Hardware acceleration of reaction-diffusion systems: a guide to optimisation of pattern formation algorithms using OpenACC

Ruth E Falconer
Alasdair N Houston
Xavier Portell
Wilfred Otten

The published paper is available from
doi: 10.23919/SpringSim.2019.8732883

HARDWARE ACCELERATION OF REACTION-DIFFUSION SYSTEMS: A GUIDE TO OPTIMISATION OF PATTERN FORMATION ALGORITHMS USING OPENACC

Ruth E Falconer
Alasdair N Houston

School of Design & Informatics
Abertay University
Bell Street, Dundee, UK

Xavier Portell
Wilfred Otten

Cranfield Soil and Agrifood Institute
Cranfield University
Bedfordshire, UK

ABSTRACT

Reaction Diffusion Systems (RDS) have widespread applications in computational ecology, biology, computer graphics and the visual arts. For the former applications a major barrier to the development of effective simulation models is their computational complexity - it takes a great deal of processing power to simulate enough replicates such that reliable conclusions can be drawn. Optimizing the computation is thus highly desirable in order to obtain more results with less resources. Existing optimizations of RDS tend to be low-level and GPGPU based. Here we apply the higher-level OpenACC framework to two case studies: a simple RDS to learn the 'workings' of OpenACC and a more realistic and complex example. Our results show that simple parallelization directives and minimal data transfer can produce a useful performance improvement. The relative simplicity of porting OpenACC code between heterogeneous hardware is a key benefit to the scientific computing community in terms of speed-up and portability.

Keywords: OpenACC, GPGPU, Hardware acceleration. Gray-Scott, Reaction Diffusion Models.

1 INTRODUCTION

Reaction Diffusion Systems (RDS) are widely used in the modeling of complex systems and have been used extensively to investigate pattern formation in both chemical and biological systems. A considerable amount of literature has been published on pattern formation driven by the foundational work of Darcy Thompson, Alan Turing (Turing, 1952) and Hans Meinhardt (Meinhardt, 2003), recognizing the remarkably interesting patterns that RDS produce. These patterns can vary over space and time in 1,2 and 3-dimensions. The first RDS's where intended to emulate chemical reactions such as the Belousov–Zhabotinsky reaction and animal/shell patterning e.g. seashells, tigers and zebra patterns (Turing (1952; Meinhardt, 2003).

RDS are a class of Partial Differential Equations (PDE's) that are required to be solved numerically. With increasing computational power they are a growing in popularity, and have been a source of inspiration for artists and scientists alike. More recent applications of RDS include texture synthesis, pattern formation, procedural content generation and theoretical ecology. Falconer and coworkers (Falconer et al. 2005; Falconer and Houston 2015) used RDS to generate the range of microbial phenotypes as observed in the laboratory; Galanter, (2014) situates RDS in the myriad of techniques used in generative art. Malheiros and Walter (2017) developed a minimalist RDS for pattern formation used in graphics and procedural content generation e.g. to generate maps in 2 or 3 dimensions which forms the space that a player occupies and interacts with. Finally, Turk, (1991) and Witkin and Kass, (1991) exploited RDS for real-time animated texture synthesis.

RDS are mathematical models which encapsulate at least two chemicals that diffuse and undergo transformations from one chemical to another - the reaction term. One of the simplest RDS is the Gray-Scott model which is typically implemented using a structured spatial grid where computation proceeds as a series of cell updates. Updating the grid involves updating each cell based on the values of neighboring cells using a stencil-based pattern of computation. As such they do not belong to the class of algorithms known to be embarrassing parallel as dependencies exists amongst the cells to be processed. Nevertheless there have been several investigations into optimizing RDS particularly using GPGPU approaches.

Harris (2005) investigated a Cg GPGPU implementation of the Gray-Scott model which mapped computational concepts to GPUs. The approach uses a single texture and a kernel that runs on the GPU, that captures the underlying Gray-Scott computation. Falconer and Houston (2015) ported a microbial RDS to the GPU using the Direct Compute framework reporting noticeable speedups. Schram (2013) developed an GPGPU application to perform Monte Carlo simulations on a subset of reaction–diffusion models. Speeding up the pipeline is important as Monte Carlo methods require many runs for statistical accuracy/correctness. The authors report that the GPU algorithm is roughly 55 times faster than an optimized version for the CPU. A major barrier to the development of effective simulation models is their computational complexity - it takes a great deal of processing power to simulate enough replicates at an appropriate spatial scale such that reliable conclusions can be drawn. Optimizing the computation is thus highly desirable in order to obtain more results with less resources (time, energy and hardware).

