BookLeaf: An Unstructured Hydrodynamics Mini-application

Workshop paper: WRAp 2018

David Truby*, Steven A. Wright†*, Robert Kevis‡, Satheesh Maheswaran‡, J. A. Herdman‡ and Stephen Jarvis*

*Department of Computer Science, University of Warwick, UK
†Department of Computer Science, University of York, UK
‡Atomic Weapons Establishment, Aldermaston UK

***Abstract*—With the age of Exascale computing causing a diversification away from traditional CPU-based homogeneous clusters, it is becoming increasingly difficult to ensure that computationally complex codes are able to run on these emerging architectures. This is especially important for large physics simulations that are themselves becoming increasingly complex and computationally expensive. One proposed solution to the problem of ensuring these applications can run on the desired architectures is to develop representative mini-applications that are simpler and so can be ported to new frameworks more easily, but which are also representative of the algorithmic and performance characteristics of the original applications.**

In this paper we present BookLeaf, an unstructured Arbitrary Lagrangian-Eulerian mini-application to add to the suite of representative applications developed and maintained by the UK Mini-App Consortium (UK-MAC). First, we outline the reference implementation of our application in Fortran. We then discuss a number of alternative implementations using a variety of parallel programming models and discuss the issues that arise when porting such an application to new architectures. To demonstrate our implementation, we present a study of the performance of BookLeaf on number of platforms using alternative designs, and we document a scaling study showing the behaviour of the application at scale.

I. Introduction

Scientific discovery has benefited enormously as the field of computational science has matured. High Performance Computing (HPC) systems are now an essential tools in scientific investigation, in particular for situations where physical experimentation is prohibitively costly, impractical or dangerous. As the performance of HPC systems has increased, so too has the complexity of the computational problems that can be tackled on them. The next major milestone for supercomputing – ExaFLOP computing (10^{18} Floating-point Operations per Second) – promises to increase this capability further.

In recent years the designs of HPC systems have become more diverse than the traditional homogeneous clusters that were previously prevalent [1]. Emerging architectures, such as many-core CPUs and GPUs, present greater challenges for program implementation than the traditional bulk-synchronous parallel model using multiple MPI processes operating in parallel. This presents particular challenges for large legacy code bases that need to be re-engineered to run on these heterogeneous systems, as their size and complexity makes hardware-specific optimisation difficult and they often cannot be shared with vendors.

One approach being explored by many HPC centres is to develop smaller representative applications that have similar algorithmic and performance characteristics to larger codes but with less complexity and no commercially sensitive content. These *mini-applications* can be shared with vendors and can be ported to emerging architectures without having to handle the complexity or commercial sensitivity of the original applications. This enables scientists to rapidly evaluate new HPC systems, architectures, programming models, algorithms and code optimisation prior to porting or re-engineering large legacy applications.

The UK Mini-App Consortium (UK-MAC) is a collaborative effort by a number of UK institutions to develop a suite of mini-applications that are broadly representative of a number of key multi-physics packages used in science and industry. This paper contributes a 2-D unstructured hydrodynamics mini-application to this benchmark suite. BookLeaf is an unstructured Arbitrary Lagrangian-Eulerian (ALE) code developed in Fortran, that has additionally been ported to the OpenMP and CUDA frameworks. In this paper, we present the algorithm, implementation and an initial performance evaluation of BookLeaf.

Specifically, this paper makes the following contributions:

- We present a new ALE mini-application called BookLeaf and describe the physics underpinning the implementation;
- We outline multiple implementations of this application using OpenMP and CUDA, and highlight the difficulties of porting to these frameworks;
- We demonstrate the performance of the application on various platforms, including Intel Xeon CPUs and NVIDIA GPUs.

The remainder of this paper is structured as follows: Section II provides a summary of similar work; Section III provides an overview of the computational physics involved in the implementation of BookLeaf; Section IV describes the porting of this application to various frameworks; Section V contains a performance analysis of the application on various architectures using these frameworks; and finally, Section VI concludes this paper.

II. Related Work

Architectural and algorithmic co-design is one proposed strategy to tackle the likely challenges of Exascale computation [2]. However, the size and complexity of legacy multi-physics applications means their use in this design space exploration may be prohibitively expensive. Mini-apps, that are broadly representative of key application kernels, are therefore increasingly being developed and used to explore this space much more rapidly. Subsequently, numerous *benchmark suites* have been compiled containing a small number of mini-apps that are of particular interest to a group or laboratory.

One of the earliest examples of this is the NAS parallel benchmark suite, compiled by NASA in 1991 [3]. These "paper-and-pencil" applications have been the focus of a number of performance investigations, including scalability studies [4], implementations in alternative frameworks [5], and porting exercises to new architectures [6].

Lawrence Livermore National Laboratory (LLNL) maintain a small collection of applications as part of their Advanced Simulation and Computing (ASC) Proxy Apps project [7]. The suite includes applications for Algebraic Multi-grid [8], Monte Carlo Particle Transport [9], Scalable I/O [10], [11] and Hydrodynamics [12].

Of particular note, LULESH is a simplified hydrodynamics application similar to BookLeaf, but is restricted to solve only a simple Sedov blast problem on a unstructured 3-D hex mesh [12]. Karlin et al. have performed a thorough performance analysis of LULESH using a large number of different programming frameworks and languages [13].

Similarly, Sandia National Laboratories (SNL) have a suite of mini-apps as part of the Mantevo project [14]. The miniMD application in particular is a good example of the co-design process [15]. It is a feature-limited molecular dynamics application that is representative of the larger LAMMPS production application [16] and has been used in a number of performance studies [17] that have each subsequently influenced the development of LAMMPS.

