Optimisation of Patch Distribution Strategies for AMR Applications

D. A. Beckingsale, O. F. J. Perks, W. P. Gaudin, J. A. Herdman, and S. A. Jarvis

Performance Computing and Visualisation,
Department of Computer Science, University of Warwick, UK
dab@dcs.warwick.ac.uk

Abstract. As core counts increase in the world's most powerful supercomputers, applications are becoming limited not only by computational power, but also by data availability. In the race to exascale, efficient and effective communication policies are key to achieving optimal application performance. Applications using adaptive mesh refinement (AMR) trade off communication for computational load balancing, to enable the focused computation of specific areas of interest. This class of application is particularly susceptible to the communication performance of the underlying architectures, and are inherently difficult to scale efficiently. In this paper we present a study of the effect of patch distribution strategies on the scalability of an AMR code. We demonstrate the significance of patch placement on communication overheads, and by balancing the computation and communication costs of patches, we develop a scheme to optimise performance of a specific, industry-strength, benchmark application.

1 Introduction

In the race to exascale, floating point operations are becoming cheaper and the real challenge is providing data to utilise the available computational power. Communications are therefore becoming more important in scientific computing and efficiently transferring data will be key to scaling the existing generation of petascale applications.

Adaptive mesh refinement (AMR) is a technique used to increase the resolution of computation in areas of interest—such as shock fronts and material interfaces—avoiding the necessity of a uniform fine-grained mesh [1, 2]. The technique uses multiple levels of refinement, where areas of interest are identified and the accuracy of computation is increased, by subdivision of the problem domain. This decomposition and refinement creates a natural computational load imbalance as work will be clustered around the areas of interest. The technique uses a distribution strategy to share the increased workload between under-utilised compute resources. The basic unit of distribution is a *patch*, a rectangular subgrid of cells which result from mesh refinement. The decrease in computation time offered by AMR comes at the cost of increased communication overheads,

caused by additional boundary communications, and data transfer between refinement levels. The management of patches and associated AMR metadata creates an additional computational overhead, however, it is not the primary focus of this paper.

In this paper we present a cost-benefit analysis of patch distribution strategies and identify optimisation opportunities. We present our study in the context of Shamrock, a 2-dimensional, Lagrangian hydrodynamics code utilising AMR, developed at the UK Atomic Weapons Establishment (AWE). Shamrock is an industry-strength benchmark supporting a range of architectures. It is a key tool in evaluating future high-performance computing technologies at AWE, and provides a robust software framework for our investigation [3].

Specifically, we study how allowing workload imbalance can, under certain conditions, reduce the communication overheads and thus, by identifying situations where patch distribution has a negative effect on overall runtime, improve the scalability of the code. Initially, we demonstrate the application of AMR on a symmetric, and therefore, naturally load balanced problem. In this case, any distribution of patches will increase communication time and harm overall performance. We extend this simple example to motivate the use of patch distribution on inherently more representative asymmetric problems, where load balance is not guaranteed. For these asymmetric problems we demonstrate the available trade-off between communication and computation costs, and highlight the potential advantages of an optimal distribution strategy.

The specific contributions of this work are as follows:

- A cost-benefit analysis of AMR patch distribution on symmetric and asymmetric input decks is documented.
- We implement an AMR-level distribution threshold to mitigate cost between fully enabled and fully disabled patch distribution strategies.
- We demonstrate the potential of an intelligent, environment-driven, patch distribution strategy, through hand tuning, motivating the case for a runtime-based heuristic distribution strategy.

The remainder of this paper is structured as follows: in Section 2, we present an overview of related work. Section 3 presents a more detailed discussion of the key aspects of AMR. In Section 4 we analyse the load imbalance caused by two different problem input decks. In Section 5 we analyse the scalability of two patch distribution schemes, and Section 6 presents an analysis of the application of a distribution threshold-based on the AMR level. Section 7 then demonstrates how, with prior knowledge of expected communication and computation costs, we can improve the runtime and scalability of the application. Finally, in Section 8 we conclude the paper and discuss future work.

2 Related Work

The structured AMR techniques developed by Berger and Oliger [1] have been successfully applied to a number of problem domains including cosmology [4, 5],

astrophysics [6], and shock hydrodynamics [7, 8]. The power of AMR in increasing solution accuracy without the requirement of a uniform fine-grained mesh has motivated investigation into ensuring the scalability of these techniques.

Codes utilising AMR typically contain a number of distinct steps that can have an impact on scalability: adding refined patches to flagged areas of interest, balancing the load of patches across processors, and communication and synchronisation between patches during calculations.

