A Comparison of CPU and OpenCL Parallelization Methods for Correlation and Graph Layout Algorithms used in the Network Analysis of High Dimensional Data

Athanasios Theocharidis[1,*], Gibran Hemani[1], Michael Kargas[2] and Tom C. Freeman[1]

[1]The Roslin Institute and R(D)SVS, University of Edinburgh, Easter Bush, Midlothian, EH25 9RG, Scotland, UK

[2]University College London, 20 Maple St, London W1T 5HB, England, UK

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT

Motivation: Many algorithms used in analysis of high dimensional data require significant processing time due to the sheer number of values compared. We describe the results of the parallelization of two algorithms central to the functionality of the network analysis tool BioLayout *Express*^{3D}; the calculation of correlation (Pearson, Spearman Rank) coefficient matrices used to define relationships in large datasets, such as between gene expression profiles in microarray analyses and the Fruchterman-Rheingold graph layout algorithm used in the visualization of the resulting networks.

Results: Initially, the Java 1.6 and ANSI C99 languages were used to provide multithreaded implementations of these algorithms and to run on all available CPUs. Secondly, the OpenCL C language was used as part of the OpenCL API to harness the processing power of GPUs. Both approaches have been implemented using a platform and hardware independent approach. We discuss the issues associated with the parallelization of these very different algorithms and provide detailed comparisons of the results where we have achieved speed-ups of more than 60x times compared to non-parallel implementations.

Availability: The code is publicly available and utilized within the current release (version 2.0) of BioLayout *Express*^{3D} (www.biolayout.org).

Contact: a.theocharidis@roslin.ed.ac.uk

1 INTRODUCTION

The application of new technologies to study biological systems has reshaped our approach to the life sciences. DNA sequencing, high density microarrays for genotyping and expression analysis, high throughput proteomics and metabolomics and biological imaging platforms etc., are all leading to the accumulation of vast amounts of data. The scale of this data and its structure raise serious challenges as to its analysis. Whilst new and old mathematical algorithms are being applied to tackle this problem, the sheer scale of the data frequently requires the analyses to be farmed out to centralized computing facilities. Even then compute times can be long and there is a pressing need to find ways to optimize these computations (Eichner *et al.*, 2009; Varré *et al.*, 2011; Razmyslovich *et al.*, 2010; Dematté and Prandi, 2010; Hemani *et al.*, 2011).

Over the last decade computers have undergone a fundamental change with the widespread introduction of multicore central processing units (CPUs) even into standard desktop computer hardware (Geer, 2005). These processor architectures combine several standard processing units (PUs) physically in one chip and provide a lot of inexpensive processing power. More recently a similar revolution has been seen in the development of graphical processing units (GPUs) where many-core architectures using hundreds or even thousands of cores have evolved (Shalf, 2007). These were initially designed for graphics processing and use less complex but many more in numbers specialized SIMD (Single Instruction Multiple Data) PUs. The aim has been to minimize the control and cache complexity, and thus their occupying space on the GPU chip. More recently, GPUs have begun to be designed to be able to perform generic computations and new APIs, such as Nvidia's CUDA and Khronos's OpenCL, have been introduced to help software engineers to harness the potential power GPU computing. Indeed, the latest GPUs are beginning to be used as a form of a co-processor next to a main processor that orchestrates the parallel processes. The potential applications of GPU computing in biology are enormous and attempts to harness the power of these new computing hardware architectures for a variety of applications are progressing rapidly (Burkitt *et al.*, 2011; Blom et al., 2011; Liu *et al.*, 2011; Zhou J. *et al.*, 2011; Yung *et al.*, 2011; *Klingbeil et al.*, 2011; Chavent *et al.*, 2011; Zhou Y. *et al.*, 2011; Davis *et al.*, 2011).