The UK Mini-App Consortium (UK-MAC) – a collaboration between AWE and UK Universities – also contributes two mini-apps to the Mantevo suite, specifically TeaLeaf and CloverLeaf. TeaLeaf is an application that solves the linear heat conduction equation implicitly using a 5-point stencil. It has been the focus of a number of performance studies, notably by Martineau et al. [18] and Kirk et al. [19].

CloverLeaf is similar to our BookLeaf application, solving the Sod shock tube problem using the ALE method, but using a structured grid [20]. CloverLeaf has subsequently been used to evaluate emerging architectures using various frameworks [21], [22] and has also been extended to use adaptive mesh refinement to improve performance and accuracy of the simulation results [23].

This paper contributes BookLeaf to the UK-MAC benchmarks. BookLeaf is an implementation of an ALE method for shock hydrodynamics, using a quadrilateral 2-D mesh.

III. Method

In this section details of the hydrodynamics discretisation in BookLeaf are given, along with an overview of the test problems provided.

A. Hydrodynamics Scheme

BookLeaf uses an Arbitrary Lagrangian Eulerian method to solve Euler's equations of compressible flow in two spatial dimensions. ALE methods are a hybrid technique that aim to exploit the strengths of both Lagrangian and Eulerian methods while avoiding their weaknesses. As bounding cases in the ALE methodology, BookLeaf has the capability to solve Euler's equations in either the Lagrangian or Eulerian frame, though an additional remap step is required for the latter case.

Euler's equations for compressible flow are a system of three partial differential equations expressing the conservation of mass, momentum and energy.

$$\frac{D\rho}{Dt} = -\rho\nabla\cdot\mathbf{u} \tag{1}$$

$$\rho\frac{D\mathbf{u}}{Dt} = -\nabla P \tag{2}$$

$$\rho\frac{D\epsilon}{Dt} = -P\nabla\cdot\mathbf{u} \tag{3}$$

where $\frac{D}{Dt}$ is the material derivative, ρ is the density, $\mathbf{u}$ is the velocity, P is the pressure and ϵ is the specific internal energy.

To close this system an additional equation called the Equation of State (EoS) is needed. BookLeaf has three options for the EoS – ideal gas, Tait, JWL – plus a void option.

The equations are solved on an unstructured mesh. The mesh comprises of quadrilateral cells, neighbouring cells connect via faces, and faces intersect at nodes. Since the mesh is unstructured, the number of cells surrounding a node is arbitrary. The discretisation used in BookLeaf uses a staggered mesh, whereby thermodynamic variables (e.g. Pressure) are centred in the cell and kinematic variables (e.g. Velocity) are centred on the node. In two dimensions there are six hydrodynamic degrees of freedom whereas a staggered quadrilateral mesh supports eight degrees of freedom. The two non-physical degrees of freedom are called hourglass or zero energy modes. Two of the most common methods for suppressing hourglass modes are filters and sub-zonal pressures. BookLeaf possesses an implementation of a filter following Hancock [24] and sub-zonal pressures following Caramana et al. [25].

Euler's equations for compressible flow are hyperbolic, so an explicit temporal discretisation is appropriate. BookLeaf uses a predictor-corrector method: a first order forward Euler method evolves the state to the half step (predictor), this is then used to time centre the states evolution to the end of step (corrector) to achieve second order accuracy [26]. The spatial discretisation of the Lagrangian step in BookLeaf employs explicitly integrated bilinear isoparametric finite elements. A compatible discretisation (see Barlow [27]) is used to ensure exact energy conservation with second order accuracy. Thermodynamic variables are represented in a piecewise constant manner, and kinematic variables use bilinear elements. Such a spatial discretisation is valid for differentiable compressible flow but inappropriate for shock hydrodynamics. An artificial viscosity term is used in the spatial discretisation in BookLeaf to smear shock discontinuities across a few mesh cells to overcome this restriction. The form of the artificial viscosity term in BookLeaf follows Caramana et al. [28]. The optional remap step in BookLeaf uses a swept volume flux approach [29] which is second order and uses limiters [30] to enforce monotonicity.

The mesh can be spatially decomposed and distributed across processes within BookLeaf using a simple RCB strategy or a hypergraph strategy via METIS [31]. Data that is required from neighbouring processes is stored in ghost layers and retrieved via MPI point-to-point communications. A single global reduction is required by BookLeaf to determine the global timestep for the explicit temporal discretisation.

B. Test problems

BookLeaf comes provided with input for four standard shock hydrodynamic test cases. These are Sod's shock tube, the Noh problem, the Sedov problem and Saltzmann's piston.

Sod's shock tube [32] consists of a shock tube containing two gases initially at rest separated by a diaphragm, when the diaphragm is removed a shock wave is formed and travels from the high pressure gas into the lower pressure gas, a rarefaction wave propagates in the opposite direction. Sod's shock tube tests a codes ability to model the fundamentals of shock hydrodynamics.

Noh's problem [33] consists of a single gas with no internal energy, uniform density and an initially uniform radially inwards velocity field. A strong shock wave is formed at the centre. Noh's problem is used to highlight the wall-heating issue commonly found with artificial viscosity methods.

The Sedov problem [34] is a blast wave emanating from a point source. In BookLeaf this is calculated on a Cartesian mesh to test the codes capability to model non-mesh aligned shocks.

Saltzmann's piston [35] is a simple one dimensional piston problem run on a distorted mesh. This is designed to exacerbate hourglass modes and therefore test a codes capability to suppress such modes.

