Proceedings Paper:
Owenson, Andrew, Wright, Steven orcid.org/0000-0001-7133-8533, Bunt, Richard et al. (3 more authors) (2018) Developing and Using a Geometric Multigrid, Unstructured Grid Mini-Application to Assess Many-Core Architectures. In: Proceedings - 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2018. 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2018, 21-23 Mar 2018 Institute of Electrical and Electronics Engineers Inc. , GBR , pp. 68-76.

https://doi.org/10.1109/PDP2018.2018.00018

Developing and Using a Geometric Multigrid, Unstructured Grid Mini-Application to Assess Many-Core Architectures

A. M. B. Owenson, S. A. Wright, R. A. Bunt and S. A. Jarvis
Department of Computer Science,
University of Warwick,
Coventry, United Kingdom
Email: a.m.b.owenson@warwick.ac.uk

Y. K. Ho and M. J. Street
Design Systems Engineering,
Rolls-Royce plc,
Derby, United Kingdom

***Abstract*—Achieving high-performance of large scientific codes is a difficult task. This has led to the development of numerous mini-applications that are more tractable to analyse, while retaining performance characteristics of their full-sized counterparts. These "mini-apps" also enable faster hardware evaluation, and for sensitive codes allow evaluation of systems outside of access approval processes.**

In this paper we develop a mini-application of a geometric multigrid, unstructured grid Computational Fluid Dynamics (CFD) code, designed to exhibit similar performance characteristics without sharing code. We detail our experiences developing this application, using guidelines detailed in existing research, and contribute further additions to these to aid future mini-application developers. Our application is validated against the inviscid flux routine of HYDRA, a CFD code developed by Rolls-Royce, which confirms that the parent kernel and mini-application share fundamental causes of parallel inefficiency.

We then use the mini-application to assess the impact of Intel's Knights Landing (KNL) on performance. We find that the mini-app and parent kernel continue to share scaling characteristics, however a comparison with Broadwell performance exposed significant differences between the kernels that were undetected by the validation.

***Keywords*-scientific computing; computational fluid dynamics; performance analysis; high performance computing**

I. Introduction

The rapid development of new hardware and software in High Performance Computing (HPC) is greatly benefiting scientific discovery; with each new development comes new opportunities for improving the performance of scientific applications. Evaluating the potential improvements offered by these developments is often a time consuming process due to the complexity of the applications involved, and the learning curve that often comes with using new machines, architectures and toolchains.

In recognition of these challenges, many HPC centres are turning to alternative tools and methodologies (e.g. predictive performance modelling [1]–[4] and hardware simulation [5], [6]) to evaluate new systems ahead of procurement. Additionally, mini-applications are increasingly being used to facilitate rapid evaluation of new hardware and programming techniques. These applications capture the key performance characteristics of a parent code, in a much more concise form; making them easier to work with but equally useful in performance engineering activities. The use of mini-applications has been well documented in the literature [7]–[10] and has spawned several suites of such applications [11], [12] for industry and the research community to examine.

While the effectiveness of mini-applications has been well demonstrated, the development of them remains a challenging process. In this paper we aim to document our experience developing, validating and using a geometric multigrid, unstructured grid Computational Fluid Dynamics (CFD) mini-application in the context of HYDRA, an application of the same class in use by Rolls-Royce. Specifically this paper makes the following contributions:

- We develop a mini-application of computation over edges which operates on datasets with the following properties: (i) unstructured grid, (ii) geometric multigrid, and (iii) a variable number of neighbours per node;
- We exercise two previously developed mini-application validation techniques on this class of application, to which they have not been applied to before. These techniques provide evidence for similarity between the mini-application and the parent code in terms of their shared memory scalability;
- We assess the utility of the mini-application through an evaluation of the Intel Knights Landing (KNL) architecture. We find that although the mini-application continues to share scaling characteristics with the target kernel, it does not receive the same speedup from MCDRAM due to unequal arithmetic intensity.
- We show that successful application of validation techniques within a single system is not necessarily sufficient to ensure that a mini-application will continue to perform similar to target kernel on different hardware.

This paper is structured as follows: in Section II we discuss related work; in Section III we summarise the functionality of HYDRA which we aim to capture within the mini-application;

in Sections IV and V we describe our experiences constructing mini-HYDRA; in Section VI we validate the performance characteristics of mini-HYDRA when compared to the target kernel; in Section VII we demonstrate the use of mini-HYDRA to assess the impact of the KNL architecture on geometric multigrid, unstructured grid codes; finally, in Section VIII we summarise the work and discuss potential future work.

II. Related Work

There are numerous benchmarks and mini-applications representing the performance of different classes of HPC applications, some of which have been released as part of projects such as the Mantevo Project [11] and the UK Mini-App Consortium [12]. Mini-applications from these repositories and other standalone mini-applications have been used in a variety of contexts. One such example is MiniMD, which has been used to explore the performance of molecular dynamics codes on the Intel Xeon Phi Knights Corner. Using a combination of AVX intrinsics and algorithmic optimisations, such as overlapping PCIe transfers with computation, the authors demonstrate a 5× speed-up for their application.

Mallinson et al. compare the performance of two PGAS programming models (OpenSHMEM and Co-Array Fortran) against MPI using CloverLeaf, a Lagrangian-Eulerian hydrodynamics mini-application [10]. The authors demonstrate that OpenSHMEM is able to outperform an equivalent MPI implementation by 7.78 iterations/sec, at 4096 sockets, when using proprietary nonblocking operations from Cray and 4 MB memory pages.

LULESH, a hydrodynamics mini-application representitive of ALE3D, is used to assess the suitability of emerging parallel programming models (e.g. Liszt and Loci) along with more established models such as OpenMP [13], in terms of programmer productivity, runtime performance and ease of optimisation. The reduced size of LULESH when compared with ALE3D allowed the authors to examine eight parallel programming models. Their conclusion highlights that while the emerging models such as Chapel and Loci enable a high level of productivity, they cannot match the performance of more established models such as MPI and OpenMP.