Whilst the introduction of multicore CPUs and GPUs ostensibly improves processing power, not all algorithms can be successfully adapted to these architectures, and those that can, often require extensive modifications. Additionally, the individual nature of algorithms makes generic solutions to parallelization difficult, for instance algorithms that perform a lot of Input/Output (I/O) tasks have a lot of data dependencies and need synchronization techniques. However, algorithms that perform a huge number of similar computational tasks are primarily executed in memory and do not have many data dependencies (Eichner *et al.*, 2009). Where parallelized solutions have been implemented, it has frequently been done so on an ad hoc basis. To tackle these problems, new APIs have been introduced to support the efficient use of hardware from various vendors for parallelization in a platform independent manner. Currently, Khronos's OpenCL (Munshi and Gaster, 2011)

*To whom correspondence should be addressed.

is perhaps the best solution from the software industry for this problem, designed for multi-platform and cross-vendor hardware usage. However, due to its very recent development and still evolving specification, few examples of its use in the parallelization of scientific algorithms are available.

BioLayout *Express*^{3D} (BL*E*^{3D}) is a standalone, multiplatform application that has been designed for the visualization and analysis of large network graphs derived primarily, but not exclusively, from biological data (Enright and Ouzounis, 2001; Freeman *et al*., 2007; Theocharidis *et al*., 2009). Core to its functionality are a number of algorithms that facilitate the calculation of similarity measures (correlations) between sets of variables e.g. expression values measured by microarrays and the layout of graphs derived from these or other measures. When the size of the input data is large, standard microarrays commonly report on values from over 50,000 probe sets and a datasets may contain hundreds or thousands of samples, calculation of the correlation matrices is computationally intensive. Similarly, the layout of graphs composing of 10's of thousands of nodes connected by millions of edges is a large computational task. We have therefore sought to speed up the implementation of these algorithms and thereby improve the users' experience potentially making the construction and analysis of large network graphs a routine procedure. This work introduces parallelization results in Java 1.6, ANSI C99 and the OpenCL C languages of correlation coefficient (Pearson, Spearman Rank) algorithms for the calculation of edge matrices and a modified Fruchterman-Rheingold (FR) layout algorithm (Enright and Ouzounis, 2001; Fruchterman and Rheingold, 1991) used for network layout in 2 and 3-dimensional space. We have implemented the parallelized versions of these algorithms within the BL*E*^{3D} codebase, providing a GUI to control their operational characteristics and then tested their performance on a range of desktop configurations and graphics cards.

2 METHODS

2.1 General Parallelization Rules

In order to efficiently use all available resources of multicore CPUs and GPUs and their available PUs and SIMD PUs respectively, parallel code must strive to meet the following requirements (Eichner *et al*., 2009):

(1) Time spent on non-parallel, sequential code blocks should be minimized. Algorithms should strive to set up everything in advance in a sequential fashion with fast initialization routines and then let parallel code execute the main processing tasks as fast as possible.

(2) An optimal load balance should be employed between all PUs to fully utilize them and leave as little idle execution PU time as possible.

(3) Parallelism overheads should be minimal. Synchronization operations should be kept to a minimum or skipped altogether if the nature of the underlying algorithm permits it. If this is unavoidable, the use of efficient native-to-hardware synchronization constructs along with profiling tests is highly recommended.

(4) Relevant for GPUs only - memory transactions between the on-board GPU Video RAM and the GPU's SIMD PUs should be minimized as much as possible. The internal Level 2 GPU local cache should be used wherever possible, as the local cache is usually an order of magnitude faster (Munshi and Gaster, 2011) than the main on-board RAM. Recent GPU hardware has made a lot of progress on the I/O issues of on-board-RAM-to-GPU transfers, but it is still slower than similar CPU transactions, mainly due to the lack of a large Level 2/3 cache.

The BL*E*^{3D} parallel code employs all the above rules along with the *data parallelism* form of parallelization, where problems are expressed in terms of collections of data elements that can be updated concurrently. The parallelism is expressed by concurrently applying the same stream of instructions (a task) to each data element: the parallelism is in the data. This approach is highly effective in matrix related calculations, such as the ones already used within the BL*E*^{3D} code framework. Data parallelism is also the standard parallelization technique supported by OpenCL.

Over the last few years all major CPU architectures introduced an efficient form of lock-free, thread-safe programming on single integer-based primitive variables and arrays of these. These variables, known as *atomics*, are highly efficient as they are guaranteed to be lock-free and thread-safe at the CPU hardware level, instead of using computationally expensive *mutexes* in program code. Recent GPU hardware also employs similar means for lock-free thread-safe variables.