Low-level approaches appear to dominate the hardware acceleration of RDS, where it appears, that portability and maintainability are sacrificed for performance. Recently frameworks have emerged that generate portable and maintainable codes for heterogeneous architectures which are now prevalent. These frameworks include OpenACC which is a high level, descriptive, directive approach to parallel programming. The programmer specifies the block of code, often a nested loop, to be run in parallel on an accelerator (GPU, multicore CPU) and leaves the exact mapping to the complier. The compiler output shows the mapping between OpenACC and CUDA constructs. OpenACC permits single source code for different architectures (multi and many-cores) promoting portability. OpenACC has two main constructs to create parallel regions to be run on the accelerator, the `parallel` and the `kernels` constructs. Although both allows offloading the computation of nested loops they are not equivalent. The former requires analysis by programmer to ensure safe parallelism and some clauses require `parallel`, e.g. reduction. The latter gives control to the compiler to undertake the parallel analysis and parallelizes what it believes safe. There is a paucity of OpenACC implementations of RDS that have been described and formulated in the literature, despite the widespread applications and uses of RDS.

2 METHODS

This work describes the OpenACC implementation of two case studies: the first the simplest RDS formulation - the Gray-Scott Reaction Diffusion (GSRD) written from scratch and optimized using OpenACC, including performance measurements. Other than its aesthetic appeal, GSRD is of interest here precisely because of its simplicity compared to other RD models that attempt to simulate phenomena observable in the real-world such as microbial growth. A more detailed description and discussion of this model may be found at: https://groups.csail.mit.edu/mac/projects/amorphous/GrayScott. The motivation was to learn the workings of the OpenACC directives before applying to a more complex and realistic example. Such simulation models tend to be more complex in every respect: irregular spatial structure in 3D, many mobile reacting components, incorporation of phases with variable pressure and temperature, flow in addition to diffusive transport, and far more complex systems of coupled partial differential equations. As an example of a higher complexity model, consider microbial activity within porous media having an effect on wet-dry cycle behavior within pore space at small length scales (tens of millimeters or less). Such behavior potentially leads to large scale phenomena affecting e.g. the drainage versus retention of surface water. An improved understanding of such phenomena may be able to better inform policy

making for more effective management within the natural and built environment. Consequently, an initial investigation into OpenACC using a simplified model is attractive: less effective strategies can be rapidly eliminated, allowing effort to be concentrated on the most promising optimizations. Finally, we present the lessons learned from translating and generalizing the learnings to an existing legacy codebase of a RDS describing microbial patterning in structured environments with complex boundaries (case study 2).

From the literature there are a several things to consider in order to produce performant OpenACC code: These include ensuring enough compute intensity and limiting data transfers from host to device. Both case studies are compute intensive and possess nested loops which can map to `parallel` or `kernel` constructs. Minimization of data transfers has also been duly considered in the two case studies below.

2.1 Case Study 1: Pattern Formation using the Gray-Scott model

The GSRD is defined as:

$$\frac{\partial U}{\partial t} = D\_u\nabla^2 U - UV^2 + F(1-U),$$

$$\frac{\partial V}{\partial t} = D\_v\nabla^2 V - UV^2 - (F+k)V$$

Where U and V are the interacting 'chemical' species. The first and last two terms of the RDS represents the diffusion and reaction properties respectively. Diffusion of the chemical species U and V is implemented using a finite difference form of the Laplacian operator applied to a scalar field on a two-dimensional Cartesian grid (with indices *x*, y, and cell size h).The GS parameters used in the OpenACC implementation are: Du = 0.5, Dv = 0.1, F= 0.023 and k =0.049.