Adding refined patches, or re-gridding, is a computationally intensive process involving identifying areas of interest, flagging the cells in these areas, and creating a set of refined patches to cover all the flagged cells. Luitjens and Berzins present a study of three common re-gridding techniques, remarking on the communication and computation complexity of the three algorithms, and demonstrating scalable results from the Berger-Rigoustous algorithm [9, 10].

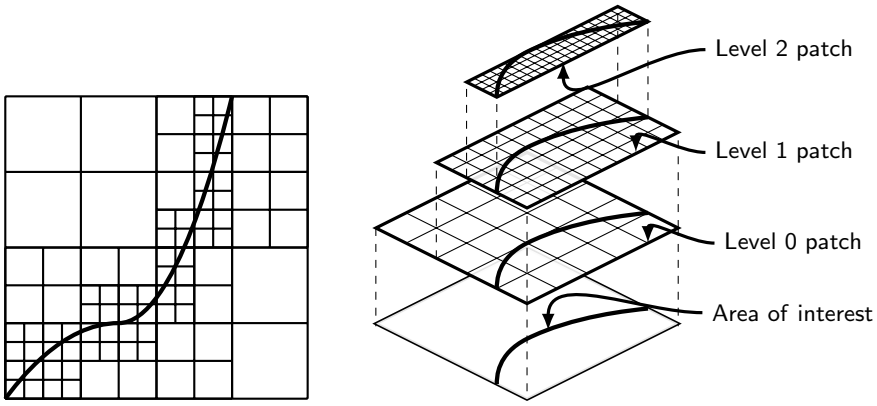
Optimal distribution of refined patches is key to achieving efficient execution and acceptable runtimes. Lan *et al.* present reductions in runtime of up to 47% when using a grid-splitting technique to move work from overloaded processors to underloaded ones [11]. Using system measurements provides an effective way to ensure work is evenly balanced [12], and the optimisations presented in Section 7 make use of a set of simple measures of system performance to develop an efficient patch distribution scheme. Our work differs from previous research by building on work in performance modelling [13, 14], using measured parameters to estimate the optimal distribution of patches.

Identifying the factors limiting the scalability of AMR is an ongoing problem, however, the key features identified by Colella *et al.* provide a platform for investigation: minimising communications, efficiently computing patch metadata, and optimising communication between coarse and fine patch boundaries [15]. Van Straalen *et al.* extend these features, concluding that the traditional concerns over load imbalance and communication volume were not as critical to application performance as identifying and isolating subtle use of non-scalable algorithms in the grid management infrastructure of the AMR framework [16].

3 Adaptive Mesh Refinement

Adaptive mesh refinement is the process of increasing the resolution of computation at specific areas of interest, allowing increased accuracy where it is most needed. Figure 1(a) illustrates how AMR would be applied to a simple area of interest. An area of interest is a portion of the problem where there is a high degree of entropy, such as at material interfaces or at shock fronts.

The AMR capability of Shamrock is provided by custom AMR library, rather than third-party AMR package such as SAMRAI [17] or Chombo [18]. Areas of interest are identified and the cells containing them are flagged. These flagged cells are then grouped into rectangular patches, which may contain some unflagged cells. Computation is carried out over all patches, and hence, all levels. Solutions are transferred between the patches on different levels, with coarse solutions being mapped up to higher levels, and the more accurate solutions



(a) A complete AMR mesh for a simple problem.

(b) Multiple levels of refined patches.

Fig. 1: AMR mesh and corresponding application of patches.

being projected back down. A typical arrangement of patches is illustrated in Figure 1(b).

The complexity of AMR, and the associated metadata required, present some problems for a parallel implementation: (i) newly created patches need to be assigned to a processor, (ii) boundaries must be communicated between patches, sometimes across AMR levels, and (iii) solutions must be mapped and projected between patches.

Once areas of the problem have been refined, work required in that area, and hence CPU time spent computing that area, will increase. In order to combat this it is common to distribute the extra work to processors with a smaller workload. Refinement may happen in only one small area of the problem, and if the newly created patches are not distributed in an effective way then the resulting load imbalance is likely to negatively affect the amount of time spent in computation. However, distributing the patches will increase the cost of both the boundary communications and the transfer of solutions between patches.