IV. Implementation

A. Reference

The reference BookLeaf implementation is written in Fortran, using a traditional massage-passing, process-based approach. Inter-process communications are performed using a custom communications library called Typhon, a distributed communications library for unstructured mesh applications. Typhon uses the MPI library as a backend, to handle operations such as halo exchanges and collectives. In BookLeaf, loop exchanges are only performed twice, once immediately before the viscosity calculation and once immediately before calculating the acceleration, meaning that the majority of the main computation loop can execute without pausing for communications. Algorithm 1 outlines the hydrodynamics loop in BookLeaf.

Algorithm 1 Pseudocode for the hydrodynamics loop in BookLeaf

```
procedure HYDRO()
    $dt \leftarrow initial\_dt$
    loop
        if after first time step then
            $dt \leftarrow$ GETDT($dt$)                ▷ Calculate time step
        end if
        LAGSTEP($dt$)                      ▷ Lagrangian calculations
        if grid requires Eulerian remap then
            ALESTEP($dt$)                   ▷ Initiate a remap
        end if
    end loop
end procedure

procedure LAGSTEP($dt$)
    Predictor:
        GETQ()              ▷ Artificial viscosity calculation
        GETFORCE()                         ▷ Calculate forces
        GETGEOM()                          ▷ Update geometry
        GETRHO()                 ▷ Calculate half-step density
        GETEIN()         ▷ Calculate half-step internal energy
        GETPC()                 ▷ Calculate half-step pressure
    Corrector:
        GETQ()
        GETFORCE()
        GETACC()
        GETGEOM()
        GETRHO()                 ▷ Calculate full-step density
        GETEIN()                    ▷ Full-step energy update
        GETPC()                 ▷ Calculate full-step pressure
end procedure

procedure ALESTEP($dt$)
    ALEGETMESH()              ▷ Select mesh to be modified
    ALEGETFVOL()                   ▷ Calculate flux volume
    ALEADVECT()  ▷ Calculate independent advection vars
    ALEUPDATE()       ▷ Update dependent advection vars
end procedure
```

B. OpenMP Host

At scale, some flat MPI applications see degraded performance due to load imbalance. Often this issue can be alleviated using a hybrid model, whereby inter-socket communication is performed using MPI, and intra-socket communication is handled through shared memory. To evaluate the effectiveness of a hybrid model on BookLeaf, an OpenMP implementation is available.

Most kernels in BookLeaf are trivially parallelisable using OpenMP pragmas, with few changes needing to be made to effectively thread the application. However the acceleration calculation kernel currently contains a data dependency that prevents parallelisation. While this potentially could be fixed by rewriting the kernel it has currently been left unchanged, adversely affecting OpenMP performance.

Additionally, the OpenMP implementation makes extensive use of Fortran intrinsics for many calculations, in particular the MINVAL and MINLOC intrinsics. OpenMP nominally provides a method for parallelising intrinsic and elemental functions in Fortran, namely the workshare directive, however the specification for this directive simply states that threads must 'share the work such that each unit is executed only once by one thread'. This specification allows all of the work to be given to a single thread in order to trivially maintain correctness, and this is how this directive is implemented in a number of compilers. As such, the intrinsic functions have been expanded out to normal loops, allowing traditional OpenMP loop directives to be used to force parallelisation.

C. OpenMP Target Offload

Building on the OpenMP host implementation, BookLeaf has been extended to offload to GPUs. This presents new challenges on top of the original implementation. In particular, with an offload implementation care must be taken to ensure the data residency of the arrays is correct and optimal. In the case of BookLeaf, the arrays are transferred to the device at the start of the main loop and are transferred back to the host at the end of the loop. As such, data transfers are only performed once and not every loop iteration. The obvious drawback of this approach is that, for larger problem sets, the entire set of arrays will not fit in GPU memory.

Another particular issue with the OpenMP offload implementation is lack of compiler support. At the time of writing only two compilers support OpenMP offload to GPUs in Fortran – the Cray compiler and the IBM XL compiler. When attempting to support the XL compiler, issues were found with the implementation of reductions, and as such the only supported compiler for OpenMP offload for BookLeaf is currently the Cray compiler. It should be noted that other compilers such as PGI are planning OpenMP offload support for Fortran, however at the time of writing this is not yet complete.

Additionally, since BookLeaf uses a custom communications library on top of MPI which is currently not GPU-aware, it is not possible for the GPU implementations to take advantage of GPU-aware MPI libraries such as OpenMPI. This means that large amounts of redundant data are copied from device to host when run on multiple nodes in order to maintain correct MPI behaviour. As a result the multi-node support in the OpenMP offload and CUDA implementations is currently suboptimal.

D. CUDA Fortran

In order to further evaluate the performance of BookLeaf on GPUs, a CUDA Fortran implementation is also provided. Theoretically such an implementation would allow an evaluation of the overhead involved in writing OpenMP applications.

During our performance evaluation of the CUDA Fortran implementation, an issue was discovered when using assumed-size arrays, which are used extensively in BookLeaf. When an assumed-size array is used as a parameter to a device kernel, the runtime transfers the dope vector associated with that array in order to determine the actual size. While these dope vectors are quite small, usually between 72 and 96 bytes per array, the time taken to transfer these from the host to the device for each kernel run in BookLeaf adds up to a significant time. This can be fixed by specifying the size of each assumed-size array inside the kernels, removing the need for a dope vector transfer. When this optimisation is applied, performance of the kernels improves dramatically; for example, the viscosity kernel runtime is improved from 4.23 seconds to 2.2 seconds for one problem set. Additionally, CUDA Fortran does not provide any reduction primitives, and libraries providing reductions such as CUDA Unbound (CUB) or Thrust that are available for CUDA C are not available for Fortran. While reductions are possible to implement in raw CUDA Fortran, this has not been attempted here. As a result the time differential kernel is run on the host rather than on the device, negatively affecting overall runtime performance.