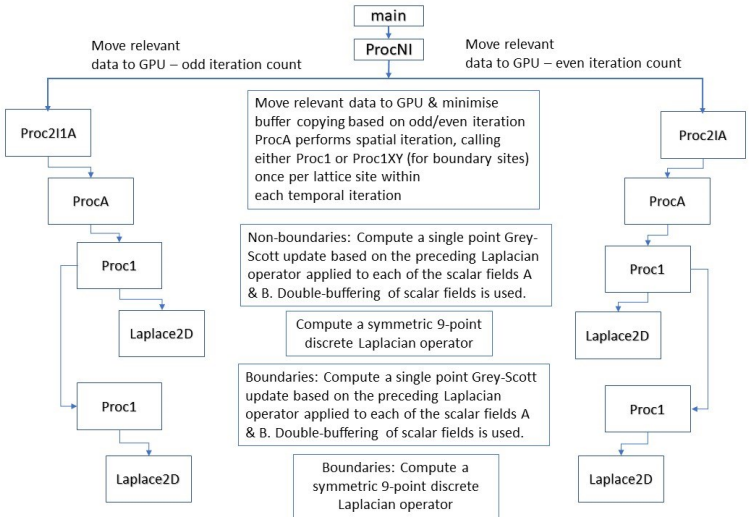

Figure 1: High-level view of the implementation of the GS OpenACC program flow – see https://github.com/DrAl-HFS/GSRD.git for source code.

2.1.1 GSRD OpenACC Implementation

Some familiarity with acceleration directives was gained from the many available guides and examples but using OpenACC effectively required the development and test of progressively less simple code in order to better understand how directives should be used for correctness and efficiency. A case in point was the handling of boundary conditions in discretized diffusion: a 1D test case was very helpful in clarifying

effective strategies. In cases where compiler output is examined to identify or clarify problems, simplification of source code is highly desireable.

A high-level overview of the OpenACC GSRD program flow is illustrated in Figure 1. All code is downloadable via Github. The directives used to transfer data and offload portions of the computation to the accelerators are described in Tables 1 and 2, depending on whether the directive is related to data movement or computation.

Table 1: Data Movement and Processing Directives.

Directive	Function
`#pragma acc data present( pR[:pO->n], pS[:pO->n], pO[:1], pP[:1] )`	Ensure all necessary data (Result and Source scalar field buffers, plus Organizational metadata and numerical model Parameters) is present on compute device.

`#pragma acc data present_or_create( pR[:pO->n] )`
`copyin( pS[:pO->n] ) copyout( pR[:pO->n] )`
`present_or_copyin( pO[:1], pP[:1] )`
	Double buffered data movement used for even numbered iteration count: following an even number of iterations, the updated state will lie within the SR (Source-Result) scalar field buffer and this is therefore copied back to the host (as well as being copied to the device before the first iteration). The TR (Temporary-Result) buffer has received the temporary result of the first and subsequent odd-numbered iterations and therefore does not require any copy from or to the host nor does it require any initialization of its contents prior to the first iteration. The Organizational metadata and numerical Parameters are copied in for read-only access.

`#pragma acc data present_or_create( pTR[:pO->n] )`
`copy( pSR[:pO->n] )`
`present_or_copyin( pO[:1], pP[:1] )`
	Double buffered data movement used for even numbered iteration count: following an odd number of iterations, the updated state will lie within the R (Result) scalar field buffer which must be copied back to the host. The initial state given within the S (Source) buffer is copied from the host prior to the first iteration. The O & P metadata and parameters are copied in for read-only access.

Table 2: Computation Directives.

Directive	Function
`#pragma acc parallel loop`	Parallelizes nested loop for updating the two chemical species values applied to each cell of the computation grid: outer for loop for processing the compute domain excluding boundaries. Iterations of subsequent nested loops are

	independent and may be executed in any order (ideally concurrently).
`#pragma acc loop vector`	Parallelizes nested loop for updating the two chemical species values applied to each cell of the computation grid: inner for loop for processing the compute domain excluding boundaries. Iterations of subsequent for loops are suitable for (SIMD) vector execution
`#pragma acc parallel`	Parallelizes the computation of the chemical species values pertaining to the boundaries e.g. the four lines. Iterations of subsequent nested loops are independent and may be executed in any order (ideally concurrently)
`#pragma acc loop vector * 4`	Parallelizes the computation of the boundaries e.g. the four lines. Iterations of subsequent for loops are suitable for (SIMD) vector execution

For case study 1 the speed-up and scalability of the GSRD system was investigated using the parallelization directives described above applied to computation on grids of size 512*512, 1024*1024 and 2048*2048. The performance gains achieved by running on multi- and many-core accelerators were investigated. Additionally for the CPU builds the number of threads the program will use to run the parallel compute regions is controlled with the environment variable ACC_NUM_CORES.