2.2 Platform and hardware independent approach to parallelization using the Java 1.6, ANSI C99 and OpenCL C languages

The Java language, from the times of its original specification, has a comprehensive set of libraries that support general parallel programming principles down to the thread level. The latest revisions of the language have added support for efficient atomic operations through the atomic package library (*java.util.concurrent.atomic*) used for fast synchronization. This comes with the usual advantages of the Java language being supported on all major operating systems (OSs), along with being able to run them without any code re-compilation, thus providing an easy means of code deployment. Additionally, the latest versions of the JVM for Java 1.6 are highly optimized and currently produce very competitive performance compared to ANSI C99. The Java language also provides a native interface, the Java Native Interface (JNI), which can be used to further optimize algorithm performance by running efficient native code written in C.

Our approach for the parallelization developments described here was heavily influenced by the fact that the application framework code base for BL*E*^{3D} was already written in a combination of Java 1.6 and ANSI C99 code (Enright and Ouzounis, 2001). In 2010, support for the OpenCL library was also introduced for Java 1.6, again written in ANSI C99 and accessed through JNI. We avoided the use of CUDA as it provides support only for a specific hardware vendor, Nvidia, while OpenCL is an industry standard framework for programming GPUs from any hardware vendor. The BL*E*^{3D} framework therefore operates in the manner below for CPU and GPU parallelization:

(1) The Java 1.6 side handles all the general thread programming and synchronization on the CPU side, practically acting as thread control code and execution, achieving N-Core parallelization through Java 1.6 (N-CP-Java).

(2) For certain OSs, ANSI C99 library code through a JNI call is being used for the core processing of the algorithm's code block inside the thread that has been issued and controlled through the aforementioned Java code (N-Core parallelization through ANSI C99, N-CP-C). If no ANSI C99 library code compilation is available for that particular OS, the program resorts to equivalent Java 1.6 code for that code block (N-CP-Java) to maintain compatibility.

(3) The host code for OpenCL is handled through the JOCL libraries with Java 1.6 while OpenCL kernel code is written in highly optimized OpenCL C language.

(4) An advanced OOP framework was implemented within BL*E*^{3D} around JOCL. If the OpenCL GPU computing fails for any reason, the parallel computation safely resorts to CPU parallelization (either N-CP-Java or N-CP-C), if available, or even to single-core processing for older hardware.

With the above combination of languages, BL*E*^{3D} obtains optimum performance whilst maintaining compatibility with all major hardware vendors and OSs.

2.3 Computational complexity of pairwise association algorithms: Correlation coefficient and Fruchterman-Rheingold layout algorithms

Many studies in biology utilize a probability measure derived from statistical measurements of similarity to calculate the pairwise association between biological entities (Ganter and Giroux, 2008; Baitaluk *et al.*, 2006). These pairwise associations can then be visualized by as a network graph consisting of nodes (vertices) representing entities and the probability of their association being represented as a line (edge) connecting them. In order to facilitate such an approach BL*E*^{3D} employs two such algorithms for the processing and visualization of very large correlation networks (Freeman *et al.*, 2007; Theocharidis *et al.*, 2009), in particular for representing co-expression networks derived from microarray data. The Pearson and Spearman Rank correlation coefficient calculations and a modified version of the FR layout algorithm (Freeman *et al.*, 2007; Su *et al.*, 2004) are used within BL*E*^{3D}. They are computationally expensive and when processing large datasets or graphs, and have proved to be the main bottleneck in terms of speed of operation for the application. They were therefore excellent candidates for parallelization. Detailed descriptions of the algorithms themselves are provided elsewhere (Enright and Ouzounis, 2001; Freeman *et al.*, 2007; Fruchterman and Rheingold, 1991) but so as to describe the computational overhead of these algorithms, a brief description of their computational structure is included here.

The underlying principle of pairwise association calculations is the requirement to calculate a matrix of the 'distance' between every entity. This pairwise measure of every entity to every other entity can be represented with an NxN square matrix, where N is the number of entities. Calculations need to be performed for only one instance of a pair of entities as order does not matter and there is no need of calculating self-associations. In this respect, pairs of entities are treated like sets, not sequences. Thus, the only calculations needing to be performed in the NxN square matrix are aggregated in the upper right (or equivalently the lower left) diagonal of the matrix, with the main diagonal excluded.