Similarly, Giles et al. examine the performance of OP2, a domain specific framework for unstructured grid codes using the AIRFOIL CFD mini-application [9]. The authors demonstrate that they are able to achieve programmer productivity and performance within 6% of a hand-coded implementation.

The CFD code included in the Rodinia benchmark suite has been used to examine the performance of a Graphics Processing Unit (GPU) when running unstructured grid applications [14]. From the results, Corrigan et al. conclude that GPUs show promise for this class of code given an increase in double precision performance in the future.

The work in this paper similarly makes use of a mini-application; however, our application additionally contains a geometric multigrid solver and supports mesh structures with variable node degree. Further, we present an additional use case of the mini-application, to examine the impact of the KNL architecture on this class of application. The HPGMG-FV and LULESH mini-applications are most similar to our mini-application however, the former operates on a structured mesh [15] and the latter does not have a multigrid solver.

Another body of work which is similar to our own and that we build upon, deals with the validation of a mini-application's performance characteristics against those of the parent code. The technique employed by Tramm et al. involves comparing the correlation of parallel efficiency loss to performance counters for both the mini-application and the target code [8]. Previously this technique has been applied to mini-applications of a neutron transport code [8]; we apply this technique to a different class of application. Messer et al. develop three mini-applications and use a comparison between the scalability of the mini-application and the original code as evidence of their similarity [16]. However, the authors focus on distributed memory scalability whereas in this work we focus on intra-node shared memory scalability.

Listing 1: Pseudo-code for HYDRA's smooth loop

```
call jacob // Jacobian preconditioning
for iter = 1 to niter do
   for step = 1 to 5 do
      if dissipative flux update then
         call grad      // compute gradient
         call vflux     // accumulate viscous fluxes
         call wfflux    // modify viscous wall fluxes
         call wvflux
      end if

      call iflux      // accumulate inviscid fluxes
      call srcsa      // Spalart-Allmaras source term
      call update     // update flow solution
   end for
end for
```

Listing 2: Pseudo-code for HYDRA's iflux routine, which also describes mini-HYDRA

```
nvar = 5
for e = 1 to nedges do
    a = edges[e].a          // read node indices
    b = edges[e].b

    da = density[a*nvar + 0] // read node data
    db = density[b*nvar + 0]
    ...

    da2 = Fa(da, db, ...) // perform arithmetic
    db2 = Fb(da, db, ...)
    ...

    flux[a*nvar + 0] += da2  // scatter-write out
    flux[b*nvar + 0] += db2
    ...
end for
```

III. Background

A. HYDRA

The manufacturing industry is increasingly making use of CFD simulation codes to aid in the design and testing process of

new products. One such code is HYDRA [17], a suite of nonlinear, linear and adjoint solvers developed by Rolls-Royce in collaboration with several UK universities. HYDRA is used to simulate the flow of fluids in and around some of their commercial aerospace products. For an in-depth discussion of HYDRA, we refer the reader to previous works [18]–[22].

In this paper, we focus on HYDRA's nonlinear solver. Specifically we examine the `vflux` and `iflux` routines in HYDRA's smooth loop (see Listing 1) as these are responsible for the majority of HYDRA's runtime. These routines perform computation on edges; the code loops over edges in the dataset, gathering properties (e.g. momenta and density) from the nodes at either end of the edge, performs some operation (e.g. a flux calculation) on these properties, then "scatter" writes the results out to the two nodes. Pseudocode for `iflux` is provided in Listing 2.

B. Multigrid

HYDRA employs multigrid methods which are designed to increase the rate of convergence for iterative solvers, and posess a useful computational property – the amount of computational work required is linear in the number of unknowns [23]. Multigrid applications operate on a hierarchy of grid levels; in this paper, we are concerned with geometric multigrid, wherein each grid level has its own explicit mesh geometry, and the coarse levels of the hierarchy are derived from the geometry of the finest level.

Starting at the finest level, multigrid applications use an iterative *smoothing* subroutine to reduce high frequency errors. Low frequency errors are then transferred to the next coarsest level (*restriction*), where they appear as high frequency errors and can thus be more rapidly smoothed by the same subroutine. Error corrections from the smoothing of coarse levels are then transferred back to finer levels (*prolongation*). The order in which prolongations and restrictions are applied is know as a cycle, of which this paper considers a single type – the V-cycle.

The potential performance implications of using a geometric multigrid solver are twofold. First there is the increased memory requirement of explicitly representing the geometries of all levels of the multigrid, and second there are the additional irregular memory accesses from prolonging and restricting corrections between levels of the multigrid.

C. Unstructured Grid

HYDRA represents its aerospace models using an unstructured grid – with reference to Fig. 1, an unstructured grid is a collection of nodes (e.g. b and u), edges (e.g. between u and v) and cells (f), with the nodes being at an arbitrary position in space. Since HYDRA operates on multigrid datasets there are also edges between related nodes of adjacent grid levels (e.g. between j and v in Fig. 1). The flexibility of the unstructured grid allows complex geometries to be represented and regions of interest to be denoted by increasing the density of the mesh in these areas.

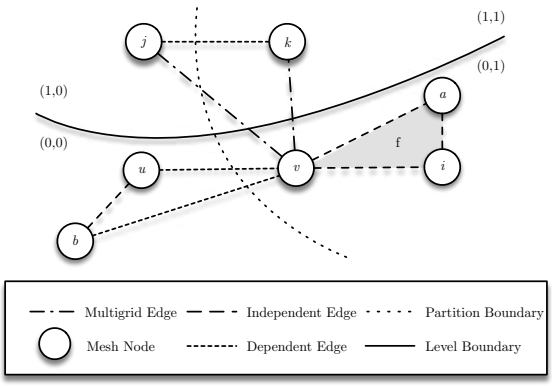

Fig. 1: Abstract representation of an unstructured grid over two multigrid levels. (x,y) notation has meaning (multigrid level, partition number)

The neighbours of a node in an unstructured grid are not implicitly defined, as is the case for a structured mesh code where the neighbours can be determined using offsets to the array indices. This means that an explicit list of neighbours must be maintained so that when computation over nodes is performed (e.g. the accumulation of fluxes) data can be read from the required locations. This of course has implications for the memory access pattern as there is no guarantee that a nodes neighbours directly and regularly succeed it in memory.