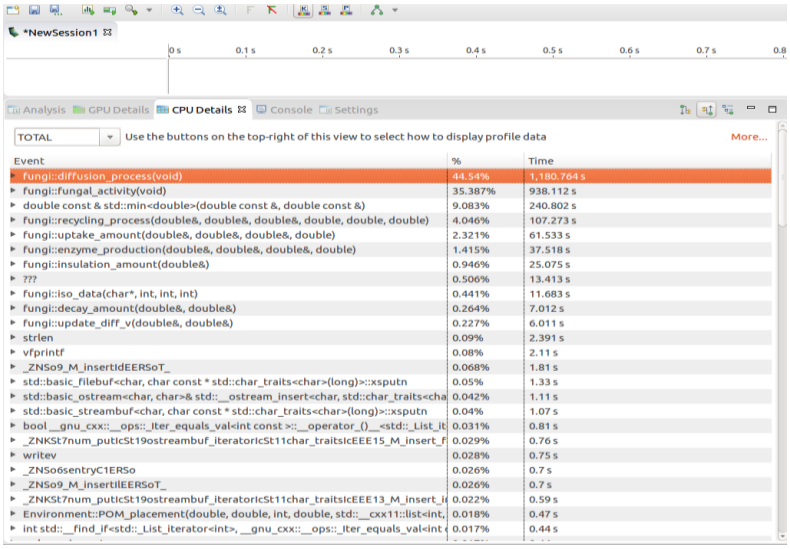

Figure 2: Bottlenecks observed in both the computation of the diffusion (diffusion process) and the reaction terms (fungal activity) prior to parallelization.

2.2 Pattern Formation in Microbial Systems

The system of equations describing the Microbial Reaction Diffusion (MRD) system is defined in Falconer and Houston (2015). There are more state variables (chemical species) in the MRD and the reaction terms

governing the transformations from one state variable to another are more complicated. This model has been used to explore the role of soil fungi in carbon degradation and the Carbon cycle.

The model has been implemented in C++ and exists as two main classes: one relating to the properties and functionality of the microbial organism (fungi.cpp) and the environment (Environment.cpp) in which it interacts. Profiling of the non-accelerated version of the code highlights bottlenecks in both the computation of the diffusion and the reaction terms of the model (Figure 2). OpenACC efforts are therefore directed to these functions. The directives used to offload portions of the computation to the accelerators are described in Tables 3 and 4 depending on whether the directive is related to data movement or computation.

2.2.1 MRD system OpenACC Implementation

For case study 2 the speed up of the algorithm was investigated by running the MRD algorithm, using the parallelization directives described in Tables 3 and 4 and running on the host, a multi- and many-core (GPU) accelerator. Additionally, the effect of using the `parallel` versus `kernel` directives when parallelizing the two most costly functions was investigated.

Table 3: Data Movement and Management Directives used in case study 2.

Directive	Function
`# pragma acc enter data copyin (this)`	In Constructors: Copies the host pointer to the device.
`# pragma acc update device (this)`	In Constructors: Copies all members to the device.
`# pragma acc exit data delete (this)`	In Destructor: Removes the data from device when the destructor is called.

`present(this)`
`present(environment)`
`present(environment.DOC[0:x][0:y][0:z])`
	Avoids unnecessary data movement between host and device. Notice that to let the compiler know the members of the environment, both the pointer (environment) and the size of the data members have to be specified (deep copy).

`pragma acc parallel`	The data directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region

Table 4: Data Processing Directives used in case study 2.

Directive	Function
`# pragma acc loop reduction(+totdtUptk)`	The reduction clause informs the compiler that all operations are summed up in some local variables, so the compiler can deal with it.
`# pragma acc loop`	The parallelisation of the nested i, j and k loops is achieved using a parallel region including the present data clause, which informs the compiler that the data required are already in the device

	(avoiding unnecessary data movement between host and device).

`#pragma acc kernels present(this),\`
`//specify otherdata on the device`
`{`
`#pragma acc loop`
`  for (int i=0; i<x; i++){`
`#pragma acc loop`
`   for (int j=0; j<y; j++){`
`#pragma acc loop`
`       for (int k=0; k<z; k++){`
	The parallelization of the nested i, j and k loops is achieved using kernel clause and so use the rules for a kernel construct. The present data clause, informs the compiler that the data required are already in the device (avoiding unnecessary data movement between host and device).