The mathematical formula for the pairwise association calculation is given below:

$$\sum_{i=0}^{N} \sum_{j=i+1}^{N} f(x) \tag{1}$$

The number of calculations performed can be given by the mathematical formula below. It is apparent that the computational complexity in this case is O(n^2):

$$NumberOfCalculations = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \tag{2}$$

Especially for the case of the FR layout algorithm, its calculations have to be iterated a set number of times for it to converge (collapse) to an optimal graph structure (Fruchterman and Rheingold, 1991). With default settings, this iteration loop is standardly performed 100 times within the BL*E*^{3D} application framework. In this case the equation is formed as below:

$$\sum_{iter=0}^{ITER} \sum_{i=0}^{N} \sum_{j=i+1}^{N} f(x) \tag{3}$$

$$NumberOfCalculations = ITER \frac{n(n-1)}{2} \tag{4}$$

In this case and for a big iteration value, the computational complexity can easily scale to the O(n^3) domain.

Having recently improved the ability of BL*E*^{3D} to render very large network graphs routinely comprising of around 20,000-30,000 nodes and millions of edges, the amount of calculations needed to prepare the graph for visualization could take many hours to complete using a single CPU, something undesirable for an interactive and UI-driven application. The only alternative was to parallelize the algorithms.

3 RESULTS

Our tests with the two pairwise association algorithms utilizing parallelization were conducted using the GNF1M mouse tissue atlas dataset (Su *et al.*, 2004), which can be found at the BioGPS website (biogps.gnf.org) or the BL*E*^{3D} website (www.biolayout.org/download/example-datasets). The input data file consisted of 36,182 rows (transcripts) and 122 columns (samples), a total file size of 29Mb. As such, the calculations required for the pairwise association algorithms were in total 654,532,380 (n = 36,182). The non-parallel variants of the two algorithms were used to verify that the outputs in all cases were identical. In order to further test I/O overheads between the various CPU and GPU implementations, the test for the correlation calculation was performed multiple times with the original GNF1M mouse tissue atlas dataset and a customized version that involved duplicating the data to a give a total of 244 data columns. The FR layout algorithm was tested with using a default correlation value of r = 0.85.

In order to test the scalability of our parallel implementations, a number of machines with different hardware configurations were employed for testing:

1 server machine equipped with:

- A server machine equipped with 2x Intel Xeon Quad Core X5482 CPUs at 3.2Ghz with 2x 6Mb of Level 2 cache and 16Gb of RAM running Ubuntu Linux 7.04 64bit (an 8 PUs machine with Hyperthreading turned off), used for benchmarking the N-CP parallel implementations for up to 8 PUs.

4 high-end desktop machines equipped with:

- Intel Core i7 970 CPU at 3.2Ghz with 8Mb of Level 2 / Level 3 cache and 12Gb of RAM running Windows 7 64bit (an 6 PUs machine with Hyperthreading turned off) and the latest NVIDIA Fermi Geforce GTX 580 GPU with 1.5GB of Video RAM, 64k-48k unified SIMD Level 2.
- Intel Core2 Quad 2.83Ghz Q9550 at 2.83Ghz with 12Mb of Level 2 cache and 4Gb of RAM running Windows XP SP3 32bit (a 4 PUs machine) and the NVIDIA Geforce GTX 285 GPU with 1GB of Video RAM, 16k Level 2 per SIMD unit.
- Intel Core2 Quad 2.4Ghz Q6600 at 2.4Ghz with 8Mb of Level 2 cache and 4Gb of RAM running Windows XP SP3 32bit (a 4 PUs machine) and the latest AMD ATI Cayman 6970 GPU with 2GB of Video RAM, 32k Level 2 per SIMD unit.
- Intel Core2 Quad 2.4Ghz Q6600 at 2.4Ghz with 8Mb of Level 2 cache and 4Gb of RAM running Windows XP SP3 32bit (a 4 PUs machine) and the AMD ATI Cypress 5870 GPU with 2GB of Video RAM, 32k Level 2 per SIMD unit.