V. Evaluation

In this section, we evaluate our mini-application on a number of systems using a variety of different parallel programming models. In each case, the results presented are the average runtime of five executions. On all platforms, executions exhibit a statistically insignificant deviation and so error bars are omitted.

A. Experimental Setup

A variety of systems, architectures and compilers were used to collect our results. The Intel Xeon E5-2699 v4 'Broadwell' CPUs and Intel Xeon Platinum 8176 'Skylake' CPU results were collected on a Cray XC50 cluster using the Cray Compiler, which consistently produced the best performing binaries in our testing. In the system used for the Broadwell CPUs there were 22 cores per socket and 2 sockets per node, and for the Skylake CPUs there were 28 cores per socket and 2 sockets per node. The NVIDIA V100 and P100 CUDA Fortran results were obtained with the PGI compiler as this is the only compiler available to us that supports CUDA Fortran, and in each case were obtained on a single GPU connected by PCI Express to an Intel Xeon CPU E5-2660 v4 CPU. OpenMP GPU results were obtained on a single NVIDIA P100 GPU attached to an Intel Xeon E5-2699 CPU in a Cray XC50 cluster using the Cray compiler as this is the only compiler available to us that has implemented OpenMP offload for GPUs. The various platforms and configurations used for these experiments are shown in Table I.

Hardware	System	Compiler	Compiler Flags
Intel Xeon Platinum 8176 'Skylake'	Cray XC50	Cray	`-h cpu=x86-skylake -h network=aries -sreal64 -sinteger -ffree -ra -Oipa3 -O3`
Intel Xeon E5-2699 v4 'Broadwell'	Cray XC50	Cray	`-h cpu=broadwell -h network=aries -sreal64 -sinteger32 -ffree -ra -Oipa3 -O3`
NVIDIA P100 (OpenMP)	Cray XC50	Cray	`-h cpu=broadwell -h accel=nvidia_60 -h network=aries -sreal sinteger32 -ffree -ra -Oipa3 -O3`
NVIDIA P100 (CUDA)	SuperMicro 2028GR-TR	PGI	`-c -r8 -i4 -Mfree -fastsse -O2 -Mipa=fast -Mcuda=cc60`
NVIDIA V100	SuperMicro 2028GR-TR	PGI	`-c -r8 -i4 -Mfree -fastsse -O2 -Mipa=fast -Mcuda=cc70`

TABLE I: Experimental configuration

Hardware	Overall	Viscosity		Acceleration		getdt		getgeom		getforce		getpc	
Skylake MPI	76.068	46.365	*(70%)*	6.663	*(9%)*	8.880	*(12%)*	3.396	*(4%)*	5.364	*(7%)*	1.314	*(2%)*
Skylake Hybrid	168.633	52.913	*(31%)*	15.923	*(9%)*	53.086	*(31%)*	26.654	*(16%)*	4.925	*(3%)*	2.054	*(1%)*
Broadwell MPI	108.978	70.116	*(64%)*	8.386	*(8%)*	11.936	*(11%)*	4.834	*(4%)*	7.348	*(7%)*	1.390	*(1%)*
Broadwell Hybrid	180.438	76.387	*(42%)*	16.142	*(9%)*	45.494	*(25%)*	20.764	*(12%)*	6.501	*(3%)*	2.108	*(1%)*
P100 OpenMP	186.506	75.873	*(41%)*	26.806	*(14%)*	12.684	*(7%)*	16.784	*(9%)*	40.853	*(22%)*	3.608	*(2%)*
P100 CUDA	261.183	97.445	*(37%)*	21.995	*(8%)*	40.433	*(15%)*	39.448	*(15%)*	0.536	*(0%)*	17.922	*(7%)*
V100 CUDA	191.636	44.981	*(23%)*	11.442	*(6%)*	44.401	*(23%)*	14.789	*(8%)*	0.651	*(0%)*	10.051	*(5%)*

TABLE II: Per-kernel performance breakdown in seconds (percentages in parentheses)

