ABSTRACTION AND IMPLEMENTATION OF UNSTRUCTURED GRID ALGORITHMS ON MASSIVELY PARALLEL HETEROGENEOUS ARCHITECTURES

István Zoltán Reguly

Theses of the Ph.D. Dissertation

Pázmány Péter Catholic University
Faculty of Information Technology

Supervisors:
András Oláh Ph.D
Zoltán Nagy Ph.D

Budapest, 2014

1 Introduction

Microprocessor design has faithfully followed Moore's Law for the past forty years. While the number of transistors on a chip has been doubling approximately every two years, other characteristics have been undergoing dramatic changes; due to increasing leakage and practical power dissipation limitations, frequency scaling ground to a halt by 2005. It had become clear that in order to maintain the growth of computational capacity it would be necessary to increase parallelism; multi-core CPUs appeared and supercomputers went through a dramatic increase in processor core count. There is also a resurgence of vector processing; CPUs feature increasingly wide vector processing capabilities, and the emergence of accelerators took these trends to the extreme; GPUs and Intel's Xeon Phi feature many processing cores that have very simplistic execution circuitry compared to CPUs, but contain much wider vector units, they support and expect a high amount of parallelism.

While the economics of processor development has pushed them to gain increasingly higher performance, the economics of memory chip development favoured increasing capacity, not performance. This is quite apparent in their development; while in the 1980's, memory access times and compute cycle times were roughly the same, at present there is at least two orders of magnitude difference, and accounting for multiple cores in modern CPUs, the difference is around a 1000×. Serially executed applications and algorithms therefore face the Von Neumann bottleneck; vast amounts of data have to be transferred through the high-latency, low-bandwidth memory channel, throughput may be much smaller than the rate at which the CPU could work.

The cost of data movement - in terms of energy and latency - is perhaps the greatest challenge facing computing, and therefore locality is of paramount importance. Deep memory hierarchies are introduced in modern architectures to avoid moving data from off-chip memory, which is often several orders of magnitude more expensive than floating point operations. By overlapping computations with data movement, parallelism can also be used to combat latency: this is the approach that GPUs take. At the same time, a further increase in parallelism is necessary to maintain the growth of computational capacity; due to the lack of single-core performance scaling, the departmental, smaller-scale high performance computing (HPC) systems in a few years will consist of the same number of processing elements as the world's largest supercomputers today [1]. Finally, huge parallel processing capabilities

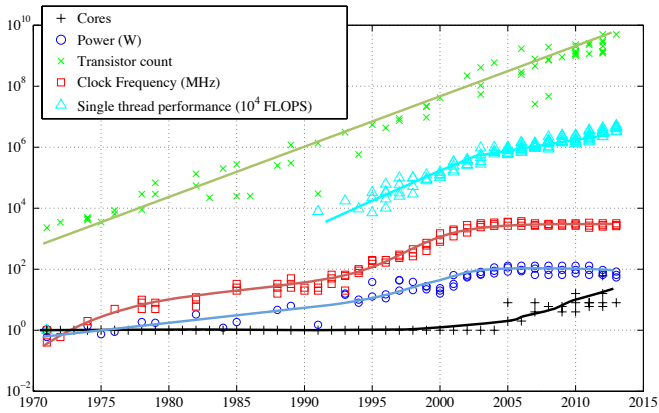


Figure 1: Evolution of processor characteristics

and deep memory hierarchies inevitably result in load balancing issues between concurrent, dependent tasks. These are the three fundamental obstacles to programmability according to [22].

Programming languages still being used today in scientific computing, such as C and Fortran, were designed decades ago with a tight connection to the execution models of the hardware of the time. Code written using these programming models was trivially translated to the hardware's execution model and then into hardware instructions. Over time, hardware and execution models have changed, but mainstream programming models remain the same, resulting in a disparity between the user's way of thinking about programs and what the real hardware is capable and suited to do. While compilers do their best to hide these changes, decades of compiler research has shown that bridging this gap is extremely hard.

There is a growing number of programming languages and extensions that aim to address these issues, but at the same time it is increasingly difficult to write scientific code that delivers high performance and is portable to current and future architectures, because often in-depth knowledge of architectures is required, and hardware-specific optimisations have to be applied. Therefore, there is a push to raise the level of abstraction; describing *what* the program has to do instead of describing *how* exactly to do it, leaving the details to the implementation of the language. Ideally, such a language would deliver generality, productivity and performance, but of course, despite decades of research, no such language exists. Recently, research into Domain Specific Languages (DSLs) applied

