Final Report

Locality and Load-Balancing in Heterogeneous Simulations and Architectures

1. Introduction

Almost four years ago, when I wrote the proposal for this grant, the fundamental motivation for my research was the fact that the hardware landscape in high performance computing (HPC) was growing more diverse. This went hand-in-hand with the increasing number of software approaches to utilize these hardware, which posed a considerable challenge for software developers for whom performance was a key criteria. Being able to productively create and maintain code that is capable of efficiently utilizing a variety of HPC architectures is an unsolved problem in general. I recently published a review paper [14] on the challenges of achieving performance, productivity, and portability in computation fluid dynamics (and more generally in HPC), where I discuss a number of approaches that try to deliver these qualities by sacrificing generality to different degrees. Indeed, this is the core motivation of my research in this project: to deliver performance on the latest HPC hardware systems to domain scientists without the pain of having to understand, and code for, these architectures. Therefore, my research focused on benchmarking, modeling, and automating the mapping to these hardware for the restricted domain of structured and unstructured mesh computations. I have also set out to expand our abstractions and create new ones to tackle problem classes that were not covered before – making our research more widely applicable.

Hardware and software trends in the last three years have underlined the importance of this work. While Intel is still dominant on the CPU market, recognizing the requirements of HPC and AI, it has entered the discrete graphical processing (GPU) market with their Xe lineup of GPUs, and won a contract from the US DoE to build the first exascale supercomputer called Aurora. AMD has also returned to the HPC market with a strong CPU offering, and its GPUs are gaining popularity in HPC and AI as well – in fact AMD won contracts to build two exascale systems for the US DoE that will utilize its CPUs and GPUs. ARM has also become a major player in HPC: the current number one supercomputer, Fugaku (based in Japan), is built on the ARM A64FX architecture. NVIDIA of course is still the most popular GPU manufacturer, but unlike Intel and AMD, they do not design their own CPUs, giving them a disadvantage in how closely CPUs and GPUs can be coupled (which is a key selling point for Intel and AMD). Perhaps in part because of this, NVIDIA is in the process of buying ARM. Looking ahead, there will be at least three flavors of pre-exascale and exascale HPC systems – and the software stacks supported by them is just as diverse; huge amounts of money was spent enabling codes to run on NVIDIA GPUs using OpenACC and CUDA, however, neither of these are compatible with Intel's and AMD's GPUs.

At the same time, FPGAs have significantly gained in popularity, and while they are still not capable of addressing nearly as many applications as general-purpose CPUs and GPUs, they are becoming more versatile and easier to program for. Intel has bought FPGA-maker Altera, and is aggressively pushing their use with OpenCL and the OneAPI software environment. As a response, AMD is now in the process of buying Xilinx.

One of the key means to achieve my goals was the study of memory hierarchies, and work towards identifying and tackling bottlenecks. These have become more complex too; Intel has introduced the byte-addressable non-volatile Optane memory, which can be used in place, or

in conjunction with regular RAM, and SSDs are now commonly used in large systems too to decrease the load on the parallel file system. Large systems now have up to 5 levels in the memory hierarchy (GPU on-chip memory, GPU off-chip/stacked memory, CPU memory, on-node non-volatile memory, parallel file system). On many systems, the CPU-GPU connection is still a considerable bottleneck – PCI-e is only 16 GB/s, compared to 1000+ GB/s on the GPU and 100-200 GB/s on the CPU – NVIDIA is already using NVLink between its GPUs and to IBM CPUs, and Intel and AMD are working on proprietary solutions too.
There is clearly tremendous diversity in HPC architectures, and the barrier to entry has been dramatically lowered in recent years thanks to cloud providers, who are offering easy and relatively inexpensive access to these high-end, otherwise very expensive, platforms. This has increased pressure on HPC codes to support as many of them as possible; researchers want to run their codes on whatever infrastructure they already have, and increasingly in the cloud, where they will choose the hardware that is delivering the most performance for the money.

Having outlined the recent trends that affect my research, in the following, I describe my results how they tie into these developments.