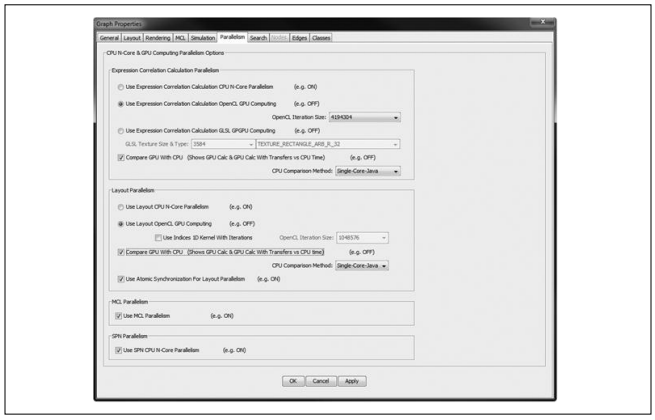

Fig. 1. The Parallelism tab button in BLE^{3D}'s Graph Properties.

The latest version of the BLE^{3D} application was programmed to add support for all the parallel implementations through an easy-to-use user interface (UI) accessed through the Parallelism tab in graph properties toolbar (*Figure 1*). CPU and GPU comparisons were directly made through the software with the appropriate option turned on (''Compare CPU with GPU') and detailed text-based logs of CPU and GPU run times were captured to files. The two correlation calculation algorithm tests were run for 157 iterations (default iteration size for the GNF1M dataset with an OpenCL transfer iteration size of 4,194,304 results) and the FR layout algorithm run for 100 layout iterations (default value in BLE^{3D}).

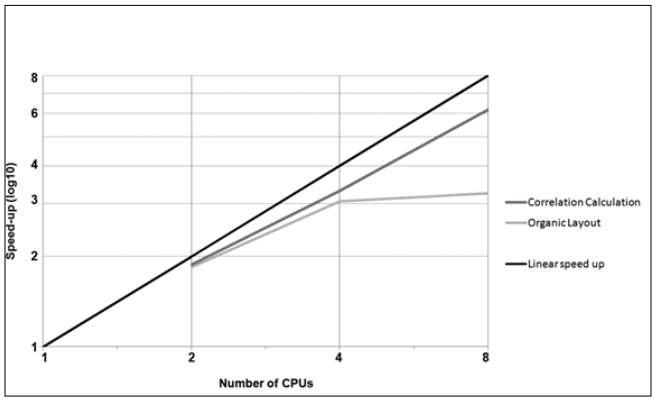

Fig. 2. Scalability results for the N-CP CPU parallel implementations for the two algorithms.

The OpenCL kernels were highly optimized using local GPU cache (for the correlation calculation) and sophisticated vectorization techniques to minimize memory I/O and SIMD processing as much as possible. The FR OpenCL kernel was also implemented using Atomics, now natively available on latest GPU hardware. Times were captured using the OpenCL profiling extension to nanosecond accuracy. These were compared to both Single-Core and N-CP-Java and N-CP-C CPU implementations (timed similarly) so as to provide real-world performance improvements when using the BLE^{3D} application.

The CPU parallelization results showed that a near-linear speed-up to the number of available processing units was reached for correlation calculations. The FR layout algorithm showed a near-linear speed-up up to 4 PUs when synchronization was used, but did not scale much further on with tests with 8 PUs (*Figure 2*). The Atomic synchronization overhead of the algorithm became apparent with more than 4 PUs. On the other hand, experiments using synchronization with old fashioned mutexes instead of Atomics, showed a prohibitive synchronization overhead running even only with 2 PUs. Enabling Intel's Hyperthreading technique also gave another 5%-10% speed-up by using 2 threads per CPU PU (16 and 12 threads for 8 and 6 PUs respectively).