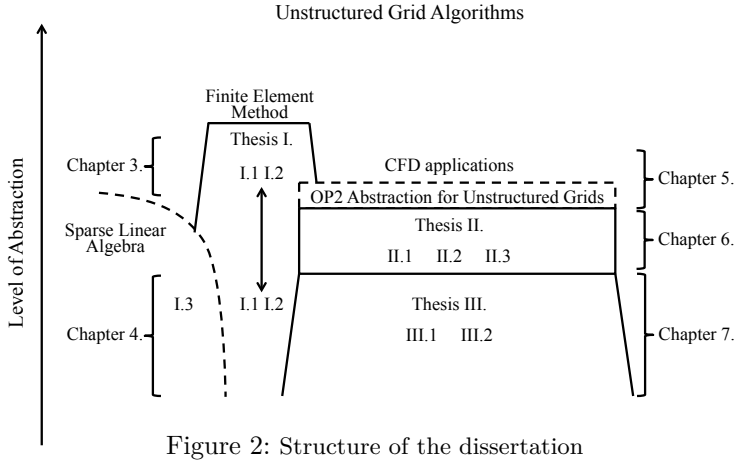


Figure 2: Structure of the dissertation

to different fields in scientific computing has shown that by sacrificing generality, it is possible to achieve performance and productivity. A DSL defines an abstraction for a specific application domain, and provides an Application Programming Interface (API) that can be used to describe computational problems at a higher level. Domain-specific knowledge can then be used to for example re-organise computations to improve locality, break up the problem into smaller parts to improve load-balancing, or map execution to different hardware, applying architecture-specific optimisations. A popular way of classifying these domains is via the 13 dwarfs identified at Berkeley [23]; OP2 [16] is such a domain specific abstraction and library targeting unstructured grid computations, being developed at the University of Oxford.

My main motivation is to address the programming challenges in modern computing; parallelism, locality, load balancing and resilience. For my research, I have chosen to focus on the field of unstructured grid computations. Thus, the aim of this dissertation is to present my research into unstructured grid algorithms, starting out at different levels of abstraction where certain assumptions are made, which in turn are used to reason about and apply transformations to these algorithms. They are then mapped to computer code, with a focus on the dynamics between the programming model, the execution model and the hardware; investigating how the parallelism and the deep memory hierarchies available on modern heterogeneous hardware can be utilised optimally.

Figure 2 shows the structure of the dissertation, the first part of my research studies the Finite Element Method, therefore starts at a rela-

tively high level of abstraction, allowing a wide range of transformations to the numerical methods involved. I present results involving changes to the balance of computations, communications, and data structures (Thesis I.1 and I.2). My research into the linear solve phase of the method yields results that contribute not only to the FEM and unstructured grids but to the related field of sparse linear algebra as well (Thesis I.3). Following the first part that focused on challenges in the context of the finite element method, I broaden the scope of my research by addressing general unstructured grid algorithms that are defined through the OP2 domain specific library[16]. I have started contributing to the project a year after its launch, carrying out research on different areas, the results of which form Thesis groups II. and III. OP2's abstraction for unstructured grid computations covers the finite element method, but also others such as the finite volume method. The entry point here, that is the level of abstraction, is lower than that of the FEM, thus there is no longer control over the numerical method, however it supports a much broader range of applications. The second part of my research investigates possible transformations to the execution of computations defined through the OP2 abstraction in order to be able to address the challenges of resiliency (Thesis II.1), locality (Thesis II.2), and utilisation of resources (Thesis II.3) at a higher level, that is not concerned with the exact implementation. Finally, the third part of my research presents results on how an algorithm defined once through OP2 can be automatically mapped to a range of contrasting programming languages, execution models and hardware, such as GPUs (Thesis III.1), CPUs, and the Xeon Phi (Thesis III.2). I show how execution is organised on large scale heterogeneous systems, utilising layered programming abstractions, across deep memory hierarchies and many levels of parallelism.

2 Methods and Tools

During the course of my research a range of numerical methods and analytical methods were used in conjunction with different programming languages, programming and execution models, and hardware. Indeed, one of my goals is to study the interaction of these in today's complex systems. The first part of my research (Thesis group I.) is based on a popular discretisation method for Partial Differential Equations (PDEs); the Finite Element Method - I used a Poisson problem, implemented loosely based on [21]. During the study of the solution of sparse linear systems,

I used the Conjugate Gradient iterative method preconditioned by the Jacobi and the Symmetric Successive Over-Relaxation (SSOR) method [24]. The sparse matrix-vector multiplication, as the principal building block for sparse linear algebra algorithms, is studied in further detail. This initial part of my research served as an introduction to unstructured grid algorithms, gaining invaluable experience that would later be applied to the rest of my research.