Boundary communications occur at patch interfaces and involve the swapping of variables in ghost cells between adjacent patches. However, boundary communications may also have to cross refinement levels, since a patch may be adjacent to another patch on a different level. Boundary communications across refinement levels involve some additional computational overhead as solution values must be interpolated onto the grid due to the differing resolution. Solution transfer occurs where patches on different refinement levels overlap, and involves the transfer of the necessary variables, between levels, for each overlapping cell. The data needs to be transferred every timestep, so high transfer costs affect the scalability of the code.

The competing factors of communication and computation need to be effectively managed in order to minimise runtime. At small scale the addition of

	Tricca	Hera
Processor	Intel E3-1240 3.3 GHz	AMD 8356 2.3 GHz
Compute Nodes	1	864
Cores/Node	4	16
Total Cores	4	13,824
Memory/Node (GB)	12	32
Interconnect	N/A	Infiniband DDR
Compiler	Intel 11.1	Intel 12.1
MPI	OpenMPI 1.4.3	OpenMPI 1.4.3

Table 1: Summary of experimental platforms.

more processors will reduce overall runtime, as the added communication overheads are insignificant compared to the reduction in computation time. At larger scale the ratio of communication to computation is no longer favourable, and the added communication cost drives up overall runtime. By finding effective ways to balance the computation-communication trade off, we can increase the performance and scalability of the code.

4 Symmetric and Asymmetric AMR

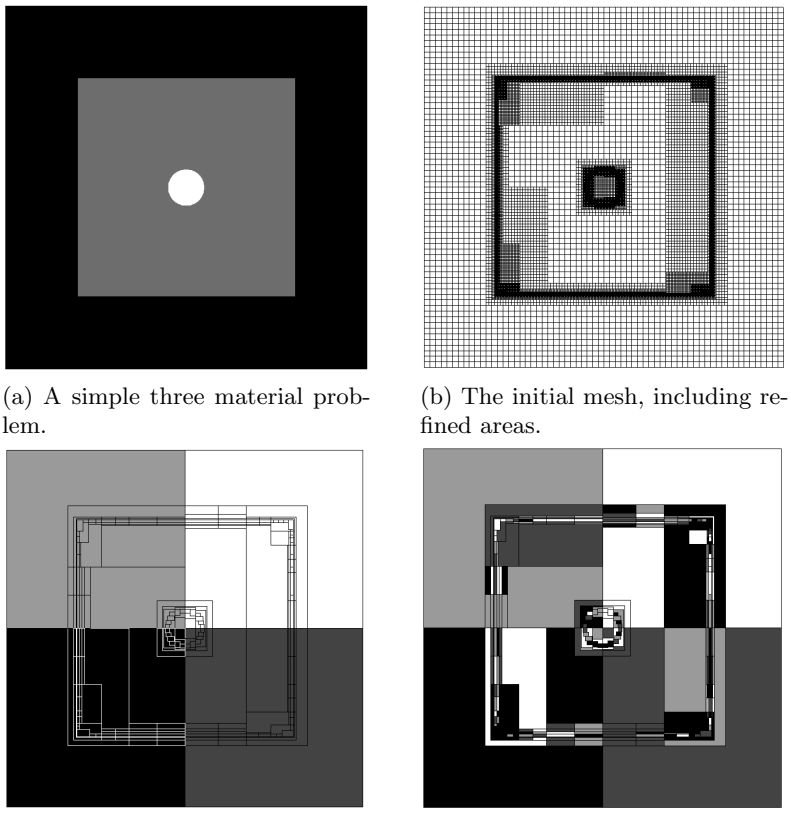
As discussed in Section 3, a key component of AMR is patch distribution. This enables local fine grained analysis in areas of interest, whilst maintaining an even workload for all processors.

With the existing patch distribution strategy, patches are always distributed, regardless of locality, with a probability based on an estimation of current computational workload. The implication of this strategy is that previously neighbouring patches, in both the horizontal and vertical domain, can be physically distributed across the whole machine, increasing communication times. The nature of Shamrock means that the majority of communication is the exchange of boundary cells and transfer of solutions, and although some global communication is required, physical proximity is crucial for keeping communication costs at a minimum, and in turn improving scalability.

Throughout this paper we make use of results obtained from two different computing platforms, illustrated in Table 1. The first is a quad-core workstation, used to demonstrate behaviour at small scale, the second is a large 127 TFLOP/s supercomputer, located at the Lawrence Livermore National Laboratories (LLNL) Open Compute Facility, used to demonstrate the scalability of our techniques.

4.1 Symmetric AMR

We demonstrate a symmetric decomposition using a square multi-material problem, executed on four processors. Although the resulting mesh is not symmet-



(a) A simple three material problem.