`#pragma acc parallel present(this),\`
`//specify other data on the device`
`{`
`#pragma acc loop`
`  for (int i=0; i<x; i++){`
`#pragma acc loop`
`   for (int j=0; j<y; j++){`
`#pragma acc loop`
`       for (int k=0; k<z; k++){`
	The parallelization of the nested i, j and k loops is achieved using parallel clause and so use the rules for a parallel construct. The present data clause, informs the compiler that the data required are already in the device (avoiding unnecessary data movement between host and device).

Table 5: Mapping of the kernel and parallel directives to the compute intensive diffusion and activity functions.

Scenario	Directives applied
S1	Kernel directives used in both diffusion and activity functions
S2	Parallel directives used in both diffusion and activity functions
S3	Parallel and Kernel directives in activity and diffusion functions respectively
S4	Kernel and Parallel directives in activity and diffusion functions respectively

2.3 Platform and Tests Performed

The simulations were launched using a PC mounting an Intel Xeon E5-2630 v4 (10 Core, 2.2GHz, 25MB cache) and an NVIDIA GeForce GTX 1080 (1708MHz GPU, 2560 CUDA Cores, 8GB 10010 MHz GDDR5X, revision number 6.1) (CUDA Driver version 8000, NVRM version NVIDIA UNIX x86_64 Kernel Module 375.66) under Ubuntu 17.04 64-bit.

The PGI Community Edition 18.4 compilers for C and C++ applications were used. Case studies 1 & 2 were compiled using the following:

pgcc -c11 -Ox -Mautoinline -Minfo=all -ta=host,multicore,tesla *.c or

pgc++ -Ox -Kieee -Minfo=all -std=c++11 -ta= host,multicore,tesla -ta=time *.cpp

The compilation flag "-Ox" sets the optimisation level to x. The compilation flag "–Kieee" forces the compiler to strict compliance to IEEE 754 floating point standard. The compiling flag "–ta=time" produces information of the time spend in the device and the CUDA grid and block used in every kernel launched. The source code can be compiled to run on both CPU and GPU via the -ta flag using host, allowing for a single thread CPU implementation; multicore, allowing for a multithreaded CPU implementation s; or tesla:cc60, generating code for the NVIDIA Geforce GTX 1080 (Pascal architecture) graphic card. The

Minfo=all command line argument informs the compiler to print out accelerator info and CUDA mappings for the parallelised code. Timing results were gathered using the chrono library and -ta=time flag for the host, multicore and GPU for each of the case studies n = 15 in all cases.

3 RESULTS

3.1 GSRD system

Figure 3 demonstrates the simulation output at four different time points up to 10000 iterations. The top row demonstrates how the emerged patterns are reminiscent of microbial growth (Falconer et al 2005). The bottom row has repeated initial conditions (of the top row) and reveals the repeated patterns that can be achieved.

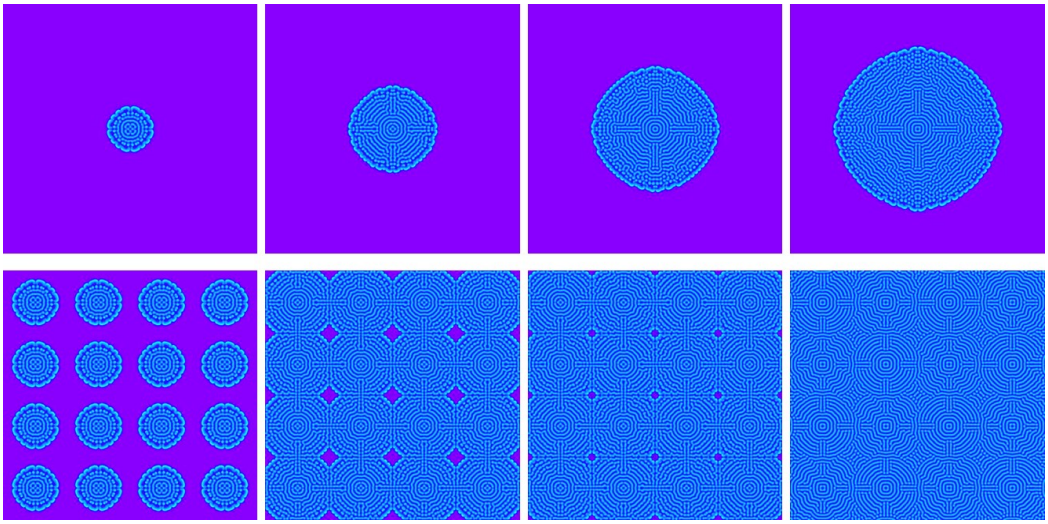

Figure 3: Patterns obtained from GSRD model.