The second and third parts of my research are based on the OP2 Domain Specific Language (or "active library"), introduced by Prof. Mike Giles at the University of Oxford [16], its abstraction carried over from OPlus [18]. There is a suite of finite volume applications that were written using the OP2 abstraction and are used to evaluate the algorithms presented in this dissertation; a benchmark simulating airflow around the wing of an aircraft (Airfoil) [25], a tsunami simulation software called Volna [20], and a large-scale production application, Hydra [19], used by Rolls-Royce plc. for the design and simulation of turbomachinery. While I do not claim authorship of the original codes, I did do most of the work transforming the latter two to the OP2 abstraction. OP2 and the Airfoil benchmark are available at [17].

Computer code was implemented using either the C or the Fortran language, using the CUDA's language extensions when programming for GPUs. Python was used to facilitate text manipulation and code generation. A number of parallel programming models were employed to support the hierarchical parallelism present in modern computer systems; at the highest level, message passing for distributed memory parallelism, using MPI libraries. For coarse-grained shared memory parallelism I used simultaneous multithreading (SMT), using OpenMP and CUDA thread blocks. Finally for fine-grained shared memory parallelism I used either Single Instruction Multiple Threads (SIMT) using CUDA, or Single Instruction Multiple Data (SIMD) using Intel vector intrinsics.

A range of contrasting hardware platforms were used to evaluate the performance of algorithms and software. When benchmarking at a small scale, single workstations were used, consisting of dual-socket Intel Xeon server processors (Westmere X5650, Sandy-Bridge E2640 and E2670). The accelerators used were: an Intel Xeon Phi 5110P, and NVIDIA Tesla cards (C2070, M2090, K20, K20X, K40). For large-scale tests the following supercomputers were used: HECToR (the UK's national supercomputing machine, a Cray XE6, with 90112 AMD Opteron cores), Emerald (the UK's largest GPU supercomputer, with 372 NVIDIA M2090 GPUs, 512 cores each) and Jade (Oxford University's GPU cluster, with

16 NVIDIA K20 GPUs, 2496 cores each). Timings were collected using standard UNIX system calls, usually ignoring the initial set-up cost (due to e.g. file I/O) because production runs of the benchmarked applications have an execution time of hours or days, compared to which, set-up costs are negligible. In most cases, results are collected from 3-5 repeated runs and averaged. Wherever possible, I provide both absolute and relative performance numbers, such as achieved bandwidth (in GB/s), computational throughput (10^9 Floating Operations Per Second - GFLOPS), and speedup over either a reference implementation on the GPU or a fully utilised CPU, not just a single core.

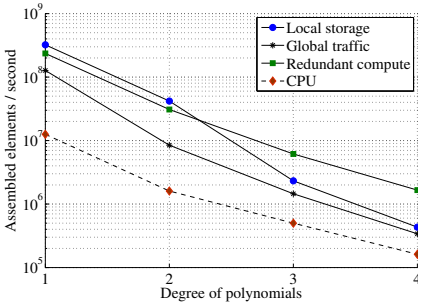
3 New scientific results

Thesis group I. *(area: Finite Element Method)* - I have introduced algorithmic transformations, data structures and new implementations of the Finite Element Method (FEM) and corresponding sparse linear algebra methods on GPUs, in order to address different aspects of the concurrency, locality, and memory challenges and quantified the trade-offs.

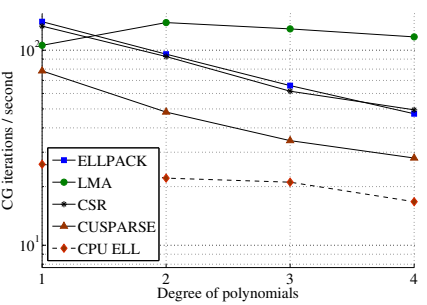
Related publications: [4, 9, 12, 13].

Thesis I.1. - By applying transformations to the FE integration that trade off computations for communications and local storage, I have designed and implemented new mappings to the GPU, and shown that the redundant compute approach delivers high performance, comparable to classical formulations for first order elements, furthermore, it scales better to higher order elements without loss in computational throughput.

Through algorithmic transformations to the Finite Element integration, I gave per-element formulations that have different characteristics in terms of the amount of computations, temporary memory usage, and spatial and temporal locality in memory accesses. The three variants are: (1 - redundant compute), where the outer loop is over pairs of degrees of freedom and the inner loop over quadrature points recomputing the Jacobian for each one, (2 - local storage) structured as (1) but Jacobians are pre-computed and re-used in the innermost loop, effectively halving the number of computations, and (3 - global memory traffic), that is commonly used in Finite Element codes, where the outermost loop is over quadrature points, computing the Jacobian once, and then the inner loop is over pairs of degrees of freedom, adding the contribution



(a) FE integration transformations



(b) Iterative solution and data structures

Figure 3: Performance of Finite Element Method computations mapped to the GPU

from the given quadrature point to the stiffness values. As illustrated in Figure 3a, I have demonstrated that approach (1) is scalable to high degrees of polynomials because only the number of computations changes, whereas with (2) the amount of temporary storage, and with (3) the number of memory transactions also increase. Implementations of these variants in CUDA applied to a Poisson problem show that for low degree polynomials (1) and (2) perform almost the same, but at higher degrees (1) is up to 8× faster than (2), and generally 3× faster than (3). Overall, an NVIDIA C2070 GPU is demonstrated to deliver up to 400 GFLOPS (66% of the ideal[1]), it is up to 10× faster than a two-socket Intel Xeon X5650 processor, and up to 120× faster than a single CPU core.