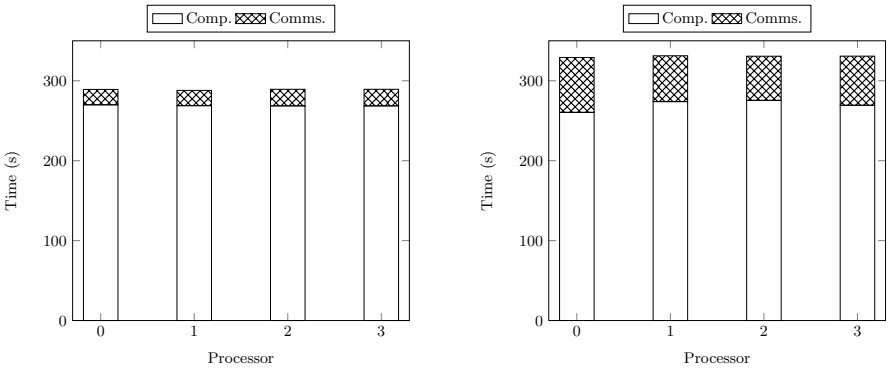
(b) The initial mesh, including refined areas.

(c) Patch assignment without distribution.

(d) Patch assignment with distribution.

Fig. 2: A symmetric multi-material problem, with the corresponding AMR mesh and patch assignments.

rical, the processors receive the same number of refined cells, giving an approximately equal and naturally balanced decomposition. With the original distribution policy, patches will be distributed to different processors, reducing the physical proximity of neighbouring patches. Whilst the goal is to reduce load imbalance, this distribution strategy actually introduces a small imbalance due to the order of patch distribution. The added communication cost of this distribution also has a significant impact on runtime. We also note that despite the obvious symmetry of the problem mesh, the refinement may be asymmetrical due to the nature of the underlying algorithm. Whilst this may not be a representative AMR computation, it demonstrates a situation where patch distribution will perform poorly. From this example we can then infer potential performance gains from alternative patch distribution strategies in more representative problems.



(a) Non distributed symmetric mesh. (b) Distributed symmetric mesh.

Fig. 3: Runtime breakdown for a symmetric decomposition on four cores.

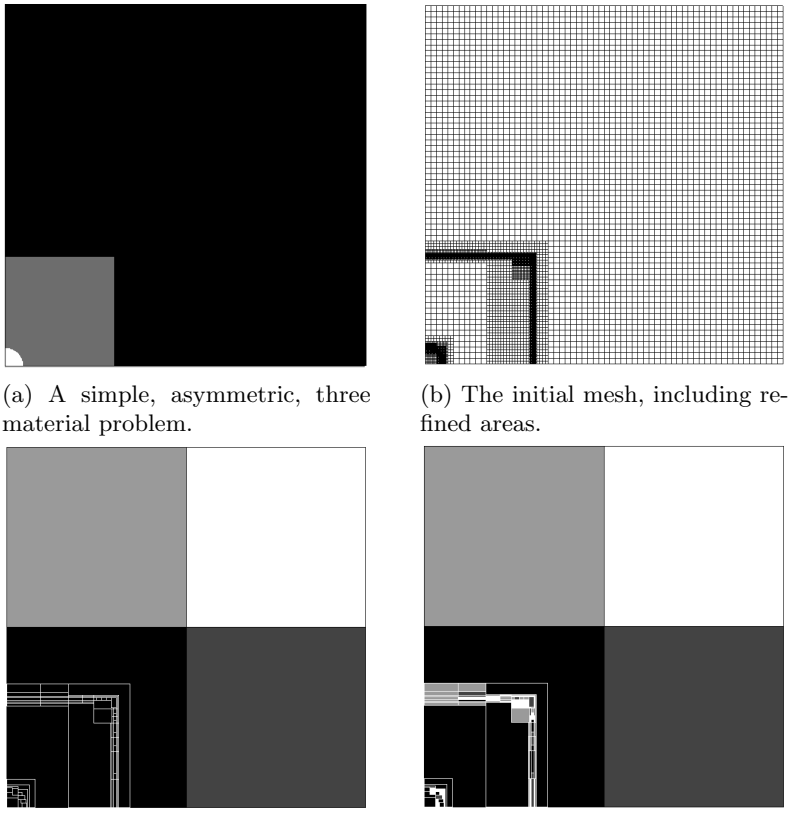
Figure 2(a) presents the problem, with its area of interest, coupled with the resulting mesh, Figure 2(b), from an AMR level of four. We illustrate the patch distribution when decomposed over four processors firstly from a no distribution policy, Figure 2(c), and secondly a round-robin distribution, Figure 2(d).