2. Unstructured mesh computations

Heterogeneity comes in many forms in modern HPC applications: the programming language (most commonly C/C++ or Fortran), the parallelization methods (MPI, OpenMP, CUDA, OpenACC, OpenCL, and many others), the computations themselves, the compiler, and the target architecture. We have explored some of this space in our paper [1] studying a representative set of unstructured mesh computations on GPUs, using different languages, parallelizations, and compilers – it demonstrated the trade-offs, and the fundamental challenge faced by application developers targeting particular combinations of these technologies. There is simply no “best” choice, and being able to automatically generate and utilize a large number of them, as our OP2 library does, allows the users to easily pick the best option. We show that performance particularly on GPUs is strongly influenced by memory locality – considering that unlike CPUs, they have very little on-chip cache. We developed a roofline performance model to characterize the efficiency of various combinations.
Motivated by this, we further investigated how data locality can be improved for GPUs in unstructured mesh computations – looking at key kernels from industrial and academic codes, we created a way to maximize locality for the very limited resources of one GPU thread block using graph partitioning methods [12]. We showed that we can consistently improve upon the state of the art by 20-150%, by exploiting local regularities in unstructured mesh computations.
We have also introduced support for the SYCL programing model, that Intel’s OneAPI framework is based on, in OP2 and evaluated its performance across Intel, AMD, ARM CPUs, as well as NVIDIA, AMD and Intel Xe GPUs, developing a detailed roofline model for each [16,19], which for the first time enabled our library to target AMD and Intel GPUs. SYCL is close to fulfilling its promise of supporting a large number of target architectures, the code is not performance portable – different architectures still require different optimizations.
For the OP2 library to automatically generate a range of different parallelizations for the same high-level code, we must ensure the reliability and correctness of these transformations. Therefore out research extents to utilizing the Clang/LLVM compiler infrastructure to parse

the high-level OP2 code, and then generate highly-optimized sequential, OpenMP, OpenMP offload, and CUDA code [6].
We also demonstrated the utility and efficiency of the OP2 library in supporting real-life applications. In cooperation with geophysicists and statisticians from UCL and UCD, we have developed the VOLNA-OP2 code that allows the simulations of tsunamis from inundation to run-up [8]. OP2 was demonstrated to be a versatile tool, supporting a wide variety of run configurations, parallel input and output, and of course a range of parallel hardware architectures. We evaluated the performance and scalability of a testcase simulating a tsunami on the Indian Ocean on up to 32 NVIDIA GPUs, Intel CPU nodes or Intel Xeon Phi machines, and discussed the cost of a simulation in terms of power consumption as well as hardware cost. In recent work with Rolls-Royce plc., we have also demonstrated how their turbomachinery simulation code can be transformed to utilize OP2, which in turn enables execution on modern heterogeneous architectures [18].

3. Structured mesh computations

To widen the scope of our research, we have also been investigating structured mesh computations, supported by the OPS library. The motivations closely follow those described above, and even though the underlying mesh is structured, the variety of computations and target hardware offer plenty of heterogeneity. A key challenge in trying to improve locality and overall performance, is that the optimization of individual computational steps (one loopnest) can only get us so far – being able to analyze and transform several together opens up a whole new range of optimizations. This is an intractable challenge in the general case: static code analysis cannot safely reason about a multi-step algorithm due to possible side effects. We have shown that given the restrictions already imposed by the OPS abstraction, it is possible to delay the evaluation of computations (parallel loops in our case) at runtime, then reason about a number of them together.
One of the most important use cases of transformations applied to multiple computational steps is improving memory locality on CPUs [2]; we demonstrated that based on data dependency analysis, we can carry out cache-blocking split tiling on applications much larger in scale compared to anything where tiling was previously shown. Our research shows that this is not only applicable to improving locality in a single CPU, but across multiple CPUs as well – data movement within and across nodes is reduced. Of course this requires a balancing algorithm to dynamically adapt reuse at runtime - delivering 1.5-3.5x speedups on different applications, and developing a roofline model showing achieved memory bandwidth above that of system DDR memory. We further extended this work to apply to GPUs as well: regarding the GPU's dedicated memory as a large cache, we can solve problems much larger than what would fit in that memory [4].
Our investigation led us to try and utilize heterogeneous computing resources – CPUs and GPUs – to cooperatively solve the same problem. For the kinds of applications that OPS supports, assigning different computational steps (parallel loops) to different hardware is not an option, as steps usually directly depend on each other. Therefore, we split each individual parallel loop and load balance them across the CPU and the GPU according to their computational cost on each hardware. Our paper [5] shows that doing this on a per-loop basis incurs too much data movement cost between the two units and negates any performance benefit. Therefore, we apply the locality improving algorithm discussed above to reduce communications between CPU and GPU. On top of less data movement, it also means fewer