Thesis I.2. - I introduced a data structure for the FEM on the GPU, derived storage and communications requirements, shown its applicability to both the integration and the sparse iterative solution, and demonstrated superior performance due to improved locality.

One of the key challenges to performance in the FEM is the irregularity of the problem and therefore of the memory accesses, which is most apparent during the matrix assembly and the sparse iterative solution phases. By storing stiffness values on a per-element basis laid out for optimal access on massively parallel architectures, I have shown that it is possible to regularise memory accesses during integration by postponing the handling of race conditions until the iterative solution

[1]The same card delivers 606 Giga Floating Operations per Second (GFLOPS) on a dense matrix-matrix multiplication benchmark

Table 1: Performance metrics on the test set of 44 matrices.

	CUSPARSE	Fixed rule	Tuned
Throughput single GFLOPS/s	7.0	14.5	15.6
Throughput double GFLOPS/s	6.3	8.8	9.2
Min Bandwidth single GB/s	28.4	58.9	63.7
Min Bandwidth double GB/s	38.7	54.0	56.8
Speedup single over CUSPARSE	1.0	2.14	2.33
Speedup double over CUSPARSE	1.0	1.42	1.50

phase, where it can be addressed more efficiently. This approach, called the Local Matrix Approach (LMA), consists of a storage format and changes to the FE algorithms in both the assembly and the solution phases, and is compared to traditional storage formats, such as CSR and ELLPACK on GPUs. I show that it can be up to two times faster during both phases of computations, due to reduced storage costs, as shown in Figure 3b, and regularised memory access patterns. A conjugate gradient iterative solver is implemented, supporting all three storage formats, using a Jacobi and a Symmetric Successive Over-Relaxation (SSOR) preconditioner, performance characteristics are analysed, and LMA is shown to deliver superior performance in most cases.

Thesis I.3. - I have parametrised the mapping of sparse matrix-vector products (spMV) for GPUs, designed a new heuristic and a machine learning algorithm in order to improve locality, concurrency and load balancing. Furthermore, I have introduced a communication-avoiding algorithm for the distributed execution of the spMV on a cluster of GPUs. My results improve upon the state of the art, as demonstrated on a wide range of sparse matrices from mathematics, computational physics and chemistry.

The sparse matrix-vector multiplication operation is a key part of sparse linear algebra; virtually every algorithm uses it in one form or another. The most commonly used storage format for sparse matrices is the compressed sparse row (CSR) format; it is supported by a wide range of academic and industrial software, thus I chose it for the basis for my study. By appropriately parametrising the multiplication operation for GPUs, using a dynamic number of cooperating threads to carry out the dot product between a row of the matrix and the multiplicand vector, in addition to adjusting the thread block size and the granularity of work assigned to thread blocks, it is possible to outperform the state of the art CUSPARSE library.

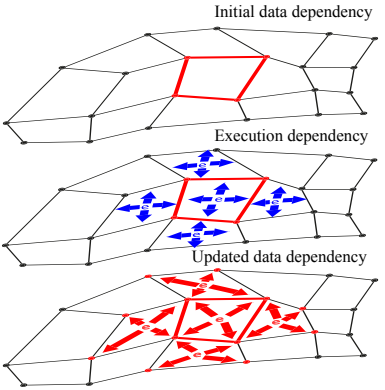
I have introduced an O(1) heuristic that gives near-optimal values for these parameters that immediately results in 1.4-2.1× performance increase. Based on the observation that in iterative solvers, the spMV is evaluated repeatedly with the same matrix, I have designed and implemented a machine learning that tunes these parameters and increases performance by another 10-15% in at most 10 iterations, achieving 98% of the optimum, found by exhaustive search. Results are detailed in Table 1. I have also introduced a communication avoiding algorithm for the distributed memory execution of the spMV, that uses overlapping graph partitions to perform redundant computations and decrease the frequency of communications, thereby mitigating the impact of latency, resulting in up to 2× performance increase.