The amount of time spent in computation time and communication time is presented in Figure 3. Both decompositions are load balanced, with each processor spending an average of 268.98s in the computation portion of the application. However, the distribution of the patches in Figure 2(d) has destroyed the spatial locality of the patches and increased the communication times to an average of 60.67s. By keeping patches local the communication time is reduced to an average of 19.96s, which provides a speedup of 1.14×. These differences are caused wholly by on-node patch distribution, where communication times are significantly lower than off-node.

4.2 Asymmetric AMR

In Section 4.1 we demonstrated the unnecessary overheads of patch distribution on a symmetric problem decomposition. However, this is an unrealistic representation of typical input data, as such data is unlikely to decompose symmetrically across a range of processors. Whilst a lack of symmetry in decomposition does not inherently imply load imbalance, we can no longer guarantee it. Patch distribution strategies may improve performance by reducing this load imbalance.

In this section we illustrate the benefit of patch distribution on asymmetric decompositions. Using an single quadrant of Figure 2(a), presented in Figure 4(a), we obtain a naturally asymmetric decomposition, through which we can compare the two distribution strategies by analysing the impact of the load imbalance. Without distribution, all of the patches obtained through mesh refinement will be assigned to the same local processor, illustrated in Figure 4(c), rather than being shared between all processors.



(a) A simple, asymmetric, three material problem.

(b) The initial mesh, including refined areas.

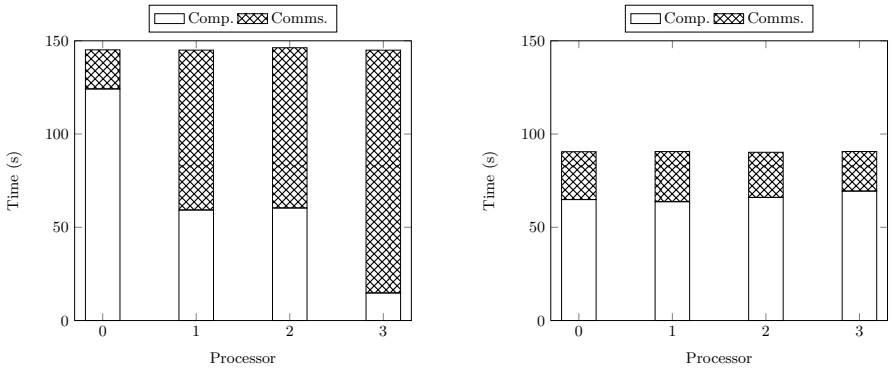
(c) Patch assignment without distribution.

(d) Patch assignment with distribution.

Fig. 4: An asymmetric multi-material problem, with the corresponding AMR mesh and patch distributions.

In Figure 5 we present the breakdown of computation and communication time for each processor, illustrating the available performance gains afforded through patch distribution. The disparity between computational workload is illustrated in the communication times of the remaining processors as they wait for the overloaded processor. For patch distribution we see a reduction in the worst case communication times, from 120s down to 24s, and a levelling of computation times, resulting in a reduction in runtime.

This extreme case illustrates the runtime increases created when the load imbalance is maximal, thus highlighting the benefit of patch distribution to load balance in AMR applications. The cost of this technique is the 25% of time spent in the communication phase, which has wide implications in the scalability of the code.



(a) Non distributed asymmetric mesh. (b) Distributed asymmetric mesh.

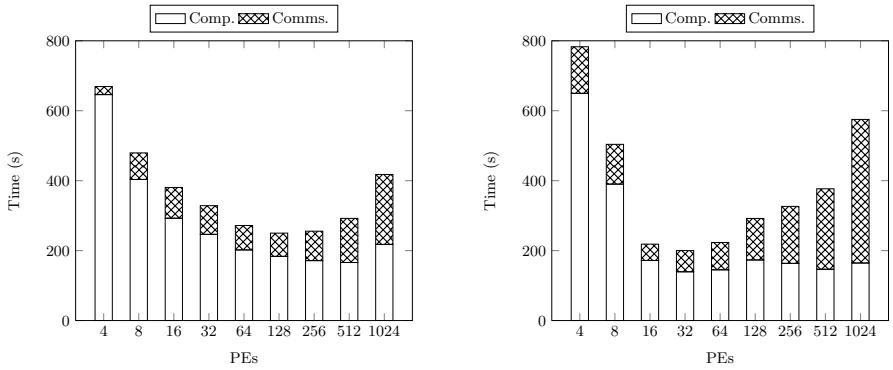
Fig. 5: Runtime breakdown for an asymmetric decomposition on four cores.