D. Experimental Setup

In Section VI, validation of the mini-application against target code behaviour is performed on a dual-socket 28-core Xeon Broadwell node. The Knights Landing evaluation is then performed on a Xeon Phi 7210 configured with flat memory mode and quadrant clustering mode. Full hardware details are provided in Table I.

The unstructured grid used for validation is derived from the geometry of Whittle Laboratory's low pressure axial flow turbine rotor cascade, a mesh of 105 K nodes and 305 K edges representing a single rotor root section (blade and hub connection) [24]. To aid visualisation a rotor section of NASA's SSME 2-stage fuel turbine is shown in Figure 2, consisting of multiple root sections with similar structure to the mesh we use [25]. The mesh is duplicated in memory by a factor of 120, producing a set of 120 disconnected meshes. Each kernel will then process each of the 120 meshes in turn. This ensures that the workload does not fit in the cache, and enables multi-threaded execution at particular process counts such that no two threads work on the same mesh.

IV. Design of mini-HYDRA

Although the benefits of mini-applications are clear (see Section II), their development is not a well-defined process as it depends largely on their intended purpose [16]. This makes their development challenging as the purpose may differ on a project-by-project basis, limiting the reuse of efforts. However,

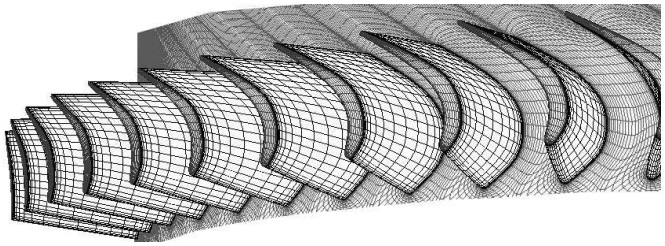

Fig. 2: Visualisation of a rotor section from NASA's SSME 2-stage fuel turbine. Blade geometry is similar to the mesh we use.

	Hardware	
	Broadwell	Knights Landing
Model	E5-2660 v4	Xeon Phi 7210
Turbo Clock (GHz)	2.4	1.3
Cores	14×2	64
Memory (GB)	128	16 HBM + 96 DDR
	Software	
Operating System	Debian 8	Debian 8
Compiler	Intel 18.0.0	Intel 18.0.0

TABLE I: Hardware/software configurations.

the literature details a list of considerations and guidelines; these are aggregated by Messer et al. and summarised here as a set of questions for reference [16].

1) Where does the application spend most of its execution time?
2) What performance characteristics will the mini-application capture?
3) Can any part of the development process be automated?
4) How can the build system be made as simple as possible?

The hope being that when these questions are answered the mini-application's developer will have a concrete understanding as to (i) which aspects of the target code the mini-application should include, and (ii) the components of the supporting configuration (e.g. tools and datasets). We apply these guidelines to the development of mini-HYDRA and because the development of each mini-application is essentially unique, we consider it a valuable exercise to document the usefulness of this approach. Additionally, we add our own considerations to this list, which come from our experiences developing mini-HYDRA.

We address the first question in Section III – the most time consuming regions of code are contained within the routines `vflux` and `iflux`, and it is these routines we should focus on capturing within the mini-application. These kernels have the same computational structure – iteration over edges, and for each edge perform an indirect read of node data (gather), arithmetic computation, then indirect writes back to the two nodes (scatter). In this work we focus on `iflux` as it captures the memory access behaviour of the unstructured grid, while consisting of less code than `vflux`. We would like to note that mini-applications do not always have to represent the most time-consuming aspects of the target code, as the developer may already have a particular characteristic in mind that they wish to study.

The second question is answered by considering the purpose of mini-HYDRA – to evaluate the impact of new hardware features based on their suitability for applications such as HYDRA. This use case suggests constructing a mini-application which ignores I/O and inter-node communication costs and focuses only on computation, encouraging us to focus on more specific regions of the code.

Next, we propose our own consideration: which aspects of the simulation (e.g. unstructured grid, finite volume, multigrid) contribute to the compute behaviour within the most expensive regions of the code? This decomposition by simulation aspect provides us with a route for including performance characteristics within the mini-application. Drawing upon other's experiences with HYDRA along with our own, we know that it is the irregular memory accesses which contribute greatly to the difficulty of running on different compute architectures. These irregular memory accesses come from two main sources: the edge updates over the unstructured grid and the restriction and prolongation of corrections between the multigrid levels (see Section III-B).

V. Implementation of mini-HYDRA

With these features in mind, we base mini-HYDRA on an existing code as (i) it is open source, so mini-HYDRA will not be restricted in terms of where it can be run; and (ii) it shares simulation features with HYDRA [14]. This existing code written in C++ implements a three-dimensional finite-volume discretisation of the Euler equations for inviscid, compressible flow over an unstructured grid. Its flux computation kernel consists of an outer loop over nodes and an inner loop over node neighbours, however `iflux` consists of a loop over edges which is a critical difference and so we modify this kernel to iterate over edges. The resulting kernel then differs to `iflux` only in the arithmetic operations performed; it is not possible for mini-HYDRA to perform the same arithmetic as `iflux` as this would mean subjecting mini-HYDRA to the same portability restrictions as HYDRA itself. We further extend this code with additional simulation features present in HYDRA. It should be noted that we do not focus on verifying the correctness of the simulation against a standard problem, as we are purely interested in performance characteristics which we validate in Section VI.