Thesis group II. *(area: High-Level Transformations with OP2)* - I address the challenges of resilience, the expression and exploitation of data locality, and the utilisation of heterogeneous hardware, by investigating intermediate steps between the abstract specification of an unstructured grid application with OP2 and its parallel execution on hardware; I design and implement new algorithms that apply data transformations and alter execution patterns.

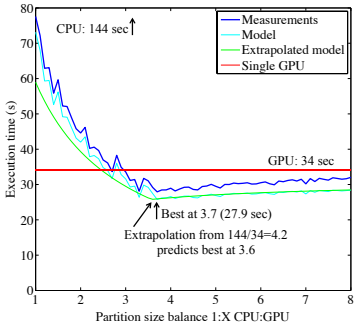
Related publications [2, 3, 5, 10]

Thesis II.1. - I have designed and implemented a checkpointing method in the context of OP2 that can automatically locate points during execution where the state space is minimal, save data and recover in the event of a failure.

As the number of components in high performance computing systems increases, the mean time between hardware or software failures may become less than the execution time of a large-scale simulation. I have introduced a checkpointing method in order to provide means to recover after a failure, that relies on the information provided through the OP2 API to reason about the state space of the application at any point during the execution and thereby to (1) find a point where the size of the state space is minimal and save it to disk and (2) in case of a failure, recover by fast-forwarding to the point where the last backup happened. This is facilitated by the OP2 library, in a way that is completely opaque to the user, requiring no intervention except for the re-launch of the application after the failure. This ensures the resiliency of large-scale simulations.



(a) Resolving data and execution dependencies for unstructured mesh tiling



(b) Heterogeneous execution and its modelling

Figure 4: High-level transformations and models based on the OP2 abstraction

Thesis II.2. - I gave an algorithm for redundant compute tiling in order to provide cache-blocking for modern architectures executing general unstructured grid algorithms, and implemented it in OP2, relying on run-time dependency analysis and delayed execution techniques.

Expressing and achieving memory locality is one of the key challenges of high performance programming; but the vast majority of scientific codes are still being designed and implemented in a way that only supports very limited locality; it is common practice to carry out one operation on an entire dataset and then another - as long as the dataset is larger than the on-chip cache, this will result in repeated data movement. However, doing one operation after the other on just a part of the dataset is often non-trivial due to data dependencies. I have devised and implemented a tiling algorithm for general unstructured grids defined through the OP2 abstraction, that can map out these data dependencies, as illustrated in Figure 4a, and enable the concatenation of operations over a smaller piece of the dataset, ideally resident in cache, thereby improving locality. The tiling algorithm can be applied to any OP2 application without the intervention of the user.

Thesis II.3. - I gave a performance model for the collaborative, heterogeneous execution of unstructured grid algorithms where multiple hard-

ware with different performance characteristics are used, and introduced support in OP2 to address the issues of hardware utilisation and energy efficiency.

Modern supercomputers are increasingly designed with many-core accelerators such as GPUs or the Xeon Phi. Most applications running on these systems tend to only utilise the accelerators, leaving the CPUs without useful work. In order to make the best use of these systems, all available resources have to be kept busy, in a way that takes their different performance characteristics into account. I have developed a model for the hybrid execution of unstructured grid algorithms, giving a lower bound for expected performance increase and added support for utilising heterogeneous hardware in OP2, validating the model and evaluating performance, as shown in Figure 4b.

Thesis group III. *(area: Mapping to Hardware with OP2)* - One of the main obstacles in the way of the widespread adoption of domain specific languages is the lack of evidence that they can indeed deliver performance and future proofing to real-world codes. Through the Airfoil benchmark, the tsunami-simulation code Volna and the industrial application Hydra, used by Rolls-Royce plc. for the design of turbomachinery, I provided conclusive evidence that an unstructured grid application, written once using OP2, can be automatically mapped to a range of heterogeneous and distributed hardware architectures at near-optimal performance, thereby providing maintainability and longevity to these codes.

Related publications: [2, 3, 5, 6, 8, 11, 14, 15]

Thesis III.1. - I have designed and developed an automated mapping process to GPU hardware that employs a number of data and execution transformations in order to make the best use of limited hardware resources, the multiple levels of parallelism and memory hierarchy, which I proved experimentally.