5 Scalability Study

Section 4 illustrates the disparity between performing AMR with and without patch distribution enabled, through both symmetric and asymmetric problem decompositions. Both of the examples presented were designed to illustrate the performance of the techniques in ideal circumstances. However, in more realistic scenarios a middle ground between the two examples is likely to be observed.

To illustrate this more realistic scenario we perform a scalability study of the two techniques applied to the same problem decomposition. For this study we use the problem from Figure 2(a), which is initially symmetrically decomposed, and strong scale it, by increasing the processor count but keeping the global problem size constant. For our study we selected a problem comprised of 1 million cells, computed on a selection of processor counts ranging from 4 to 1024, in powers of 2. Evaluating the proportion of runtime spent in computation and communication allows us to comment on the scalability of each distribution strategy.

Figure 6 presents the breakdown of computation and communication times for runs of the symmetric problem at 9 different processor counts on Hera. The reduction in communication time with patch distribution disabled is shown to vastly increase the scalability of the code. Initially, the problem is symmetrically decomposed over 4 processors, hence the lower runtime when patches are not distributed. As core counts increase, the computation and communication times become more diverse between the two distribution strategies. Whilst the lowest overall runtime time is found on 32 processors with patch distribution enabled, as the number of processors increases communication becomes the dominant factor – accounting for 71.4% of the total runtime on 1024 processors. When patch distribution is disabled, we no longer see such a marked increase in communication times and total runtime, although average computation time remains higher due to the load imbalance. It is this trade off, spending more time computing



(a) Patch distribution disabled.

(b) Patch distribution enabled.

Fig. 6: Scalability study of strong scaled AMR problem evaluating patch distribution.

locally in order to communicate less, that provides the improved run times at scale.

In the following sections we try to address the balance between computation time and communication time, finding a threshold to achieve both a reduction in runtime and an increase in scalability.

6 Level-Based Distribution Threshold

In Section 5 we demonstrated the influence of problem symmetry in the performance of patch distribution techniques. In the following section we illustrate how applying a distribution level threshold can form a 'middle ground' between these two techniques. To control communication overheads we apply a threshold on patch refinement levels for distribution. With a threshold level of 2, only patches at the second level of refinement and higher will be considered for distribution. With a higher distribution threshold, we will increase the load imbalance, but minimise the communication time, and vice versa.

Using the asymmetric problem presented in Figure 4(a) we demonstrate the effect of a patch distribution threshold as the problem is strong scaled. Figure 7 shows how the different threshold levels span the runtimes of fully enabling or disabling patch distribution. What is also clear, and intuitive, is that all threshold-based results are bounded. Therefore the motivation for a threshold-based patch distribution strategy lies in risk mitigation. Without ahead-of-time runtime prediction, through modelling or execution, the best strategy is unknown, thus a threshold strategy reduces risk of poor performance and scalability.

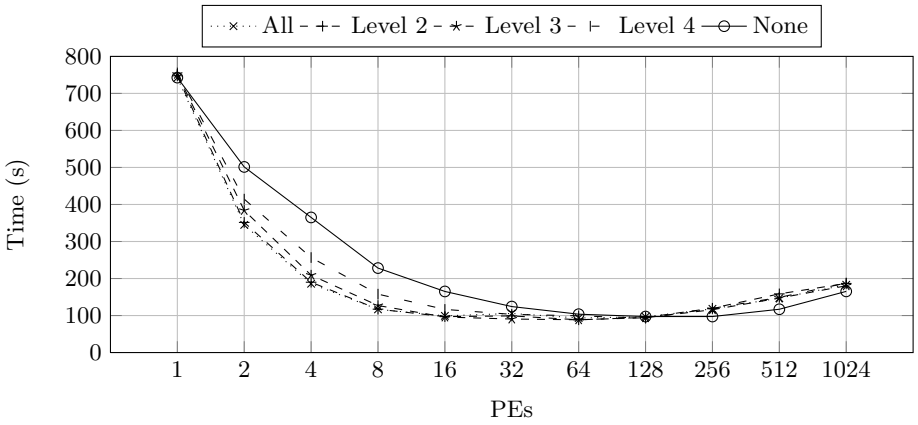


Fig. 7: Scalability analysis of patch distribution threshold levels.