Support for the computational behaviours of multigrid were implemented by augmenting the construction of the Euler solver presented by Corrigan et al. with crude operators to transfer the state of the simulation between the levels of the multigrid. These operators are defined by Equations 1 and 2 which serve as restriction (fine to coarse grid) and prolongation (coarse to fine grid) operators respectively [26]. Where u_j^l represents simulation property u of node j at level l, and N_j^l is the set of node indices which are linked to node j at level l from $l-1$ of the grid.

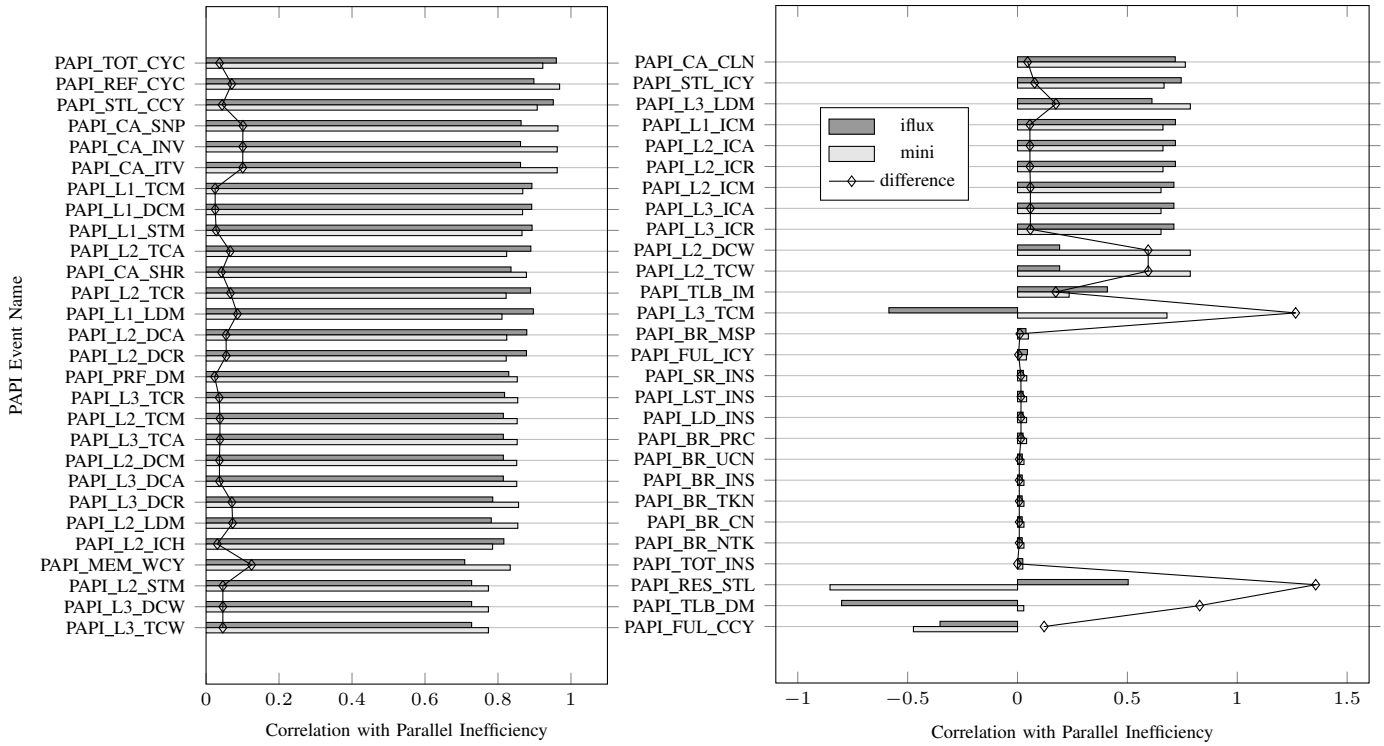

Fig. 3: Comparison between the mini-app and `iflux` of their correlation between PAPI preset performance counters and parallel inefficiency.

$$u_j^l = \frac{\sum_{i \in N_j^l} u_i^{l-1}}{|N_j^l|} \tag{1}$$

$$u_{i \in N_j^l}^{l-1} = u_j^l \tag{2}$$

The restriction operator (Equation 1) primes the simulation properties with an average across nodes from the finer grid level – this mapping between levels is defined as part of the input deck. The prolongation operator (Equation 2) reverses restriction by injecting the values from the coarse grid to the fine grid as dictated by the mapping.

The final code change made is to allow for an arbitrary number of neighbours rather than the fixed four in the flux summation. The summation is already weighted by the surface area of the interface between nodes in the mesh, so no correction to the underlying mathematics is necessary to support this change.

A. Supporting Tools

Part of what makes a mini-application a useful tool is its simplicity, this however is not only restricted to the application itself and must apply to the processes surrounding the mini-application and target application that take time (e.g. building, job submission).

We opt to simplify the building process by removing all reliance on third-party libraries such as the Hierarchical Data Format 5 (HDF5) library and the communications library. These can both be safely removed as the purpose of this mini-application is not to investigate I/O performance, inter-node communication performance nor the overheads introduced by library abstractions. Removing these dependencies allows the application to be built swiftly with minor adjustmend of compiler and its flags in the Makefile. Another obstruction to benchmarking is the need to create job submissions scripts so we include examples of these scripts for several common schedulers: SLURM, LSF and Moab.

Utilities have been included to validate the final state of the simulation after changes to the configuration (e.g. compiler flags, code optimisations, porting to accelerators) of the code. Additionally we include tools to extract the geometries from the datasets used to prime HYDRA and transform them into a form which is understood by the mini-application. We do this to reduce the number of factors which could cause differences in runtime behaviour between HYDRA and its mini-application.