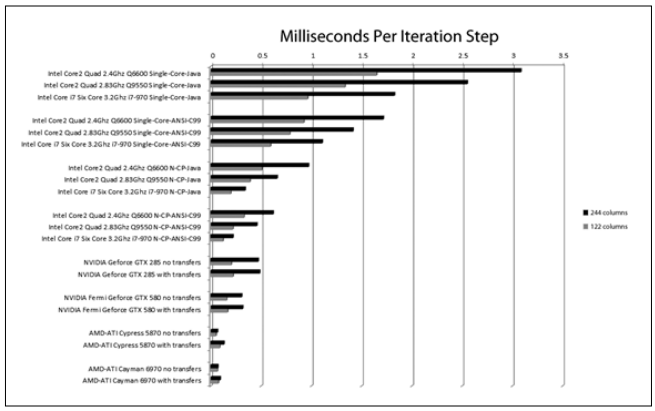

Fig. 3. Combined algorithm execution times between the Single-Core, N-CP and OpenCL kernel runs for the Correlation Calculation algorithm.

The GPU parallelization tests showed that the main bottleneck of the correlation calculation OpenCL kernel was the memory I/O for calculating the profiles' sum of the pair in question. Optimization methods differed: the I/O transfers vectorization code proved not to help with Nvidia based hardware whereas it significantly improved matters with the latest AMD/ATI based hardware achieving very competitive speed-ups compared to CPUs. On-chip Level 2 caching improved performance on both vendors' GPU hardware (*Figure 3*).

On the other hand, the FR OpenCL kernel scaled well on both hardware vendors' GPUs. As soon as Atomics were introduced there was a big performance drop compared to the latest generation of Intel CPUs. Still, the Nvidia Fermi-based hardware showed a very big speed improvement compared to any other GPUs with Atomics turned on (*Figure 4*).

4 DISCUSSION

The CPU parallelization of the two algorithms used for this work produced a near-linear scalability with the number of CPUs if synchronization was not employed or needed. In the case of the modified FR layout algorithm (Enright and Ouzounis, 2001; Fruchterman and Rheingold, 1991), when synchronization was employed through the efficient atomic Java API, a near-linear scalability was apparent for up to 4 PUs but then the overhead of synchronization prevented further linear speed up of the algorithm as demonstrated in tests utilizing 8 PUs (*Figure 2*). Beyond 8 PUs it is not easy to predict how well the N-CP implementation will scale for CPUs comprised of 16 or more PUs. However, N-CP has been designed with these architectures in mind and we are confident that a near-

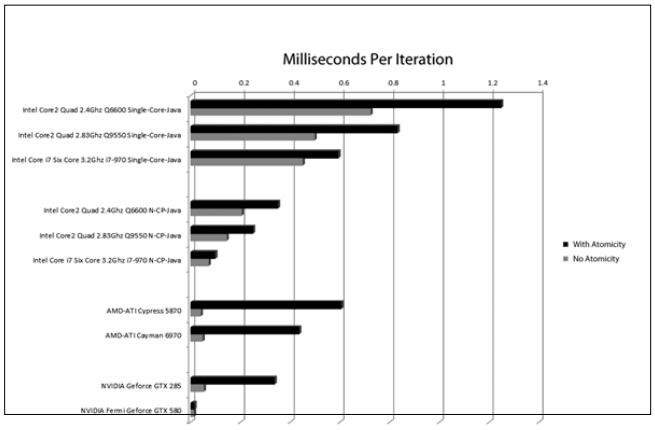

Fig. 4. Combined algorithm execution times between the Single-Core, N-CP and OpenCL kernel runs for the FR Layout algorithm.

linear speed up in computational times will be achievable with unsynchronized algorithms with newer CPUs.

With OpenCL kernels, we witnessed an order of magnitude speed-up against a variety of CPUs with a best-case scenario of 60x times speed-up (Intel Core2 Quad 2.4Ghz Q6600 CPU using 1 PU vs. NVIDIA Fermi Geforce GTX 580 GPU, *Figure 4*). Even when the CPUs were fully harnessed through N-CP, GPU Computing through OpenCL kernels still provided very competitive performance gains. The results when compared between different generations of GPU hardware are as a result of GPU hardware vendors introducing better GPU architectures to cope with Atomicity and memory I/O transfers with every new generation of their GPU/graphics cards. GPU computing is a very cost-effective way improving application scalability, especially when compared with the cost (at the time of writing) of the latest generation Intel i7 970 CPU (£750) to a latest Nvidia Fermi-based Geforce GTX 580 (£380) and the latest AMD ATI Radeon 6970 (£280). This makes GPU computing a very viable option for low-cost application scalability on desktop machines.