Mapping execution to GPUs involves the use of the Single Instruction Multiple Threads (SIMT) model, and the CUDA language. I have created an automatic code generation technique that, in combination with run-time data transformation, facilitates near-optimal execution on NVIDIA Kepler-generation GPUs. I show how state-of-the-art optimisations can be applied through the code generator, such as the use of the read-only cache, or data structure transformation from Array-of-Structures (AoS) to Structure-of-Arrays (SoA), in order to

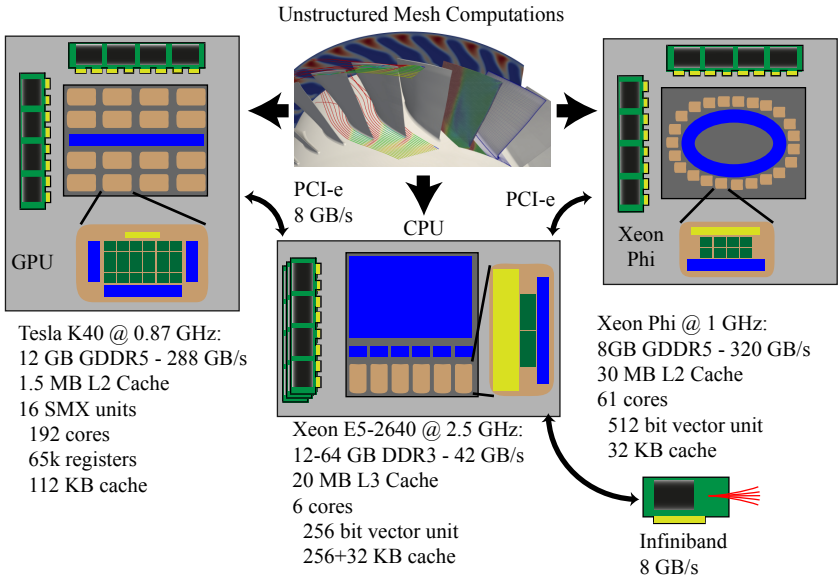


Figure 5: The challenge of mapping unstructured grid computations to various hardware architectures and supercomputers

make better use of the execution mechanisms and the memory hierarchy. These are then deployed to a number of applications and tested on different hardware, giving 2-5× performance improvement over fully utilised Intel Xeon CPUs. Performance characteristics are analysed, including compute and bandwidth utilisation, to gain a deeper understanding of the interaction of software and hardware, and to verify that near-optimal performance is indeed achieved. I discuss how OP2 is able to utilise supercomputers with many GPUs, by automatically handling data dependencies and data movement using MPI, and I demonstrate strong and weak scalability on Hydra.

Thesis III.2. - I have designed and implemented an automated mapping process to multi- and many-core CPUs, such as Intel Xeon CPUs and the Intel Many Integrated Cores (MIC) platform, to make efficient use of multiple cores and large vector units in the highly irregular setting of unstructured grids, which I proved experimentally.

Modern CPUs feature increasingly longer vector units, their utilisation is essential to achieving high performance. However, compilers consistently fail at automatically vectorising irregular codes, such as un-

structured grid algorithms, therefore low-level vector intrinsics have to be used to ascertain the utilisation of vector processing capabilities. I have introduced a code generation technique that is used in conjunction with C++ classes and operator overloading for wrapping vector intrinsics and show how vectorised execution can be achieved through OP2, by automatically gathering and scattering data. Performance is evaluated on high-end Intel Xeon CPUs and the Xeon Phi, and a 1.5-2.5× improvement is demonstrated over the non-vectorised implementations. In-depth analysis reveals what hardware limitations determine the performance of different stages of computations on different hardware. I demonstrate that these approaches are naturally scalable to hundreds or thousands of cores in modern supercomputers, evaluating strong and weak scalability on Hydra.

4 Applicability of the results

The applicability of the results related to the Finite Element Method are many-fold; the practice of designing algorithms that have different characteristics in terms of computations, memory requirements and memory traffic is useful in different contexts as well, but the results can be directly used when designing a general-purpose Finite Element library. There are already some libraries, such as ParaFEM [26], which take a similar matrix-free approach as LMA, therefore my results are directly applicable, should GPU support be introduced, or the need for more advanced sparse linear solvers arise. Results concerning the sparse matrix-vector product are pertinent to a much wider domain of applications: sparse linear algebra. The heuristic published in [9] was subsequently adopted by the NVIDIA CUSPARSE library [27], and the runtime auto-tuning of parameters is a strategy that, though few libraries have adopted, could become standard practice as hardware becomes even more diverse and thus performance predictions become more uncertain. During my internship at NVIDIA, I have developed the distributed memory functionality of the sparse linear solver software package that became AmgX [28], incorporating many of the experiences gained working on the FEM, designing it from the outset in a way so that optimisations such as redundant computations for avoiding communications could be adopted.