VI. Mini-HYDRA Validation

We validate our mini-application using two existing methods. First we compare the OpenMP parallel efficiency of both `iflux` and mini-HYDRA for all levels of the multigrid [16]. Fig. 4 presents the scaling performance of level 1, showing that both codes exhibit similar strong scaling behaviour (scaling of the other multigrid levels are almost identical to level 1 so are not shown). A strong correlation between scaling behaviour does not imply the underlying causes of

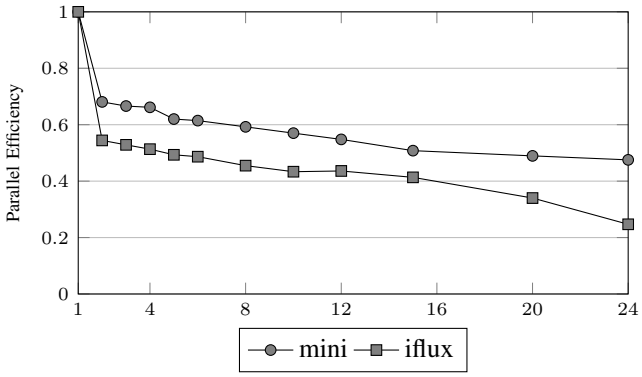

Fig. 4: Parallel efficiency of mini-HYDRA and `iflux`.

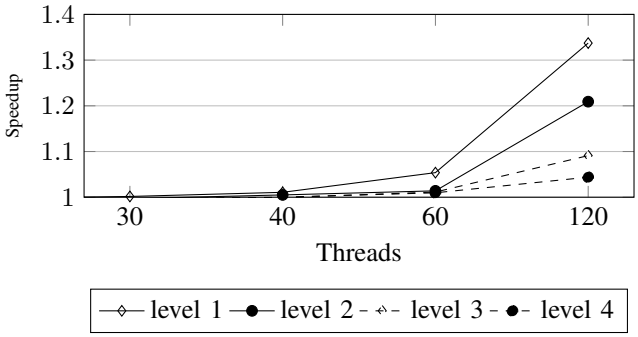

Fig. 5: Speedup provided by KNL's MCDRAM memory to `iflux` over DDR. mini-app received no speedup

the observed behaviour are the same, and so we further strengthen the comparison using a second approach. This involves comparing the correlation of parallel efficiency loss to performance counters for both the mini-application and the target code [8]. Fig. 3 These comparisons highlight differences and similarities between the two applications which we will address in turn. It should be noted that we compare mini-HYDRA against a direct Fortran-to-C port of `iflux`, rather than the original Fortran implementation. We do this to ease and remove the effects of language from the comparison process; arguably this moves us further away from the true performance characteristics of the target code, but it still allows the examination of language independent features, such as memory access patterns and arithmetic intensity.

The PAPI library is used to collect performance counter data, which provides easy access to available performance counters and additionally defines a set of 108 "preset" counters that include performance counters typically found in many processors [27]. Fig. 3 shows the correlation between each PAPI preset performance counter and parallel inefficiency. To account for variance of performance counters between runs the mean of three measurements is used. For most of these events the difference in correlation between the mini-app and `iflux` is less than 0.1, indicating that both codes share many performance characteristics. There are two significant differences, one of which relates to the event PAPI_RES_STL. This event counts the number of cycles where instruction allocation is stalled for any reason, whether that be cache misses or the processor pipeline running at capacity. Because this event is so broad it is not a useful performance indicator, and so the two codes differing here is not considered a problem. The other event on which the codes differ is PAPI_L3_TCM which counts level 3 cache misses. Although the difference is large, both correlations are less than 0.7 absolute which is a weak correlation. This is confirmed by examining this counter, observing that as thread count increases its value changes very little, evidenced by the standard deviation being just 2% of the mean.

Where the correlation between a performance counter and parallel efficiency loss is greater than 0.8, this indicates that the corresponding hardware activity that triggers the counter has a strong influence on scaling performance. Many of these are related to cache read and miss events, and the differences between the correlations of the two codes are small, indicating that performance of both codes are similarly affected by the cache hierarchy performance. Another event affecting parallel efficiency is PAPI_MEM_WCY, which counts cycles stalled due to a lack of available store buffers, indicates a performance bottleneck on memory writes.

VII. The Impact of Intel Knights Landing on the mini-app

In this section we present a short case study of our validated mini-application being used to evaluate an architecture distinct from that which was used for the validation. Specifically we examine what impact the KNL architecture may have on unstructured grid, geometric multigrid codes.

The Intel KNL architecture provides a number of significant changes over its predecessor Knights Corner (KNC) – high-bandwidth MCDRAM memory, change from co-processor to host processor and so direct access to main memory, a 2D mesh interconnect between cores, and the new AVX-512 instruction set including conflict detection (AVX-512CD). These provide peak double-precision compute of 1331 GFLOPs and peak MCDRAM bandwidth of 450 GB/s, which is significantly greater than the Xeon Broadwell node with peaks of 538 GFLOPs and 150 GB/s. However as this analysis will use unvectorised codes, and as these codes do not benefit from fused multiply-add instructions, then the expected peak compute performance is 83.2 GFLOPs on KNL and 67.2 GB/s on the Broadwell node. The reason for using unvectorised codes is that vectorising the scatter-write operations is not arbitrary, and although KNL introduces conflict detection instructions for precisely this purpose it has not been possible to incorporate them into this study. We acknowledge that this will skew the codes to be compute-bound, and we plan to address this in future work.

A. Memory performance

Firstly the benefit of MCDRAM is analysed in Fig. 5. No speedup is seen with the mini-app, however `iflux` does receive a speedup, reaching 1.35× at 120 threads for level 1.