synchronization points allowing our load balancing algorithm to balance across a whole sequence of parallel loops, not just individual ones. We demonstrated that our approach is efficient on different hardware – Intel Haswell generation Xeon CPUs, IBM Power8 CPUs, and NVIDIA's K40 and K80 GPUs, delivering 1.25-1.7x speedup over just utilizing the GPU alone. Unfortunately, given the widening performance gap between CPUs and GPUs, and the trend to pack 4-6 GPUs in a single server, the performance contribution of CPUs to the whole system is decreasing, reducing the potential gain from this technique.

Targeting FPGA architectures poses a new set of challenges in exploiting memory locality as discussed in our paper [17]; data dependency analysis and code transformation have to consider dependencies across individual iterations of different parallel loops. While not fully automated yet, our work on financial and seismic applications built on OPS shows that performance is comparable and better than on the latest NVIDIA GPUs, and 2x better energy efficiency.

Our research also addresses data locality at the other end of the memory hierarchy – creating applications snapshots, or checkpoints, in non-volatile memory. This is motivated by having to improve the resiliency of large-scale applications which are running on thousands of compute nodes – these systems are vulnerable to component failure even within the timeframe of a single run. Our research [10] introduces a fully automated method to save the state space of an application that uses OPS, then restore this state in the event of a failure. We carry out data dependency analysis to determine at what recurring points during execution the state space is the smallest. Our work introduces a number of strategies to save this to non-volatile memory utilizing node-local SSDs as well as the parallel file system, optionally asynchronously to allow the simulation itself to proceed while the checkpoint is being written. We evaluated the scalability of our algorithms on the Titan supercomputer up to 16384 cores.

We demonstrated the utility of the OPS library in delivering performance, productivity, and portability across a wide range of architectures to representative [3] and large-scale academic codes [9]. We demonstrated the simplicity of the code written using OPS, and explored performance on a variety parallel architectures and supercomputers.

4. Extension to Multi-material computations

A key part of my research was to study irregularity at an additional level; in situations where there is a varying number of components or materials at each discretization point. This allows to simulate complex physical systems where multiple materials are getting mixed, and interacting with each other. For multi-material computations and data structures we have explored the existing work in the literature, and designed an abstraction, embedded in the C++ language, that allows the simple description of multi-material computations, without specifying the underlying data structures, or the specifics of execution [7]. The abstraction allows the user to define different types of data on a mesh – material-invariant spatial quantities that have the same number of variables per grid point (such as coordinates), material-dependent spatial quantities, that have the same number of variables per grid point per material, and material-depedent, spatial-invariant quantities, which have the same number of variables per material (such as constant material properties). Operations are defined as an iteration across grid points and materials, with potential reductions in either (or both). Using this abstraction, we have been able to implement three key algorithmic patterns common in multi-material codes: averaging densities of different materials in the same cell,

calculating pressures for each material in each cell, and calculating average density of materials in a local spatial neighbourhood. By utilising our abstraction, there is only one implementation of these algorithms, yet, the underlying data structure can be seamlessly swapped between commonly used variants, such as the “full matrix” and the “compact” storage.