Figure 4 presents the results of the OpenACC implementation run on a multi-core processor and shows how the compute times scale with image size when all ten cores are used. As expected a non-linear scaling is observed and computation time increases with image size.

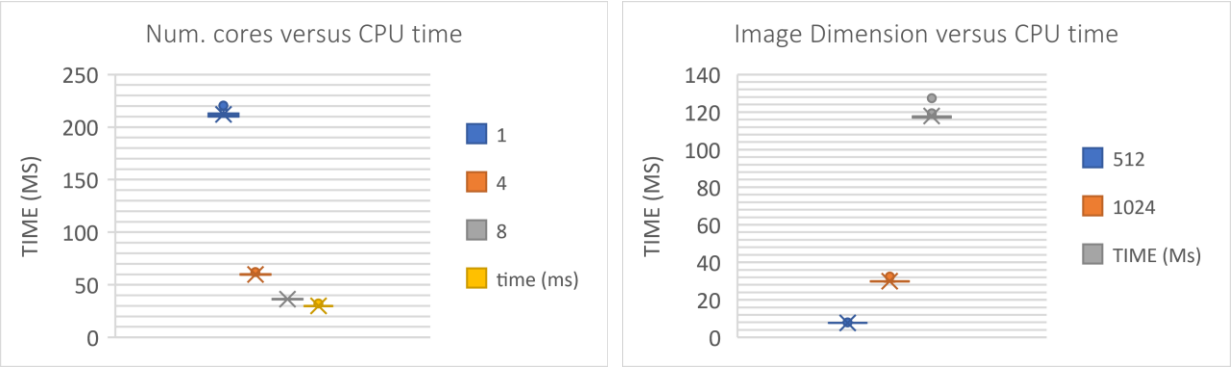

Figure 4a: Box plots (n=15) showing the results of the OpenACC GSRD implementation on a multi-core processor and shows how the timings are affected by the number of cores used. The number of cores can be varied using the environment variable ACC_NUM_CORES. As expected the compute time decreases with a higher number of cores used. Figure 4b:Box plots (n=15)showing the effect of offloading the

computation to the multicore CPU using all available cores for images of size: 512*512, 1024*1024 and 2048*2048.

The timings gathered when the same GSRD computation for different grid sizes was offloaded to the GPU is presented in Fig 5a. It can be observed that the timings are an order of magnitude faster than the equivalent multicore results (Figure 4b).

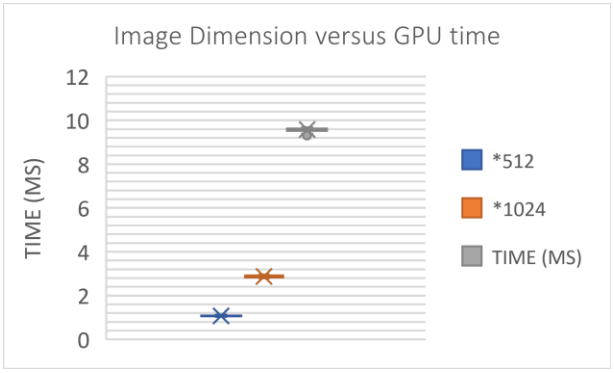

Figure 5: Effect of offloading the computation to GPU for images of size: 512*512, 1024*1024 and 2048*2048. Box plots are presented with n=15.

3.2 MRD system

Figure 6 shows the timings of the MRD being run using a single thread, multithreaded (max number of threads = 10) as well as a GPU application (S3) specified in Table 5. The results demonstrate the substantial speed-ups that can be achieved using some form of parallelization for either the CPU or GPU.

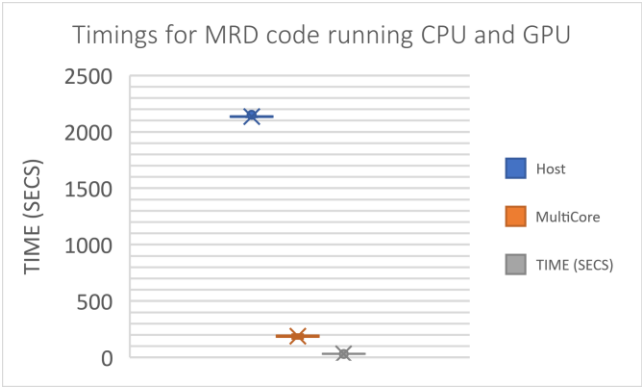

Figure 6: Effect of offloading the computation using the Host, Multicore and GPU (S3) settings.