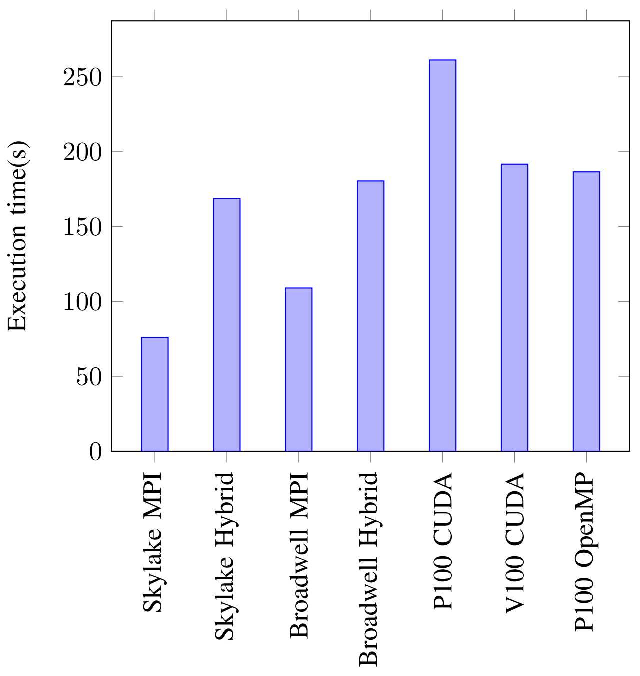


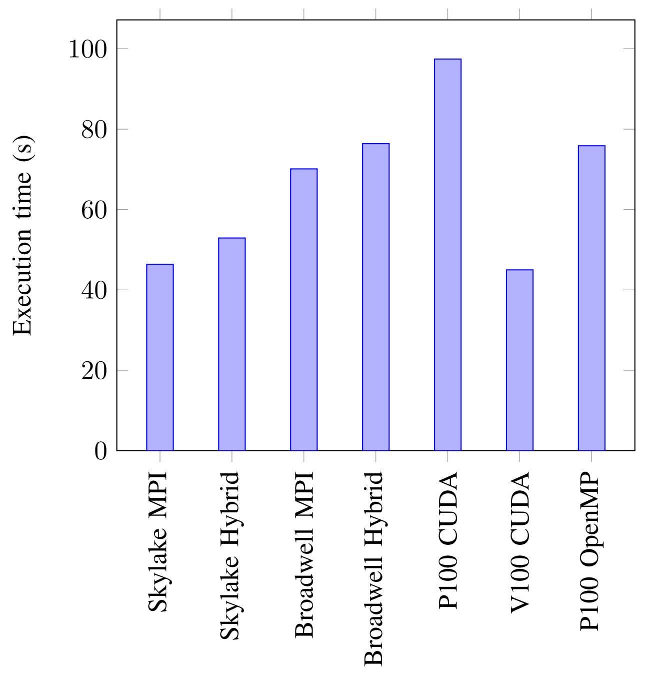
Fig. 1: Overall performance for the Noh problem on a single node.

B. Single Node Performance

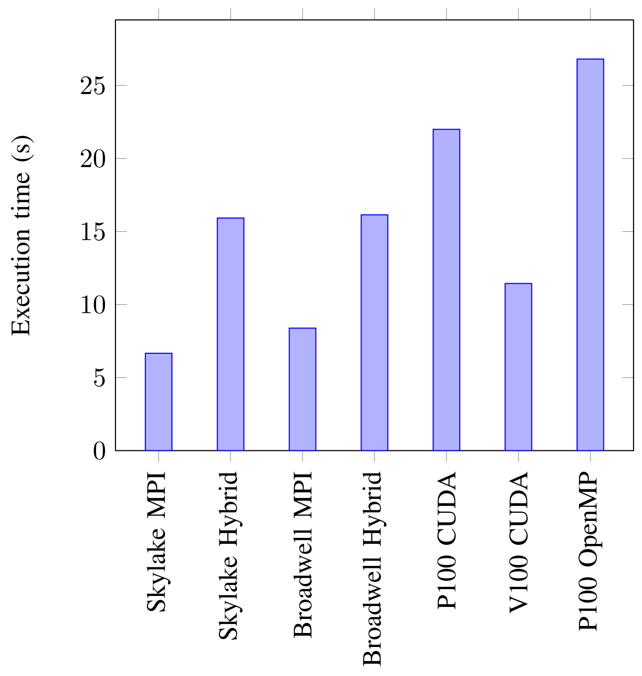
First, we present a performance analysis of BookLeaf on various architectures in single node configurations. CPU results were obtained with two different configurations for comparison, flat MPI with one process per physical core and hybrid MPI+OpenMP with one process per NUMA region. A per kernel performance breakdown of BookLeaf on each platform with the Noh problem set is shown in Table II.

The overall performance for the Noh solver shown in Figure 1 demonstrates a performance discrepancy between flat MPI and hybrid MPI+OpenMP, with the flat MPI model performing better in both cases. However, this performance discrepancy is not so notable in the most theoretically computationally expensive kernel in the application, the viscosity calculation kernel as shown in Figure 2a, where the hybrid solution is within 5% of the performance of the flat MPI solution. The performance difference becomes particularly noticeable in the second most computationally expensive kernel, the acceleration calculation kernel as shown in Figure 2b. Here there is a data dependency in the main calculation loop, severely impacting parallelisation opportunities when using a thread-level framework such as OpenMP, an issue that does not apply to the flat MPI solution.

Additionally the performance on GPUs is shown to be slightly worse overall than that of the CPUs. Part of the reason for the poor performance in the CUDA implementation in particular is that the time differential kernel is performed on the host, meaning the relevant arrays have to be copied from the device to the host for the calculation once per timestep. This is not the case for the OpenMP 4 offload implementation, which can perform the reductions in the kernel correctly, leading to better performance for OpenMP 4 overall than for CUDA. Additionally, the performance for the viscosity calculation is better in the OpenMP offload implementation than in the CUDA implementation. This is due to better register utilisation in the OpenMP offload implementation; this greatly affects GPU performance as registers are shared between threads in the same streaming multiprocessor (SM) so a lower register count allows more threads to be run simultaneously. As such, the CUDA implementation would benefit greatly from further optimisation to reduce the required number of registers in each thread. However one conclusion that can be drawn from this is that although it is often shown that optimised CUDA implementations can outperform OpenMP offload implementations, the work required is much greater



(a) Viscocity calculation kernel



(b) Acceleration calculation kernel

Fig. 2: Per-kernel execution times for the Noh problem on a single node.

and if the CUDA implementation is not carefully optimised then performance can be worse than a simpler pragma-based offload solution.