Parallelising multi-material computations is also a challenging task, given the highly irregular nature of computations – there are different numbers, and kinds of materials in different cells. To explore the best approaches to parallelisation, targeting different architectures, programming models, and compilers, I have carried out a detailed study [15], where I studied ARM, IBM, and Intel processors (5 different kinds in total), as well as NVIDIA GPUs (2 types). Performance was evaluated using the Intel, PGI, Clang, GNU, and Cray compilers, utilising OpenMP, CUDA, OpenACC, and SYCL.

As discussed in detail in my second-year report, due to a project collaborator not releasing a large-scale representative multi-material application, I was forced to scale back this aspect of the research. While I did create an abstraction, and studies how to map it to different data structures, parallelizations, and hardware, I did not integrate this into the OP2 library.

5. Extension to financial computations

A new direction for my research was motivated by a collaboration with the Numerical Algorithms Group, whose primary business is consulting for financial companies. Based on the requirements of the financial industry, we carried out research into how the OPS abstraction can be extended to support these types of computations, and map them to a variety of hardware architectures.

One key requirement and challenge was the ability to support the solution of multiple identically sized systems. The traditional programming approach is to solve these systems one by one, or to parallelize across different systems, but we have shown that neither of these are suitable for modern massively parallel architectures [11]. We discuss the extension of the OPS abstraction, and how it can map to different data structures storing these systems, as well as a variety of parallelization approaches, and explored their performance on modern CPU and GPU architectures.

The other requirement for financial computations is being able to automate sensitivity analysis: to compute the adjoint by way of algorithmic differentiations. Our research has minimally extended the OPS abstraction to be able to automatically produce the derivative of computational loops using TAPENADE, and utilizing the delayed execution techniques described, carry out the back-propagation of error through the chain of computations [13]. OPS is now capable of parallelizing these adjoint computations on the CPU using OpenMP, as well as the GPU using CUDA. In total, the overhead of collecting checkpoints and computing adjoints is 3-5x compared to the baseline computation which does neither.

6. Summary

My work has significantly moved the state of the art forward by addressing the challenges of performance, portability, and productivity for scientific computations. Focusing on two problem domains, structured and unstructured mesh computations, and extending them to cover multi-material computations as well as financial calculations, I have demonstrated that it is indeed possible to implement what needs to be computed at a high level, and then

automatically convert it to high performance implementations targeting a variety of parallel architectures. My research studied how to efficiently use the deep memory hierarchies of heterogeneous architectures, and how to map heterogeneous algorithms to these hardware. My results have already been applied to a number of applications used in industry and academia that utilize the OP2 and OPS libraries.

References

[1] Balogh G D, Reguly I Z, Mudalige G R: Comparison of Parallelisation Approaches, Languages, and Compilers for Unstructured Mesh Algorithms on GPUs, In: Jarvis Stephen, Wright Steven, Hammond Simon High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017, Proceedings. New York: Springer International Publishing, 2017. pp. 22-43., 2017

[2] I Z Reguly, G R Mudalige, M Giles: Loop Tiling in Large-Scale Stencil Codes at Run-time with OPS, IEEE T PARALL DISTR 29: (4) pp. 873-886., 2017

[3] R O Kirk, G R Mudalige, I Z Reguly, S A Wright, M J Martineau, S A Jarvis: Achieving Performance Portability for a Heat Conduction Solver Mini-Application on Modern Multi-core Systems, In: 2017 IEEE International Conference on Cluster Computing (CLUSTER) . Honolulu (HI), Amerikai Egyesült Államok, 2017.09.05-2017.09.08. Kiadvány: Piscataway (NJ): IEEE, 2017. pp. 834-841., 2017

[4] Reguly Istan Z, Mudalige Gihan R, Giles Michael B: Beyond 16GB: Out-of-Core Stencil Computations, In: Yonghong Yan (szerk.) (szerk.) Proceedings of the Workshop on Memory Centric Programming for HPC. New York: ACM Press, 2017. pp. 20-29., 2017

[5] Balint Siklosi, Istvan Reguly, Gihan Mudalige: Heterogeneous CPU-GPU Execution of Stencil Applications, In: Douglas, Doerfler; Rob, Neely; Kathryn, O'Brien; John, Pennycook (szerk.) International Workshop on Performance, Portability and Productivity in HPC (P3HPC), (2018) p. 1., 2018