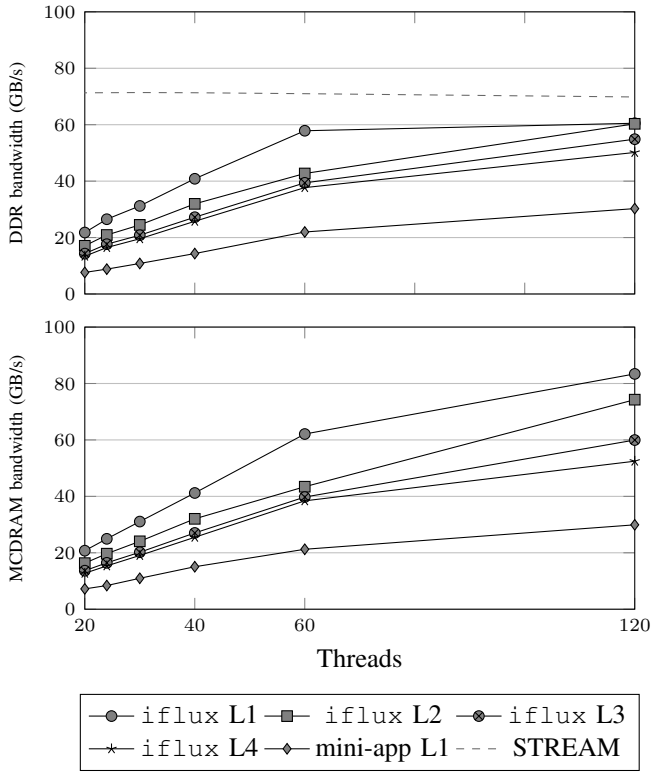

Fig. 6: Memory bandwidth consumption of `iflux`, mini-HYDRA, and STREAM benchmark. `iflux` in DDR appears to become bandwidth-bound beyond 60 threads.

An additional observation is as the multigrid level increases, the speedup received by `iflux` decreases towards 1.0. This variance of benefit can be understood be examining the memory bandwidth consumption of both kernels as shown in Fig. 6. Consumption is measured from native uncore performance counters, and the maximum achievable bandwidth as measured by the STREAM benchmark is shown [28]. At 60 threads and below the bandwidth consumption of both kernels scales linearly, with `iflux` consistently consuming 2.9× more bandwidth than the mini-app. This difference in bandwidth is a result of `iflux` performing 3× less instructions per iteration than the mini-app while reading and writing the same quantity of data.

At 120 threads the mini-app continues to consume more bandwidth, 38% more than at 60 threads for level 1, while `iflux` consumes only 4.6% more. Similarly for multigrid levels 2, 3 and 4 `iflux` achieves a smaller increase in bandwidth than the mini-app. This suggests that `iflux` has become bandwidth-bound despite not reaching the maximum GB/s70 achieved by STREAM, and so will benefit from a move to the MCDRAM. Fig. 6 shows that a move to MCDRAM allows `iflux` to significantly increase bandwidth consumption to 83 GB/s, driving the 1.34× speedup of runtime observed for `iflux` of level 1. At higher multigrid levels the bandwidth consumption is lower, reducing the speedup that MCDRAM can provide. It is not known why meshes

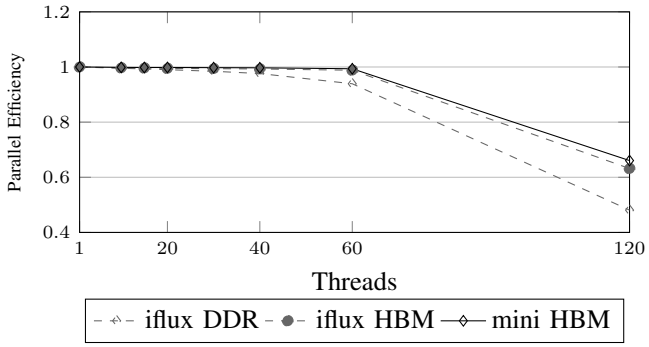

Fig. 7: Parallel efficiency comparison of the mini-app and `iflux` on KNL.

of higher multigrid levels consume less bandwidth, however one reason may be that spatial locality is reduced and so performance becomes increasingly latency-bound from greater cache misses. It is also clear that mini-HYDRA consumes no more MCDRAM bandwidth than DDR and hence receives no speedup. We can conclude from this that more complex HYDRA kernels with similar data access patterns but a higher flop-to-byte ratio will not benefit from the MCDRAM.

Fig. 7 presents the strong scaling on KNL. The movement of `iflux` from DDR into MCDRAM improves its parallel efficiency to match that of the mini-app, achieving linear scaling across the physical cores. However it should be noted that this linear scaling is likely to be a consequence of not fully using the vector units and so being compute-bound. Vectorising these codes with double-precision AVX-512 instructions will increase memory traffic by up to 8×, which for `iflux` at 60 threads could increase bandwidth consumption to 496 GB/s. This would exceed the maximum achievable MCDRAM bandwidth of 432 GB/s as measured by STREAM and so a vectorised `iflux` could become bandwidth-bound in MCDRAM.

B. Compute performance

KNL retains the 4-way Simultaneous Multithreading (SMT) of KNC, but with a change to instruction scheduling as a thread in a KNL core is no longer restricted to executing on every other clock cycle. Therefore there is value in evaluating whether SMT benefits mini-HYDRA. Allocating two threads to each core provides a runtime speedup of approximately 1.3× to both mini-HYDRA and `iflux`. This speedup is consistent across all four multigrid levels, seemingly irrespective of the reduced spatial locality of the higher levels, and so this speedup is likely to be observed with other turbomachinery mesh geometries.