C. Multi-node Performance

Next, we present a strong scaling study of the BookLeaf Sod solver using the hybrid MPI+OpenMP implementation on a Cray XC50 cluster. The reason for selecting the hybrid implementation for this analysis is that currently the partitioner in BookLeaf is serial, meaning that when trying to scale a large problem up to many hundreds of MPI processes the partitioner begins to dominate the application runtime. Additionally, the serial partitioner requires that the arrays used for partitioning must exist initially in a single process; this results in the root process quickly approaching the maximum amount of memory available on a single node. As mentioned in Section IV-C the GPU offload implementations currently exhibit very poor performance with MPI due to the lack of a GPU aware version of the communications library used by BookLeaf and so have also been omitted from this analysis.

The results shown in Figure 3 show promising performance when scaling BookLeaf. In particular, it can be seen that BookLeaf actually scales superlinearly between 8 and 16 nodes and continues to scale almost linearly when the number of nodes is increased beyond this. The reason for the initial superlinear scaling is the significantly better cache utilisation that can be achieved on each core when the problem set is divided to a certain size. This is particularly significant in BookLeaf since there are very few MPI communications during the main loop and no data-dependent loops except for the acceleration kernel, allowing the better cache utilisation

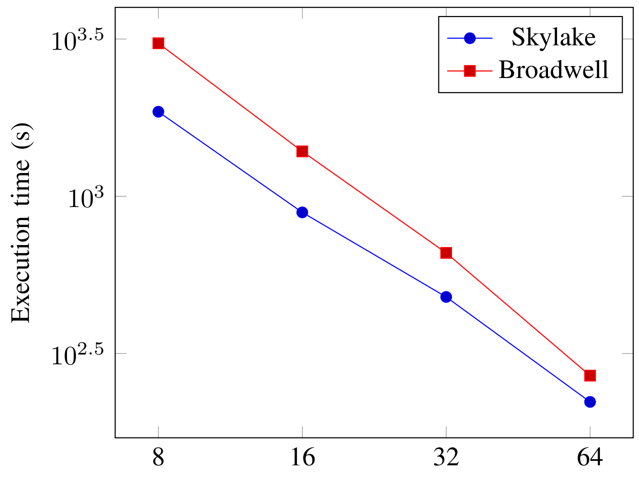
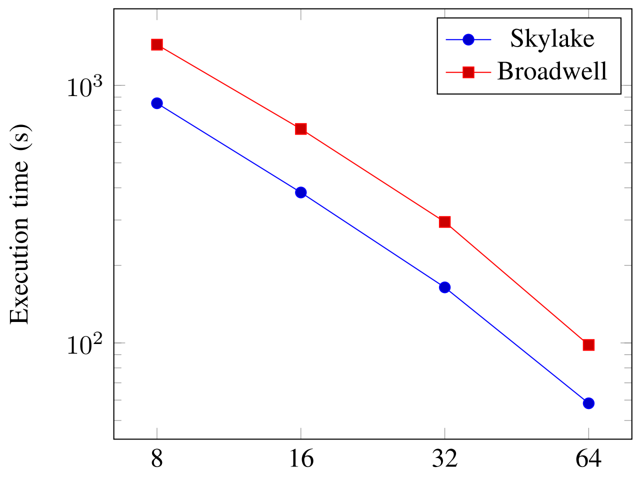


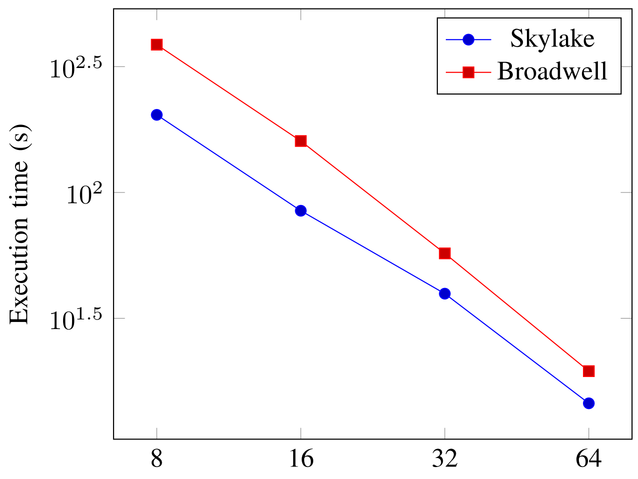
Fig. 3: Overall execution time for the Sod problem when strong scaling.

to be easily visible when scaling. Since there are very few communications calls, the performance continues to scale well even once cache utilisation has become optimal, as can be seen when scaling beyond 16 nodes. It can also be seen here that while the Skylake results are overall better than the Broadwell results, the scaling curve is similar to that of Skylake, showing that the scaling behaviour is portable across CPU architecture generations.

The results for specific kernels shown in Figures 4a and 4b show similar scaling patterns to the overall scaling performance. Again, the kernels scale superlinearly up to 16 nodes



(a) Viscosity calculation kernel



(b) Acceleration calculation kernel

Fig. 4: Per-kernel execution times for the Sod problem when strong scaling.

and then continue to scale almost linearly beyond that. This demonstrates that both kernels are well parallelised and dominate application performance when the application is run at scale. Since both these kernels contain communications this also shows that the communication overhead for these kernels does not cause a significant issue when increasing node counts.

VI. Conclusion

As supercomputing enters the era of Exascale it is increasingly important to ensure applications are capable of running optimally on the platforms taking us to this new level of performance. To allow easier porting to these platforms mini-applications are being developed that are representative of larger applications but are smaller and so easier to port to new platforms for evaluation and benchmarking purposes.