Results of the research carried out in the context of the OP2 framework are immediately applicable to scientific codes that use OP2; after converting the Volna tsunami simulation code [20] to OP2, it was adopted by Serge Guillas's group at the University College London and

subsequently by the Indian Institute of Science in Bangalore, and it is currently being used for the simulation of tsunamis in conjunction with uncertainty quantification, since the exact details of the under-sea earthquakes are often not known. Similarly, the conversion of Rolls-Royce Hydra [19] to OP2 is considered a success, performance bests the original, and support for modern heterogeneous architectures is introduced, thereby future-proofing the application; discussions regarding the use of the OP2 version in production are ongoing. However, many of these results, especially the ones under Thesis II that describe generic algorithms and procedures, are relevant to other domains in scientific computations as well; our subsequent research into structured grid computations is going to be employing many of these techniques, and some are already used in research on molecular dynamics carried out in collaboration with chemical physicists that resulted in [7].

5 Acknowledgements

First of all, I am most grateful to my supervisors, Dr. András Oláh and Dr. Zoltán Nagy for their motivation, guidance, patience and support, as well as to Prof. Tamás Roska for the inspiring and thought-provoking discussions that led me down this path. I thank Prof. Barna Garay for all his support, for being a pillar of good will and honest curiosity. I am immensely grateful to Prof. Mike Giles for welcoming me into his research group during my stay in Oxford, for guiding and supporting me, and for opening up new horizons.

I would like to thank all my friends and colleagues with whom I spent these past few years locked in endless debate; Csaba Józsa for countless hours of discussion, Gábor Halász, Endre László, Helga Feiszthuber, Gihan Mudalige, Carlo Bertolli, Fabio Luporini, Zoltán Tuza, János Rudan, Tamás Zsedrovits, Gábor Tornai, András Horváth, Dóra Bihary, Bence Borbély, Dávid Tisza and many others, for making this time so much enjoyable.

I thank the Pázmány Péter Catholic University, Faculty of Information Technology, and Prof. Péter Szolgay, for accepting me as a doctoral student and supporting me throughout, and the University of Oxford, e-Research Centre. I am thankful to Jon Cohen, Robert Strzodka, Justin Luitjens, Simon Layton and Patrice Castonguay at NVIDIA for the summer internship, while working on AmgX together I have learned a tremendous amount.

Finally, I will be forever indebted to my family, enduring my presence

and my absence, and for being supportive all the while, helping me in every way imaginable.

References

Journal publications by the author

[1] M. B. Giles and **I. Z. Reguly**. "Trends in high performance computing for engineering calculations". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* (2014). Invited paper, accepted with minor revisions.

[2] G. R. Mudalige, M. B. Giles, J. Thiyagalingam, **I. Z. Reguly**, C. Bertolli, P. H. J. Kelly, and A. E. Trefethen. "Design and initial performance of a high-level unstructured mesh framework on heterogeneous parallel systems". In: *Parallel Computing* 39.11 (2013), pp. 669–692. DOI: 10.1016/j.parco.2013.09.004.

[3] M. B. Giles, G. R. Mudalige, B. Spencer, C. Bertolli, and **I. Z. Reguly**. "Designing OP2 for GPU Architectures". In: *Journal Parallel and Distributed Computing* 73.11 (Nov. 2013), pp. 1451–1460. DOI: 10.1016/j.jpdc.2012.07.008.

[4] **I. Z. Reguly** and M.B Giles. "Finite Element Algorithms and Data Structures on Graphical Processing Units". In: *International Journal of Parallel Programming* (2013). ISSN: 0885-7458. DOI: 10.1007/s10766-013-0301-6.

[5] **I. Z. Reguly**, G. R. Mudalige, C. Bertolli, M. B. Giles, A. Betts, P. H. J. Kelly, and D. Radford. "Acceleration of a Full-scale Industrial CFD Application with OP2". In: *submitted to ACM Transactions on Parallel Computing* (2013). Available at: http://arxiv.org/abs/1403.7209.

[6] **I. Z. Reguly**, E. László, G. R. Mudalige, and M. B. Giles. "Vectorizing Unstructured Mesh Computations for Many-core Architectures". In: *submitted to Concurrency and Computation: Practice and Experience special issue on programming models and applications for multicores and manycores* (2014).

[7] L. Rovigatti, P. Šulc, **I. Z. Reguly**, and F. Romano. "A comparison between parallelisation approaches in molecular dynamics simulations on GPUs". In: *submitted to The Journal of Chemical Physics* (2014). Available at: http://arxiv.org/abs/1401.4350.

International conference publications by the author

[8] G. R. Mudalige, **I. Z. Reguly**, M. B. Giles, C. Bertolli, and P. H. J. Kelly. "OP2: An Active Library Framework for Solving Unstructured Mesh-based Applications on Multi-Core and Many-Core Architectures." In: *Proceedings of Innovative Parallel Computing (InPar '12)*. San Jose, CA. US.: IEEE, May 2012. DOI: 10.1109/InPar.2012.6339594.

[9] **I. Z. Reguly** and M. B. Giles. "Efficient sparse matrix-vector multiplication on cache-based GPUs." In: *Proceedings of Innovative Parallel Computing (InPar '12)*. San Jose, CA. US.: IEEE, May 2012. DOI: 10.1109/InPar.2012.6339602.