The main downside is that it is still non-trivial to write efficient GPU code as many manual optimizations need to be written. We also found variable performance differences with the different optimization techniques used between the two main GPU vendors, Nvidia and AMD/ATI. We believe this is because GPU computing is still in its evolutionary phase. With newer GPU hardware and better OpenCL C compilers, some of these manual techniques needed now to achieve optimum performance may no longer be required.

A sophisticated OpenGL GLSL Shader using Shader Model 4.0 was also attempted for the correlation calculation but did not provide as competitive results compared to the equivalent OpenCL C kernels. This proves that the GPU vendors have moved a lot in the direction of providing an efficient GPU computing language with the OpenCL API and OpenCL C compared with the early days of experimental GPGPU computing through the OpenGL API and GLSL Shaders.

As an indication of the practical speed-ups achieved for the BL*E*^{3D} Version 2.0 framework, a usual run of the correlation calculation for the GNF1M dataset and the associated FR layout algorithms for a chosen correlation value of r = 0.70 in the past took more than 25 minutes. With the combination of the N-CP implementations in Java 1.6, ANSI C99 native libraries and OpenCL C through OpenCL kernels these times are now down to less than a minute. This makes the BL*E*^{3D} Version 2.0 application a much more interactive feeling to the end user, along with making them able to better experiment with larger datasets without the time overhead that this is usually associated with doing so. Additionally, the scalability of this approach combined with the future CPUs and GPUs having even more PUs and SIMD PUs integrated on-chip, makes us confident that, as biology needs continue to grow in computational terms, the BL*E*^{3D} framework will continue to be able to support future datasets that go way beyond the 60,000 rows and need to layout many more than 45,000 nodes and 4,500,000 edges to create a network graph as currently achieved (www.biolayout.org/about/gallery).

5 CONCLUSION

In this paper we present parallelization techniques which are incorporated and extensively used within the BioLayout *Express*^{3D} Version 2.0 application framework. The code has been designed to provide a generic framework and therefore to be reusable in the future for other computationally intensive routines. We also present as case studies two pairwise association algorithms. The Java 1.6 and ANSI C99 programming languages were used for the multi-core CPU N-CP implementation, where Java 1.6 was used for parallel thread programming and synchronization and ANSI C99 for efficiently executing the core computations on each PU. Optimized OpenCL C kernels where used for the GPU Computing side. To the best of our knowledge, this is the first manuscript to describe using a combination of these three languages used for parallelization on multiple OSs and cross-vendor hardware in one unified code framework.

The BL*E*^{3D} code base, which includes all the parallel implementations, is freely available under Version 3 of the GNU General Public License (GNU GPLv3) or any later version. (www.biolayout.org/download).

6 AUTHOR'S CONTRIBUTION

AT conceived, designed and implemented the parallel algorithms in the BL*E*^{3D} framework and was the primary author of the paper. HG and MK helped optimizing the parallel code in Java 1.6, ANSI C99 and OpenCL C kernels. TCF acts as primary beta test user of BL*E*^{3D}, leads the application development and co-authored this paper. All authors drafted, read and approved the final manuscript.

7 FUNDING

This work was supported by the Biotechnology and Biological Sciences Research Council (BBSRC) [grant number BB/I001107/1].

8 ACKNOWLEDGEMENTS

The authors would like to thank all people in the Freeman group in Roslin Institute for their valuable inputs, fruitful discussions, comments and beta testing sessions of the BL*E*^{3D} Version 2.0 application framework. Many thanks also to the Edinburgh Compute and Data Facility (ECDF, www.ecdf.ed.ac.uk) and in particular Yuan Wan and John Blair-Fish, for their valuable help and support during testing with their Linux 64bit system. Many thanks also to Dr. Nigel Binns, Division of Pathway Medicine, University of Edinburgh, for his insightful comments on this work.

9 REFERENCES

Baitaluk M.S., Ray M., Gupta A. (2006). Biological Networks: visualization and analysis tool for systems biology. *Nucleic Acids Res*;34:W466-71.

Blom J., Jakobi T., Doppmeier D., Jaenicke S., Kalinowski J., Stoye J. and Goesmann A. (2011). Exact and complete short read alignment to microbial genomes using GPU programming. *Bioinformatics*. Mar 30.

