. n.d. Web. 27 Mar 2012.

[17] "Next Generation CUDA Architecture, Code Named Fermi." Nvidia. n.d. Web. 27 Mar 2012.

[18] Friedrichs, M.S. et al. "Accelerating Molecular Dynamic Simulation on Graphics Processing Units". Journal of Computational Chemistry 30 (6): 864–72, 2009. Web. 27 Mar 2012.

[19] Pande, Vijay and Stanford University. "Folding@home." Stanford, California: Stanford University, 2012. Web. 27 Mar 2012.

[20] Pande, Vijay and Stanford University. "Folding@home team stats pages." Stanford, California: Stanford University, 2012. Web. 27 Mar 2012.

[21] Fung, et al. "Mediated Reality Using Computer Graphics Hardware for Computer Vision". In Proceedings of the International Symposium on Wearable Computing 2002 (ISWC2002), Seattle, WA, 7-10 2002, 83--89. Web. 27 Mar 2012.

[22] Harris, Mark. "Mapping computational concepts to GPUs." In ACM SIGGRAPH 2005 Courses (Los Angeles, California, 31 July – 4 August 2005). J. Fujii, Ed. SIGGRAPH '05. ACM Press, New York, NY, 50. Web. 27 Mar 2012.

[23] "About the Khronos Group." Khronos Group, n.d. Web. 27 Mar 2012.

[24] Fang, Jianbin, et al. "A Comprehensive Performance Comparison of CUDA and OpenCL." In Parallel Processing(ICPP), 2011 International Conference on 13-16 Sept. 2011. Web. 27 Mar 2012.

[25] Jääskeläinen, P.O. "OpenCL-based design methodology for application-specific processors." In Embedded Computer Systems (SAMOS), 2010

International Conference on, pp. 223- 230. Web. 27 Mar 2012.

[26] Li, T.Y.; Yorke, J.A. (1975). "Period Three Implies Chaos" (PDF). American Mathematical Monthly 82 (10): 985–92. Web. 27 Mar 2012.

[27] Farber, Rob. "Cuda, Supercomputing for the Masses: Part 17." In Dr. Dobb's, 14 Apr 2010. Web. 27 Mar 2012.

[28] Weisstein, Eric W. "Logistic Map." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/LogisticMap.html

[29] "Mandel zoom 01 head and shoulder.jpg.' Wikimedia Commons. Web. 27 Mar 2012.

[30] "Mandel zoom 00 mandelbrot set.jpg." Wikimedia Commons. Web. 27 Mar 2012.

[31] "TwoLorenzOrbits.jpg." Wikimedia Commons. Web. 27 Mar 2012.

[32] "Logistic Bifurcation map High Resolution.png." Wikimedia Commons. Web. 27 Mar 2012.