Finally, Fig. 8 shows the speedup of KNL over our Xeon Broadwell node. `iflux` receives an average speedup of 1.44× when running in the DDR memory, which is greater than the 1.24× increase in expected peak compute performance, indicating an increase in work per cycle. However mini-HYDRA receives an average speedup of 1.07×, which translates to a reduction in the work per cycle performed. This is

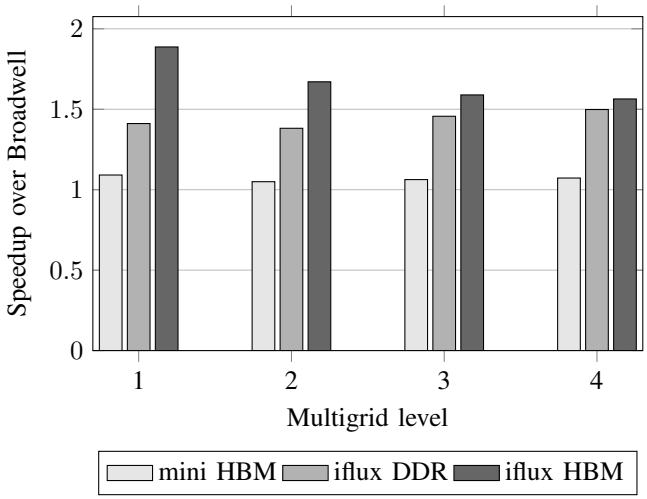

Fig. 8: Speedup of KNL (120 threads) over Xeon Broadwell (24 threads).

an interesting result given that neither kernel is bandwidth-bound on the Broadwell node, and that `iflux` becomes bandwidth-bound on KNL in DDR and so mini-HYDRA is expected to receive the greater speedup. An analysis of performance counters provides the answer, showing that while mini-HYDRA on Broadwell mini-HYDRA performs $2\times$ more instructions than `iflux`, KNL performs $3\times$ more instructions than `iflux` per iteration. This means that when switching from Broadwell to KNL, the arithmetic differences between the two codes cause them to react differently to computational changes in the architectures. This limits use of the current version of mini-HYDRA as a general benchmarking utility in place of HYDRA.

VIII. Conclusions

In this paper, we have extended existing research on mini-application development proposing an additional consideration: which aspects of the simulation contribute to the compute behaviour within the most expensive regions of code. In our experience this consideration aids with the acquisition/development of a mini-application as it maps performance features to implementable code features. Following this, a survey of existing mini-applications was carried out to assess their suitability at representing geometric multigrid, unstructured grid applications. A mini-application which was similar to the target code was extended to include multigrid behaviours so that it could support its input geometries.

Next, we applied two previously developed mini-application validation techniques to a class of code which they have not previously been used on. This validation demonstrated that the mini-application was similar to HYDRA's `iflux` routine. Further analysis highlighted that the scaling behaviour achieved was similar, and that the hardware was being stressed in a similar way by both `iflux` and mini-HYDRA according to hardware counters.

Finally, we demonstrated the use of the mini-application by assessing the impact of the KNL architecture on geometric multigrid, unstructured grid applications, and contrasted this with Xeon Broadwell. Both the mini-app and target code scale linearly across the physical cores, however this is likely a consequence of the codes being unvectorised and so compute-bound. Both codes also benefit equally from executing two hyperthreads per core. We found that the MCDRAM memory can provide a benefit to certain mesh geometries, but this benefit appears to reduce as spatial locality falls. However when KNL performance is contrasted with Broadwell, both codes receive different speedups from the change in architecture which is attributed to differences in the arithmetic operations performed. This shows that successful application of validation techniques within a single system is not necessarily sufficient to ensure that the mini-application will continue to perform similar to the target kernel on different hardware, particularly when the mini-application is not directly derived from the parent code.

A. Further Work

The mini-application covers the memory access patterns of at least 40% of the code base, however the arithmetic intensity of the mini-application is closest to that of one particular parent kernel that accounts for just 11% of its runtime. Another parent kernel with significantly greater compute intensity accounts for a much larger proportion of the runtime, and so we plan to create a mini-app of this kernel.

A limitation of our Knights Landing evaluation is the absence of vectorisation in the codes. To address this we intend to incorporate AVX-512 conflict detection instructions into the mini-app, allowing the evaluation of the vector units and of the memory system under greater load. To address the variation in response to architectural change seen with mini-HYDRA, we have identified a number of arithmetic optimisations which when implemented will cause it to receive a similar speedup to the target kernel of KNL over Broadwell.

The mini-application in its current form does have utility, however the inability to execute it at any thread count or to use vector units on any architecture does impose limits. To address this we plan to incorporate the OP2 framework into the mini-app, which will enable vectorisation, safe decomposition across any number of threads and multi-process parallel execution. This will also allow for execution on large single meshes that are more representiative of modern workloads, such as NASA's Rotor37 [29].

IX. Acknowledgment

This research is supported by Rolls-Royce plc. through the Clean Sky project, and by the Engineering and Physical Sciences Research Council (EPSRC) and Intel Corporation (CASE award 15220082). The authors would like to thank Rolls-Royce plc. for the provided support and for granting permission to publish this work.

REFERENCES

[1] G. R. Mudalige, M. K. Vernon, and S. A. Jarvis, "A Plug-And-Play Model for Evaluating Wavefront Computations on Parallel Architectures," in *Proceedings of the 22nd International Parallel and Distributed Processing Symposium 2008 (IPDPS'08)*. Miami, Florida: IEEE Computer Society, Los Alamitos, CA, April 2008, pp. 1–14.

[2] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. Sancho, "Using Performance Modeling to Design Large-Scale Systems," *Computer*, vol. 42, no. 10, pp. 0042–49, November 2009.

[3] R. A. Bunt, S. J. Pennycook, S. A. Jarvis, L. Lapworth, and Y. K. Ho, "Model-Led Optimisation of a Geometric Multigrid Application," in *Proceedings of the 15th High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing 2013 (HPCC&EUC'13)*. Zhang Jia Jie, China: IEEE Computer Society, Los Alamitos, CA, November 2013, pp. 742–753.