[6] Balogh G. D., Mudalige Gihan R., Reguly I. Z., Antao S. F., Bertolli C: OP2-Clang : a source-to-source translator using Clang/LLVM LibTooling, In: Hal, Finkel (szerk.) The Fifth Workshop on the LLVM Compiler Infrastructure in HPC, (2018) p. 1., 2018

[7] Becker Daniel, Reguly István Zoltán, Mudalige Gihan: An abstraction for local computations on structured meshes and its extension to handling multiple materials, In: Zarándy, Á (szerk.) CNNA 2018.16th International Workshop on Cellular Nanoscale Networks and their Applications, VDE (2018) pp. 1-4., 2018

[8] Reguly Istvan Z., Giles Daniel, Gopinathan Devaraj, Quivy Laure, Beck Joakim H., Giles Michael B., Guillas Serge, Dias Frederic: The VOLNA-OP2 tsunami code (version 1.5), GEOSCIENTIFIC MODEL DEVELOPMENT 11: (11) pp. 4621-4635., 2018

[9] Mudalige G.R., Reguly I.Z., Jammy S.P., Jacobs C.T., Giles M.B., Sandham N.D.: Large-scale performance of a DSL-based multi-block structured-mesh application for Direct Numerical Simulation, JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 131: pp. 130-146., 2019

[10] Reguly I.Z., Mudalige G.R., Giles M.B., Maheswaran S.: Improving resilience of scientific software through a domain-specific approach, JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 128: pp. 99-114., 2019

[11] Reguly Istvan Z., Moore Branden, Schmielau Tim, du Toit Jacques, Mudalige Gihan R: Batch solution of small PDEs with the OPS DSL, Proceedings of 4th International Workshop on

Performance Portable Programming models for Manycore or Accelerators (P^3MA), International Supercomputing Conference, 2019
[12] Sulyok András Attila, Balogh Gábor Dániel, Reguly István Z, Mudalige Gihan R: Locality optimized unstructured mesh algorithms on GPUs, JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 134: pp. 50-64., 2019
[13] D. Balogh G., Z. Reguly I.: Automatic parallel implementations of adjoint codes for structured mesh applications, IEEE, 2020vlc
[14] Reguly István Z., Mudalige Gihan R.: Productivity, performance, and portability for computational fluid dynamics applications, COMPUTERS AND FLUIDS 199: p. 104425., 2020
[15] Reguly I., R. Mudalige G.: Performance Portability of Multi-Material Kernels, Proceedings of the International Workshop on Performance, Portability and Productivity in HPC (P3HPC) held as part of SC19, The International Conference for High Performance Computing, Networking, Storage and , 2019 https://doi.org/10.1109/P3HPC49587.2019.00008
[16] Szilniczky-Eross Botond, Reguly Istvan Z.: Performance Portability of the MG-CFD Mini-App with SYCL, In: IWOCL '20: Proceedings of the International Workshop on OpenCL, (2020) 21, 2020
[17] Kamalakkannan Kamalavasan, Mudalige Gihan R, Reguly Istvan Z, Fahmy Suhaib A.: High-Level FPGA Accelerator Design for Structured-Mesh-Based Explicit Numerical Solvers, Accepted to 35th IEEE International Parallel & Distributed Processing Symposium, May 17-21, 2021, Portland, Oregon USA
[18] Reguly Istvan Z, Mudalige Gihan R.: Modernising an Industrial CFD Application, Proceedings of the Eighth International Symposium on Computing and Networking, 7th International Workshop on Large-scale HPC Application Modernization. Naha, Okinawa, Japan, November 24-27, 2020
[19] Reguly Istvan Z, Powell Archie, Jarvis Steven A, Mudalige Gihan R, Owenson Andrew: Under the Hood of SYCL - An Initial Performance Analysis With an Unstructured-mesh CFD Application. Submitted to: International Supercomputing Conference, Frankfurt, 2021