7 Optimising Patch Distribution

In previous sections we have demonstrated the importance of using patch distribution techniques to balance computational work across processors against the communication costs incurred by data transfers. However, these schemes were developed using simple, intuitive heuristics, without consideration for the more complex relationship between computation time and communication time.

To make more optimal decisions about the distribution of patches during program execution we utilise a model based on three measured parameters: computation time per cell (W_g), network latency (l), and network bandwidth (G). We use these parameters to estimate the time required to compute and communicate a patch q on a processor p using the following formula:

$$t_{q,p} = W_g\left(cells_p\right) \times \left(l_p\left(q\right) + \frac{q}{G_p\left(q\right)}\right) \tag{1}$$

where l_p and G_p are the latency and bandwidth obtained when sending the patch to processor p. These network parameter values will change based on the type of communication (on or off-node) being performed.

Rather than evaluating only computational workload, we take communication overheads into account to select the most appropriate processor for the current patch, considering: (i) increased communication time incurred by the patch, and (ii) how this overhead compares with the estimated computational saving. Our new patch distribution scheme utilises this information by maintaining a list of the total estimated work on each processor, and selecting a processor for a given patch that will increase the current maximum runtime the least. Equation 2 describes this scheme mathematically:

$$p_q = \min_{p \in processors} \left(t_p + t_{q,p}\right) \tag{2}$$

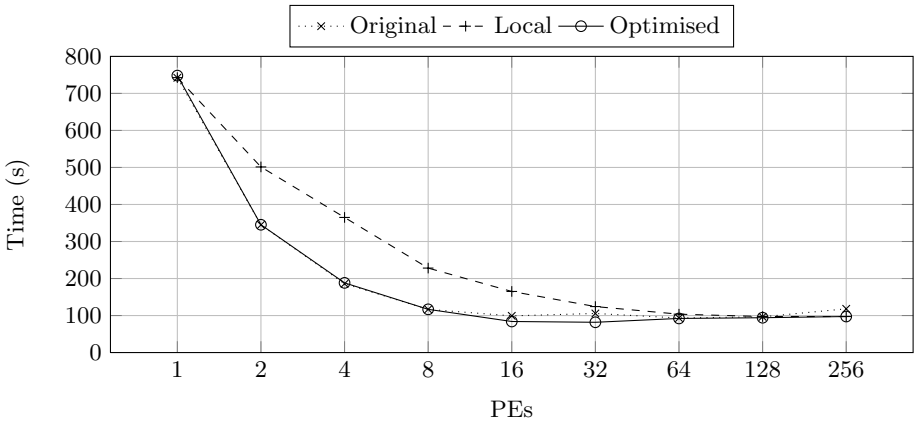


Fig. 8: Runtime comparison of patch distribution strategies.

where p_q is the processor selected for patch q, and t_p is the total estimated work time currently assigned to processor p.

Using W_g values measured from a single processor run on the target system, and latency and bandwidth values estimated using the SKaMPI benchmark [19], Figure 8 demonstrates the obtained performance increases of our model-based patch distribution strategy.

The performance of our optimised strategy is generally equivalent to the best performance of out the two previous strategies, but out performs it in certain configurations, on 16 and 32 cores by 18.1% and 29.1% respectively. The lack of performance increase in other configurations is attributed to lack of sufficient computation and the increased overheads of maintaining this new mapping. More efficient metadata management would decrease the overhead of patch selection and improve performance of the optimised strategy.

8 Conclusions and Future Work

As the size of high-performance computers increases, applications will be constrained not by computational power, but by data availability. AMR presents a technique to increase the efficiency of computation by focusing work on areas of interest. This, however, will create an imbalance of work, as refined patches will typically occur in highly localised areas of the problem. Load balancing can be used to reduce load imbalance at the expense of increased communication costs. We have demonstrated the benefits and limitations of patch distribution strategies on both symmetric and asymmetric problem decompositions. Additionally we have shown how a threshold-based distribution strategy can mitigate risk between the two extremes. By utilising an optimised patch-distribution strategy that considers the runtime impact of patch distribution, we can make informed decisions about more optimal patch locations, with up to 29% improvements over

the current strategies. In future research we plan on employing a more sophisticated performance model of the Shamrock code to enhance performance predictions to improve patch distribution decisions. Combined with a more efficient metadata management strategy, runtime improvements offered by the optimised distribution strategy are expected to be even more significant. We will combine this investigation with work to port Shamrock to an external AMR framework, to benefit from a wide variety of research into scalable AMR techniques.