[10] M.B. Giles, G.R. Mudalige, C. Bertolli, P.H.J. Kelly, E. László, and **I. Z. Reguly**. "An Analytical Study of Loop Tiling for a Large-Scale Unstructured Mesh Application". In: *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* 2012, pp. 477–482.

[11] **I. Z. Reguly**, E. László, G. R. Mudalige, and M. B. Giles. "Vectorizing Unstructured Mesh Computations for Many-core Architectures ". In: *Proceedings of the 2014 International Workshop on Programming Models and Applications for Multicores and Manycores*. PMAM '14. Orlando, Florida, USA: ACM, 2014. DOI: 10.1145/2560683.2560686.

Other publications by the author

[12] **I. Z. Reguly** and M. B. Giles. "Efficient and scalable sparse matrix-vector multiplication on cache-based GPUs." In: *Sparse Linear Algebra Solvers for High Performance Computing Workshop*. July 8-9, Warwick, UK, 2013.

[13] **I. Z. Reguly**, M. B. Giles, G. R. Mudalige, and C. Bertolli. "Finite element methods in OP2 for heterogeneous architectures". In: *European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012)*. September 10-14, Vienna, Austria, 2012.

[14] **I. Z. Reguly**, M. B. Giles, G. R. Mudalige, and C. Bertolli. "OP2: A library for unstructured grid applications on heterogeneous architectures". In: *European Numerical Mathematics and Advanced Applications (ENUMATH 2013)*. August 26-30, Lausanne, Switzerland, 2013.

[15] **I. Z. Reguly**, G. R. Mudalige, C. Bertolli, M. B. Giles, A. Betts, P. H. J. Kelly., and D. Radford. "Acceleration of a Full-scale Industrial CFD Application with OP2". In: *UK Many-Core Developer Conference 2013 (UKMAC'13)*. December 16, Oxford, UK, 2013.

Related publications

[16] M. B. Giles, G. Mudalige, Z. Sharif, G. Markall, and P. H. J. Kelly. "Performance Analysis and Optimization of the OP2 Framework on Many-Core Architectures". In: *The Computer Journal* 55.2 (2012), pp. 168–180.

[17] *OP2 GitHub Repository*. `https://github.com/OP2/OP2-Common`. 2013.

[18] P. I. Crumpton and M. B. Giles. "Multigrid Aircraft Computations Using the OPlus Parallel Library". In: *Parallel Computational Fluid Dynamics: Implementations and Results Using Parallel Computers* (). A. Ecer, J. Periaux, N. Satofuka, and S. Taylor, (eds.), North-Holland, 1996., pp. 339–346.

[19] Michael B. Giles, Mihai C. Duta, Jens-Dominik Müller, and Niles A. Pierce. "Algorithm Developments for Discrete Adjoint Methods". In: *AIAA Journal* 42.2 (2003), pp. 198–205.

[20] Denys Dutykh, Raphaël Poncet, and Frédéric Dias. "The VOLNA code for the numerical modeling of tsunami waves: Generation, propagation and inundation". In: *European Journal of Mechanics - B/Fluids* 30.6 (2011), pp. 598 –615. ISSN: 0997-7546. DOI: `j.euromechflu.2011.05.005`.

[21] M. S. Gockenbach. *Understanding and implementing the finite element method*. SIAM, 2006. ISBN: 978-0-89871-614-6.

[22] B. Dally. "Power, Programmability, and Granularity: The Challenges of ExaScale Computing". In: *Proceedings of the Parallel Distributed Processing Symposium (IPDPS'11)*. 2011, pp. 878–878. DOI: `10.1109/IPDPS.2011.420`.

[23] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. *The Landscape of Parallel Computing Research: A View from Berkeley*. Tech. rep. UCB/EECS-2006-183. EECS Department, University of California, Berkeley, Dec. 2006. URL: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html`.

[24] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. ISBN: 0898715342.

[25] M. B. Giles, D. Ghate, and M. C. Duta. "Using Automatic Differentiation for Adjoint CFD Code Development". In: *Computational Fluid Dynamics Journal* 16.4 (2008), pp. 434–443.

[26] I.M. Smith, D.V. Griffiths, and L. Margetts. *Programming the Finite Element Method*. Wiley, 2013. ISBN: 9781118535936. URL: `http://books.google.co.uk/books?id=iUaaAAAAQBAJ`.

[27] NVIDIA. *cuSPARSE library, last accessed Dec 20th.* `http://developer.nvidia.com/cuSPARSE`. 2012.

[28] *NVIDIA AmgX Library*. `https://developer.nvidia.com/amgx`. 2013.