The final set of results as in Figure 7, looked at the different configurations of using parallel and kernel directives on the two compute intense functions. The results show that the different ways to express the parallelism can have an effect on the timings. It shows that S3 with the `parallel` and `kernels` directives in the activity and diffusion functions respectively maximizes parallelization.

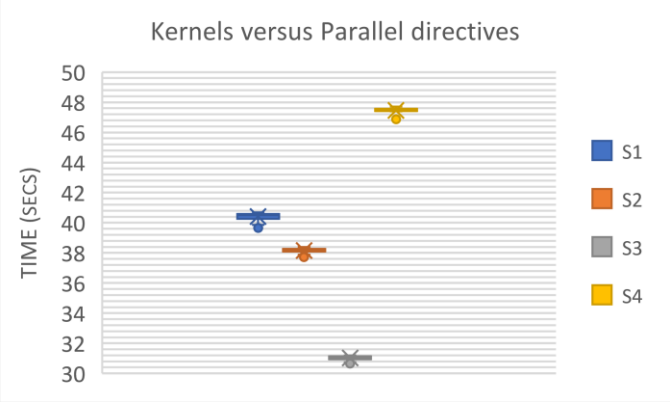

Figure 7: Effect of different configurations of the parallel and kernel directives applied to the diffusion and activity functions – see table 5 for details of the scenarios S1 -S4.

4 DISCUSSION AND CONCLUSIONS

For each case study substantial speed ups were observed using multi (CPU) and many (GPU) core accelerators. OpenACC positions itself as an 'easy to accelerate' option which is certainly true when the existing code has many nested loops and is compute heavy. Another key advantage of OpenACC is the ability to have a single source code generating a unified binary for different architectures, therefore enhancing portability. In the scientific computing community portable and optimised code is highly desirable.

Achieving efficient parallel computation on a GPU with OpenACC requires efficient data flow between host and device. Well-structured data with a clear separation of readable and writable components helps to keep the necessary OpenACC copy directives manageable. For a dynamically allocated buffer the original address only must be used in a copy directive (a derived pointer to some segment of a large buffer generates runtime errors). In the GS code, numerous buffers are individually allocated to avoid cumbersome directives (copying a sub-buffer requires offset, address and sub-length which may impact readability).

Moving from a trivial to a more complex example required the use of higher-level directives for device data creation, deletion and synchronisation linked to unstructured data regions. Unstructured data regions are used to define data regions when scoping does not permit the use of normal structured data regions (e.g. the constructor/destructor of a class). `enter` and `exit data` defines the start and end of an unstructured data lifetime.

It may not always be clear which directive to employ to maximise parallelism (`parallel versus kernels`). It was assumed that the activity function would be more performant using the kernels directive. The activity function is complicated as it possesses many mobile reacting components with several nested loops. It was assumed that by giving control to the compiler more efficient parallel code would be generated by combining the nested loops into single kernel or by creating multiple parallel kernels. The results however show that using the `parallel` directive lead to better performance for the activity function. It is not clear why this is the case and is the focus of future efforts. Other future tests include using the `collapse` and `tile` to further improve performance. The diffusion algorithm may benefit from tiling as threads in the same tile access the same shared, and faster access, memory – limiting the number of reads from global memory therefore maximizing data locality.

It was noted that with MRD that optimizations above -O2 lead to limited parallelisation e.g. the compiler output: "Complex loop carried dependencies prevent parallelization". There is little information on the source of this and perhaps having less conditional statements in the code might permit higher level

optimizations to be applied. The programmer therefore must ensure model correctness when directives are applied and consult the OpenACC complier output to determine what has actually been done. Fine tuning of the application requires a good understanding of how the OpenACC constructs (gang, worker and vector) map to CUDA constructs (grid, block, warp) however this may affect portable optimization potential.

Finally, as can be observed vast speed ups are achieved using the simplest parallel directives applied to nested loops if due attention is given to minimising data transfers, therefore OpenACC can be a fast route to parallelism. This has relevance to those subjects disciplines that exploit RDS such as computer graphics, visual arts and computational biology and ecology.