Acknowledgements

We wish to give special thanks to Todd Gamblin and Scott Futral for providing continued assistance in accessing the Open Compute Facility resources at Lawrence Livermore National Laboratory, specifically the Hera machine on which the majority of experiments in this paper have been performed.

This work is supported in part by The Royal Society through their Industry Fellowship Scheme (IF090020/AM) and by the UK Atomic Weapons Establishment under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme). The performance modelling research is also supported jointly by AWE and the TSB Knowledge Transfer Partnership grant number KTP006740.

References

1. Berger, M.J., Oliger, J.: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. Journal of Computational Physics **53**(3) (March 1984) 484–512
2. Berger, M.J., Colella, P.: Local Adaptive Mesh Refinement for Shock Hydrodynamics. Journal of Computational Physics **82**(1) (May 1989) 64–84
3. Herdman, J.A., Gaudin, W.P., Turland, D., Hammond, S.D.: Benchmarking and Modelling of POWER7, Westmere, BG/P, and GPUs: An Industry Case Study. SIGMETRICS Performance Evaluation Review **38**(4) (March 2011) 16–22
4. Bryan, G.: Fluids in the Universe: Adaptive Mesh Refinement in Cosmology. Computing in Science & Engineering **1**(2) (April 1999) 46–53
5. Wu, J., Gonzalez, R., Lan, Z., Gnedin, N., Kravtsov, A., Rudd, D., Yu, Y.: Performance Emulation of Cell-Based AMR Cosmology Simulations. In: Proceedings of the 13th IEEE International Conference on Cluster Computing. (September 2011) 8–16
6. Fryxell, B., Olson, K., Ricker, P., Timmes, F., Zingale, M., Lamb, D., MacNeice, P., Rosner, R., Truran, J., Tufo, H.: FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. The Astrophysical Journal Supplement Series **131** (November 2000) 273–334
7. Baeza, A., Mulet, P.: Adaptive mesh refinement techniques for high-order shock capturing schemes for multi-dimensional hydrodynamic simulations. International Journal for Numerical Methods in Fluids **52**(4) (October 2006) 455–471
8. Quirk, J.: A parallel adaptive grid algorithm for computational shock hydrodynamics. Applied numerical mathematics **20**(4) (April 1996) 427–453

9. Luitjens, J., Berzins, M.: Scalable parallel regridding algorithms for block-structured adaptive mesh refinement. Concurrency and Computation: Practice & Experience **23**(13) (September 2011) 1522–1537
10. Berger, M.J., Rigoutsos, I.: An Algorithm for Point Clustering and Grid Generation. IEEE Transactions on Systems Man and Cybernetics **21**(5) (October 1991) 1278–1286
11. Lan, Z., Taylor, V., Bryan, G.: Dynamic Load Balancing for Structured Adaptive Mesh Refinement Applications. In: Proceedings of the 30th International Conference on Parallel Processing. (September 2001) 571–579
12. Sinha, S., Parashar, M.: Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. Proceedings of the 3rd IEEE International Conference on Cluster Computing **22** (October 2001) 435–442
13. Davis, J.A., Mudalige, G.R., Hammond, S.D., Herdman, J.A., Miller, I., Jarvis, S.A.: Predictive Analysis of a Hydrodynamics Application on Large-Scale CMP Clusters. Computer Science - Research and Development **26**(3-4) (June 2011) 175–185
14. Mudalige, G.R., Vernon, M.K., Jarvis, S.A.: A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures. In: Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium. (April 2008) 1–14
15. Colella, P., Bell, J., Keen, N., Ligocki, T., Lijewski, M., Straalen, B.: Performance and Scaling of Locally-Structured Grid Methods for Partial Differential Equations. Journal of Physics: Conference Series **78** (July 2007) (13) 012013
16. Van Straalen, B., Shalf, J., Ligocki, T., Keen, N., Yang, W.S.: Scalability Challenges for Massively Parallel AMR Applications. In: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium. (May 2009) 1–12
17. Wissink, A.M., Hornung, R.D., Kohn, S.R., Smith, S.S., Elliott, N.: Large Scale Parallel Structured AMR Calculations Using the SAMRAI Framework. In: Proceedings of the 14th ACM/IEEE Conference on Supercomputing. (November 2001) 6–19
18. Colella, P., Graves, D., Ligocki, T., Martin, D.: Chombo Software Package for AMR Applications Design Document. Technical report, Lawrence Berkeley National Laboratory (April 2009)
19. Reussner, R., Sanders, P., Prechelt, L., Müller, M.: SKaMPI: A Detailed, Accurate MPI Benchmark. In: Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. (September 1998) 52–59