[4] R. A. Bunt, S. A. Wright, S. A. Jarvis, M. Street, and Y. K. Ho, "Predictive Evaluation of Partitioning Algorithms Through Runtime Modelling," *Proceedings of The 23rd IEEE International Conference on High Performance Computing, Data, and Analytics*, pp. 351–361, 2016.

[5] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, A. J. Herdman, and A. Vadgama, "WARPP: A Toolkit for Simulating High-Performance Parallel Scientific Codes," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques 2009 (ICSTT'09)*. Rome, Italy: ICST, Gent, Belgium, March 2009, pp. 1–10.

[6] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A Simulator for Large-Scale Parallel Computer Architectures," *International Journal of Distributed Systems and Technologies*, vol. 1, no. 2, pp. 57–73, April 2010.

[7] 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 International Parallel and Distributed Processing Symposium 2013 (IPDPS'13)*. Boston, MA: IEEE Computer Society, Los Alamitos, CA, May 2013, pp. 1085–1097.

[8] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench-the Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis," in *Proceedings of the Role of Reactor Physics Toward a Sustainable Future 2014 (PHYSOR'14)*. Kyoto, Japan: Taylor and Francis, Oxford, UK, September 2014, pp. 1–12.

[9] I. Z. Reguly, G. R. Mudalige, and M. B. Giles, "Design and Development of Domain Specific Active Libraries With Proxy Applications," in *Proceedings of Cluster Computing 2015 (CLUSTER'15)*. Chicago, IL: IEEE Computer Society, Los Alamitos, CA, September 2015, pp. 738–745.

[10] A. C. Mallinson, S. A. Jarvis, W. P. Gaudin, and A. J. Herdman, "Experiences at Scale With PGAS Versions of a Hydrodynamics Application," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models 2014 (PGAS'14)*. Eugene, Oregon: ACM, New York, NY, October 2014, pp. 9–20.

[11] M. Heroux and R. Barrett, "Mantevo Project," https://mantevo.org/ (accessed March 3, 2016), March 2016.

[12] UK Mini-App Consortium, "UK Mini-App Consortium," http://uk-mac.github.io/papers.html (accessed March 6, 2016), 2016.

[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 and Distributed Processing Symposium 2013 (IPDPS'13)*. Boston, MA: IEEE Computer Society, Los Alamitos, May 2013, pp. 919–932.

[14] A. Corrigan, F. F. Camelli, R. Lhner, and J. Wallin, "Running Unstructured Grid-Based CFD Solvers on Modern Graphics Hardware," *International Journal for Numerical Methods in Fluids*, vol. 66, no. 2, pp. 221–229, May 2011.

[15] M. F. Adams, J. Brown, J. Shalf, B. Van Straalen, E. Strohmaier, and S. Williams, "HPGMG 1.0: A Benchmark for Ranking High Performance Computing Systems," Lawrence Berkeley National Laboratory, Tech. Rep., 2014.

[16] O. E. B. Messer, E. D'Azevedo, J. Hill, W. Joubert, S. Laosooksathit, and A. Tharrington, "Developing MiniApps on Modern Platforms Using Multiple Programming Models," in *Proceedings of Cluster Computing 2015 (CLUSTER'15)*. Chicago, IL: IEEE Computer Society, Los Alamitos, CA, September 2015, pp. 753–759.

[17] L. Lapworth, "HYDRA-CFD: A Framework for Collaborative CFD Development," in *Proceedings of the International Conference on Scientific and Engineering Computation 2004 (IC-SEC'04)*, Singapore, June 2004.

[18] M. S. Campobasso and M. B. Giles, "Stabilization of a Linearized Navier-Stokes Solver for Turbomachinery Aeroelasticity," in *Proceedings of the 2nd International Conference on Computational Fluid Dynamics 2002 (ICCFD'02)*. Sydney, Australia: Springer, Berlin, Germany, July 2002, pp. 343–348.

[19] ——, "Effects of Flow Instabilities on the Linear Analysis of Turbomachinery Aeroelasticity," *Journal of Propulsion and Power*, vol. 19, no. 2, pp. 250–259, March 2003.

[20] P. Moinier, J. Mller, and M. B. Giles, "Edge-Based Multigrid and Preconditioning for Hybrid Grids," *AIAA Journal*, vol. 40, no. 10, pp. 1945–1953, October 2002.

[21] M. C. Duta, M. B. Giles, and M. S. Campobasso, "The Harmonic Adjoint Approach to Unsteady Turbomachinery Design," *International Journal for Numerical Methods in Fluids*, vol. 40, no. 3–4, pp. 323–332, September 2002.

[22] M. B. Giles, M. C. Duta, J. Mller, and N. A. Pierce, "Algorithm Developments for Discrete Adjoint Methods," *AIAA Journal*, vol. 41, no. 2, pp. 198–205, February 2003.

[23] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*. Elsevier, Amsterdam, The Netherlands, 2001.

[24] H. P. Hodson and R. G. Dominy, "Three-Dimensional Flow in a Low-Pressure Turbine Cascade at Its Design Condition," *Journal of Turbomachinery*, vol. 109, no. 2, pp. 177–185, April 1987.

[25] https://www.grc.nasa.gov/www/5810/rvc/tcgrid.htm (accessed November, 8th 2017).

[26] W. L. Briggs, *Multigrid Tutorial*. SIAM, Philadelphia, PA, 1987.

[27] S. Browne, C. Deane, G. Ho, and P. Mucci, "Papi: A portable interface to hardware performance counters," in *Proceedings of Department of Defense HPCMP Users Group Conference*, June 1999.

[28] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, December 1995.

[29] L. Reid and R. D. Moore, "Design and Overall Performance of Four Highly Loaded, High Speed Inlet Stages for an Advanced High-Pressure-Ratio Core Compressor," NASA Lewis Research Center, Cleveland, OH, Tech. Rep. NASA TP 1337, October 1987.