In this paper we present BookLeaf, a new representative mini-application solving a number of shock hydrodynamic problems using an Arbitrary Lagrangian-Eulerian method on a 2-D unstructured quadrilateral mesh. We present the development of this application and present a number of alternative implementations of BookLeaf using the OpenMP framework and CUDA Fortran. Further, we demonstrate our OpenMP implementation in both its traditional use as a CPU parallelisation framework as well as using the newer GPU offload features. Additionally, we outline a number of challenges in porting to these frameworks that other similar codes may also face.

Finally, this paper provides a broad performance analysis of the resulting implementations on a number of platforms, showing the implications of the challenges faced when designing them and demonstrating the performance of BookLeaf itself in a number of different configurations. We also show a strong scaling analysis of BookLeaf on CPU platforms, showing the scaling properties of the application and demonstrating that BookLeaf scales well up to larger numbers of nodes.

A. Future Work

The implementation of Fortran ports of BookLeaf was hampered by lack of support for Fortran in some frameworks and compilers, as well as intricacies surrounding the interaction of Fortran with these frameworks. In order to evaluate whether the same issues would be found when using another language a C++ port of BookLeaf is underway. This will also allow evaluation of CUDA in C++, which has the advantage of additional primitive libraries that are available to C++ CUDA applications. In particular for BookLeaf the reduction primitives provided by the NVIDIA CUDA Unbound (CUB) library allow a proper implementation of the time differential calculation on GPUs.

Furthermore a number of recent parallelisation frameworks are not available for Fortran applications. The C++ port of BookLeaf will also allow the evaluation of the code with a broader range of frameworks. In particular, a C++ GPU implementation using the RAJA performance portability framework from LLNL [36] is planned, allowing further evaluation of GPU results to compare to the implementations described in this paper. These additional implementations will allow a comparison of the state of the art of GPU programming models on both C++ and Fortran.

Acknowledgements

This work was supported by the UK Atomic Weapons Establishment under grant CDK0724 (AWE Technical Outreach Programme). Professor Stephen Jarvis is an AWE William Penney Fellow.

The authors would also like to express our gratitude to Cray for the use of their Marketing Partner Network system used for the performance analysis.

This work would not have been possible without the assistance of a number of members of the Applied Computer Science group at AWE, to whom we would also like to express our gratitude.

Bookleaf can be obtained from GitHub (see https://uk-mac.github.io/BookLeaf/).

References

[1] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "BEOWULF: A Parallel Workstation for Scientific Computation," in *Proceedings of the International Conference on Parallel Processing (ICPP'95)*, 1995, pp. 11–14.

[2] S. S. Dosanjh, R. F. Barrett, D. W. Doerfler, S. D. Hammond, K. S. Hemmert, M. A. Heroux, P. T. Lin, K. T. Pedretti, A. F. Rodrigues, T. G. Trucano, and J. P. Luitjens, "Exascale Design Space Exploration and Co-design," *Future Generation Computer Systems*, vol. 30, pp. 46–58, 2014.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[4] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler, "Architectural Requirements and Scalability of the NAS Parallel Benchmarks," in *Proceedings of the 11th ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'99)*. Portland, OR: IEEE Computer Society, Washington, DC, November 1999, pp. 1–10.

[5] H. Shan, F. Blagojević, S.-J. Min, P. Hargrove, H. Jin, K. Fuerlinger, A. Koniges, and N. J. Wright, "A Programming Model Performance Study Using the NAS Parallel Benchmarks," *Scientific Programming*, vol. 18, pp. 153–167, 2010.

[6] S. J. Pennycook, S. D. Hammond, G. R. Mudalige, S. A. Wright, and S. A. Jarvis, "On the Acceleration of Wavefront Applications using Distributed Many-Core Architectures," *The Computer Journal*, vol. 55, no. 2, pp. 138–153, February 2012.

[7] Lawrence Livermore National Laboratory, "LLNL ASC Proxy Apps," https://codesign.llnl.gov/proxy-apps.php (accessed July 12, 2018), 2018.

[8] P. Samfass, "Porting AMG2013 to Heterogeneous CPU+GPU Nodes," Lawrence Livermore National Laboratory, Livermore, CA, Tech. Rep. LLNL-TR-720025, January 2017.

[9] P. S. Brantley, R. C. Bleile, S. A. Dawson, N. A. Gentile, M. S. McKinley, M. J. O'Brien, M. M. Pozulp, D. F. Richards, J. A. W. David E. Stevens, and H. Childs, "LLNL Monte Carlo Transport Research Efforts for Advanced Computing Architectures," in *Proceedings of the International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, April 2017.

[10] J. Dickson, S. A. Wright, S. Maheswaran, J. A. Herdman, M. C. Miller, and S. A. Jarvis, "Replicating HPC I/O workloads with Proxy Applications," in *1st Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS'16)*. IEEE Computer Society, Los Alamitos, CA, November 2016.

[11] J. Dickson, S. A. Wright, D. Harris, S. Maheswaran, J. A. Herdman, M. C. Miller, and S. A. Jarvis, "Enabling Portable I/O Analysis of Commercially Sensitive HPC Applications Through Workload Replication," in *Proceedings of the 39th Cray User Group (CUG'17)*. Redmond, WA: IEEE Computer Society, Los Alamitos, CA, May 2017.