ACKNOWLEDGMENTS

The research reported in this article was made possible by the financial support of the Natural Environment Research Council (NE/P014208/1). Data underlying this paper can be accessed at https://doi.org/10.17862/cranfield.rd.7560518.

REFERENCES

Falconer, R. E. *et al.* (2005) 'Biomass recycling and the origin of phenotype in fungal mycelia.', *Proceedings. Biological sciences / The Royal Society*, 272(1573), pp. 1727–34. doi: 10.1098/rspb.2005.3150.

Falconer, R. and Houston, A. (2015) 'Visual Simulation of Soil-Microbial System Using GPGPU Technology', *Computation*. Multidisciplinary Digital Publishing Institute, 3(1), pp. 58–71. doi: 10.3390/computation3010058.

Galanter, P. (2014) 'XEPA - Autonomous Intelligent Light and Sound Sculptures That Improvise Group Performances', *Leonardo*. MIT Press, 47(4), pp. 386–393. doi: 10.1162/LEON_a_00844.

Harris, M. (NVIDIA) (2005) 'Mapping Computational Concepts to GPUs', in Pharr, M. and Fernando, R. (eds) *GPU gems 2 : programming techniques for high-performance graphics and general-purpose computation*. Professional, Addison-Wesley.

Malheiros, M. and Walter, M. (2017) 'Pattern formation through minimalist biologically inspired cellular simulation', in *Proceedings of Graphics Interface 2017*. Canadian Human-Computer Communications Society / Société canadienne du dialogue humain-machine (GI 2017), pp. 148–155. doi: 10.20380/GI2017.19.

Meinhardt, H. (2003) 'Pattern Forming Reactions and the Generation of Primary Embryonic Axes BT - Morphogenesis and Pattern Formation in Biological Systems: Experiments and Models', in Sekimura, T. et al. (eds). Tokyo: Springer Japan, pp. 3–19. doi: 10.1007/978-4-431-65958-7_1.

Schram, R. D. (2013) 'Reaction–diffusion model Monte Carlo simulations on the GPU', *Journal of Computational Physics*, 241, pp. 95–103. doi: https://doi.org/10.1016/j.jcp.2013.01.041.

Thompson, D. W. (1992) *On Growth and Form*, *Canto*. Cambridge: Cambridge University Press. doi: DOI: 10.1017/CBO9781107325852.

Turing, A. M. (1952) 'The Chemical Basis of Morphogenesis', *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*. The Royal Society, 237(641), pp. 37–72. Available at: http://www.jstor.org/stable/92463.

Turk, G. (1991) 'Generating Textures on Arbitrary Surfaces Using Reaction-diffusion', *SIGGRAPH Comput. Graph.* New York, NY, USA: ACM, 25(4), pp. 289–298. doi: 10.1145/127719.122749.

Witkin, A. and Kass, M. (1991) 'Reaction-diffusion Textures', in *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM

(SIGGRAPH '91), pp. 299–308. doi: 10.1145/122718.122750.

AUTHOR BIOGRAPHIES

RUTH E FALCONER is a Professor of Complex Systems Modeling at Abertay University. She holds a PhD in Theoretical Ecology and a BSc Hons in Physics. Her research interests are the broader use of video game technology (infrastructure (GPUs) and game engines etc.) to develop playable simulations in the built and natural environments. Her email address is r.falconer@abertay.ac.uk.

ALASDAIR N HOUSTON received his Ph.D. from Abertay University and is currently a freelance software developer. He has worked on many industry and research software projects including game development as well as novel quantitative methods for the 3D image analysis. This has resulted in freely available software, for image quality assessment, image segmentation and morphological analysis. His email address is al.houston@gmail.com.

WILFRED OTTEN is a Full Professor of Soil Biophysics at Cranfield University (United Kingdom). He holds a Ph.D. in Horticultural Sciences from Wageningen University (The Netherlands). His research interests include biological invasions in heterogeneous crop and soil environments, emergent soil properties and soil behavior in response to environmental changes, and the development of novel technologies to characterize porous media. His email address is wilfred.otten@cranfield.ac.uk.

XAVIER PORTELL is a Research Fellow in Modelling Soil Processes at Cranfield University (United Kingdom). He holds a Ph.D. in Agri-food Technology and Biotechnology from Universitat Politècnica de Catalunya (Spain). His research interests lie in the mechanistic modeling of microbial systems at scales directly relevant for microorganisms. His email address is xavier.portell@cranfield.ac.uk