Burkitt M., Walker D., Romano D.M. and Fazeli A. (2011). Constructing Complex 3D Biological Environments from Medical Imaging using High Performance Computing. *IEEE/ACM Trans Comput Biol Bioinform*. Mar 31.

Chavent M., Lévy B., Krone M., Bidmon K., Nominé J.P., Ertl T. and Baaden M. (2011). GPU-powered tools boost molecular visualization. Brief Bioinform. 2011 Feb 9.e Systems Biology Toolbox 2 for MATLAB. *Bioinformatics*. Feb 25.

Davis N.A., Pandey A. and McKinney B.A. (2011). Real-world comparison of CPU and GPU implementations of SNPrank: a network analysis tool for GWAS. *Bioinformatics*. Jan 15;27(2):284-5.

Dematté L. and Prandi D. (2010). GPU computing for Systems Biology. *Briefings in Bioinformatics*, Volume11, Issue3, Pp. 323-333.

Eichner H., Klug T. and Borst A. (2009). Neural simulations on multi-core architectures. *Front Neuroinformatics*;3:21.

Enright A.J. and Ouzounis CA. (2001). BioLayout - an automatic graph layout algorithm for similarity visualization. *Bioinformatics*;17:853-4.

Freeman T.C., Goldovsky L., Brosch M., Dongen S., Enright A.J, et al. (2007). Construction, visualisation, and clustering of transcription networks from microarray expression data. *PLoS Comput Biol*;3:2032-42.

Fruchterman T.M. and Rheingold E.M. (1991). Graph drawing by force directed placement. *Softw Exp Pract*;21:1129-64.

Ganter B. and Giroux C.N. (2008). Emerging applications of network and pathway analysis in drug discovery and development. *Curr Opin Drug Discov Devel*;11:86-94.

Geer D. (2005) Chip Makers Turn to Multicore Processors. *IEEE Computer Society*.

Hemani G., Theocharidis A., Wei W. and Haley C. (2011). EpiGPU: Exhaustive pairwise epistasis scans parallelised on consumer level graphics cards. *Bioinformatics*.

Klingbeil G., Erban R., Giles M. and Maini P.K. (2011). STOCHSIMGPU: Parallel stochastic simulation for the Systems Biology Toolbox 2 for MATLAB. *Bioinformatics*. Feb 25.

Liu Y., Schmidt B. and Maskell D.L. (2011). DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI. *BMC Bioinformatics*. Mar 29;12(1):85.

Munshi A., Gaster B., Mattson T. and Ginsburg D. (2011). OpenCL Programming Guide, Rough Cuts. *Addison-Wesley Professional*.

Razmyslovich D., Marcus G., Gipp M., Zapatka M. and Szillus A. (2010). Implementation of Smith-Waterman Algorithm in OpenCL for GPUs. Enschede, Netherlands, September 30-October 01, *ISBN: 978-0-7695-4265-2*.

Shalf J. (2007). The New Landscape of Parallel Computer Architecture. *Journal of Physics: Conference Series* 78 012066.

Su A.I., Wiltshire T., Batalov S., et al. (2004). A gene atlas of the mouse and human protein-encoding transcriptomes. *Proc Natl Acad Sci USA*;101:6062-7.

Theocharidis A, van Dongen S., Enright A.J. and Freeman T.C. (2009). Network Visualisation and Analysis of Gene Expression Data using BioLayout Express3D. *Nature Protocols*;4.

Varré J.S., Schmidt B., Janot S. and Giraud M. (2011). Manycore high-performance computing in bioinformatics. *Advances in Genomic Sequence Analysis and Pattern Discovery*.

Yung L.S., Yang C., Wan X. and Yu W. (2011). GBOOST : A GPU-based tool for detecting gene-gene interactions in genome-wide case control studies. *Bioinformatics*. Mar 3.

Zhou J., Liu X., Stones D.S., Xie Q. and Wang G. (2011). MrBayes on a Graphics Processing Unit. *Bioinformatics. Mar 16*.

Zhou Y., Liepe J., Sheng X., Stumpf M.P. and Barnes C. (2011). GPU accelerated biochemical network simulation. *Bioinformatics*. Mar 15;27(6):874-6.