[12] R. D. Hornung, J. A. Keasler, and M. B. Gokhale, "Hydrodynamics Challenge Problem," Lawrence Livermore National Laboratory, Livermore, CA, Tech. Rep. LLNL-TR-490254, July 2011.

[13] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still, "Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application," in *Proceedings of the 27th IEEE International Parallel & Distributed Processing Symposium (IPDPS'13)*. Boston, MA: IEEE Computer Society, Los Alamitos, CA, May 2013.

[14] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-Applications," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2009-5574, September 2009.

[15] P. S. Crozier and S. Plimpton, "miniMD v. 1.0," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. MINIMDV1.0; 002380MLTPL00, June 2009.

[16] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995.

[17] S. J. Pennycook, C. J. Hughes, M. Smelyanskiy, and S. A. Jarvis, "Exploring simd for molecular dynamics, using intel xeon processors and intel xeon phi coprocessors," in *Proceedings of the 27th IEEE International Symposium on Parallel and Distributed Processing (IPDPS'13)*. Boston, MA: IEEE Computer Society, Los Alamitos, CA, 2013, pp. 1085–1097.

[18] M. Martineau, S. McIntosh-Smith, and W. P. Gaudin, "Assessing the Performance Portability of Modern Parallel Programming Models Using TeaLeaf," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 15, 2017.

[19] R. O. Kirk, G. R. Mudalige, I. Z. Reguly, S. A. Wright, M. J. Martineau, and S. A. Jarvis, "Achieving Performance Portability for a Heat Conduction Solver Mini-Application on Modern Multi-core Systems," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'17)*. Honolulu, HI: IEEE Computer Society, Los Alamitos, CA, September 2017, pp. 834–841.

[20] A. C. Mallinson, D. A. Beckingsale, W. P. Gaudin, J. A. Herdman, J. M. Levesque, and S. A. Jarvis, "CloverLeaf: Preparing Hydrodynamics Codes for Exascale," in *Proceedings of the 35th Cray User Group (CUG'13)*, Napa, CA, May 2013.

[21] S. McIntosh-Smith, M. Martineau, T. Deakin, G. Pawelczak, W. P. Gaudin, P. Garrett, W. Liu, R. Smedley-Stevenson, and D. A. Beckingsale, "TeaLeaf: A Mini-Application to Enable Design-Space Explorations for Iterative Sparse Linear Solvers," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'17)*. Honolulu, HI: IEEE Computer Society, Los Alamitos, CA, September 2017, pp. 842–849.

[22] M. Martineau, S. McIntosh-Smith, and W. P. Gaudin, "Evaluating OpenMP 4.0's Effectiveness as a Heterogeneous Parallel Programming Model," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'16)*. Chicago, IL: IEEE Computer Society, Los Alamitos, CA, May 2016, pp. 338–347.

[23] D. A. Beckingsale, W. P. Gaudin, J. A. Herdman, and S. A. Jarvis, "Resident block-structured adaptive mesh refinement on thousands of graphics processing units," in *Proceedings of the 44th International Conference on Parallel Processing (ICPP'15)*. Beijing, China: IEEE Computer Society, Los Alamitos, CA, September 2015, pp. 61–70.

[24] S. Hancock, "PISCES 2DELK Theoretical Manual," *Physics International*, 1985.

[25] E. J. Caramana and M. J. Shashkov, "Elimination of Artificial Grid Distortion and Hourglass-type Motions by Means of Lagrangian Subzonal Masses and Pressures," *Journal of Computational Physics*, vol. 142, no. 2, pp. 521–561, 1998.

[26] A. Barlow, "An Adaptive Multi-material Arbitrary Lagrangian Eulerian Algorithm for Computational Shock Hydrodynamics," Ph.D. dissertation, University of Wales Swansea, 2002.

[27] A. J. Barlow, "A Compatible Finite Element Multi-material ALE Hydrodynamics Algorithm," *International Journal for Numerical Methods in Fluids*, vol. 56, no. 8, pp. 953–964, 2008.

[28] E. J. Caramana, M. J. Shashkov, and P. P. Whalen, "Formulations of Artificial Viscosity for Multi-dimensional Shock Wave Computations," *Journal of Computational Physics*, vol. 144, no. 1, pp. 70–97, 1998.

[29] D. J. Benson, "An Efficient, Accurate, Simple ALE Method for Nonlinear Finite Element Programs," *Computer Methods in Applied Mechanics and Engineering*, vol. 72, no. 3, pp. 305–350, 1989.

[30] B. Van Leer, "Towards the Ultimate Conservative Difference Scheme. IV. A New Approach to Numerical Convection," *Journal of computational physics*, vol. 23, no. 3, pp. 276–299, 1977.

[31] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[32] G. A. Sod, "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws," *Journal of Computational Physics*, vol. 27, no. 1, pp. 1–31, 1978.

[33] W. F. Noh, "Errors for Calculations of Strong Shocks Using an Artificial Viscosity and an Artificial Heat Flux," *Journal of Computational Physics*, vol. 72, no. 1, pp. 78–120, 1987.

[34] G. I. Taylor, "The Formation of a Blast Wave by a Very Intense Explosion," *Proceedings of the Royal Society of London A*, vol. 201, no. 1065, pp. 159–174, 1950.

[35] J. K. Dukowicz and B. J. Meltz, "Vorticity Errors in Multidimensional Lagrangian Codes," *Journal of Computational Physics*, vol. 99, no. 1, pp. 115–134, 1992.

[36] R. D. Hornung and J. A. Keasler, "The RAJA Performance Portability Layer: Overview and Status," Tech Report, LLNL-TR-661403, 09 2014.