jh160014–ISH

Mohamed Wahib, RIKEN Advanced Institute for Computational Science

Data Locality Optimization Strategies for AMR Applications on GPU-accelerated Supercomputers

Problem Statement

Motivation

Adaptive Mesh Refinement (AMR), which is a model for adapting the resolution of a stencil mesh locally. AMR is one of the paths to multi-scale exascale applications. However, producing efficient AMR code is hard, especially for GPUs. As a result, typical AMR frameworks require the user to write his own optimized code for the target architecture. In addition the generated AMR code is not optimized for reducing data transfer in GPU-accelerated supercomputers.

Our approach to achieve target

- A compiler-based framework for producing efficient AMR code (for GPUs)
- Architecture-independent interface provided to the user
- A performance model for quantifying the efficiency

Key results

Our framework generates code comparable in speedup and scalability to hand-written optimized GPU AMR implementations using up to ~1000 GPUs.

AMR Framework

Octree-based AMR

The mesh is organized into a hierarchy of refinement levels. The mesh is usually decomposed into relatively small fixed-sized octants of mesh cells.

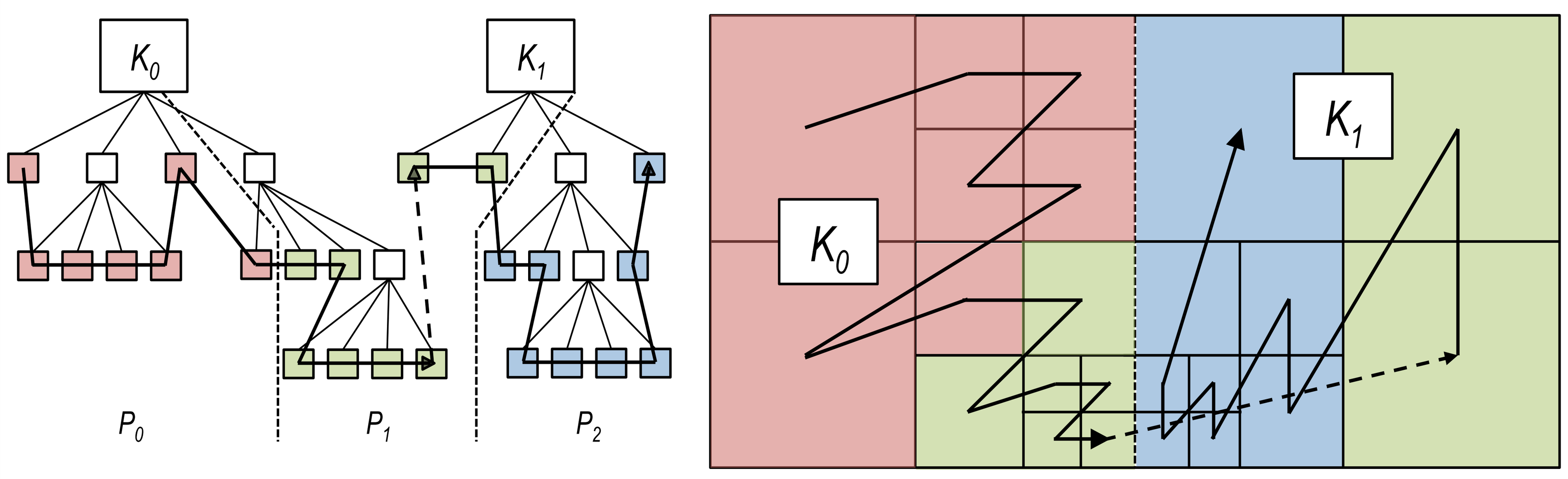

Figure 1: An example of a quadtree for a 2D AMR mesh (work divided on three processors)

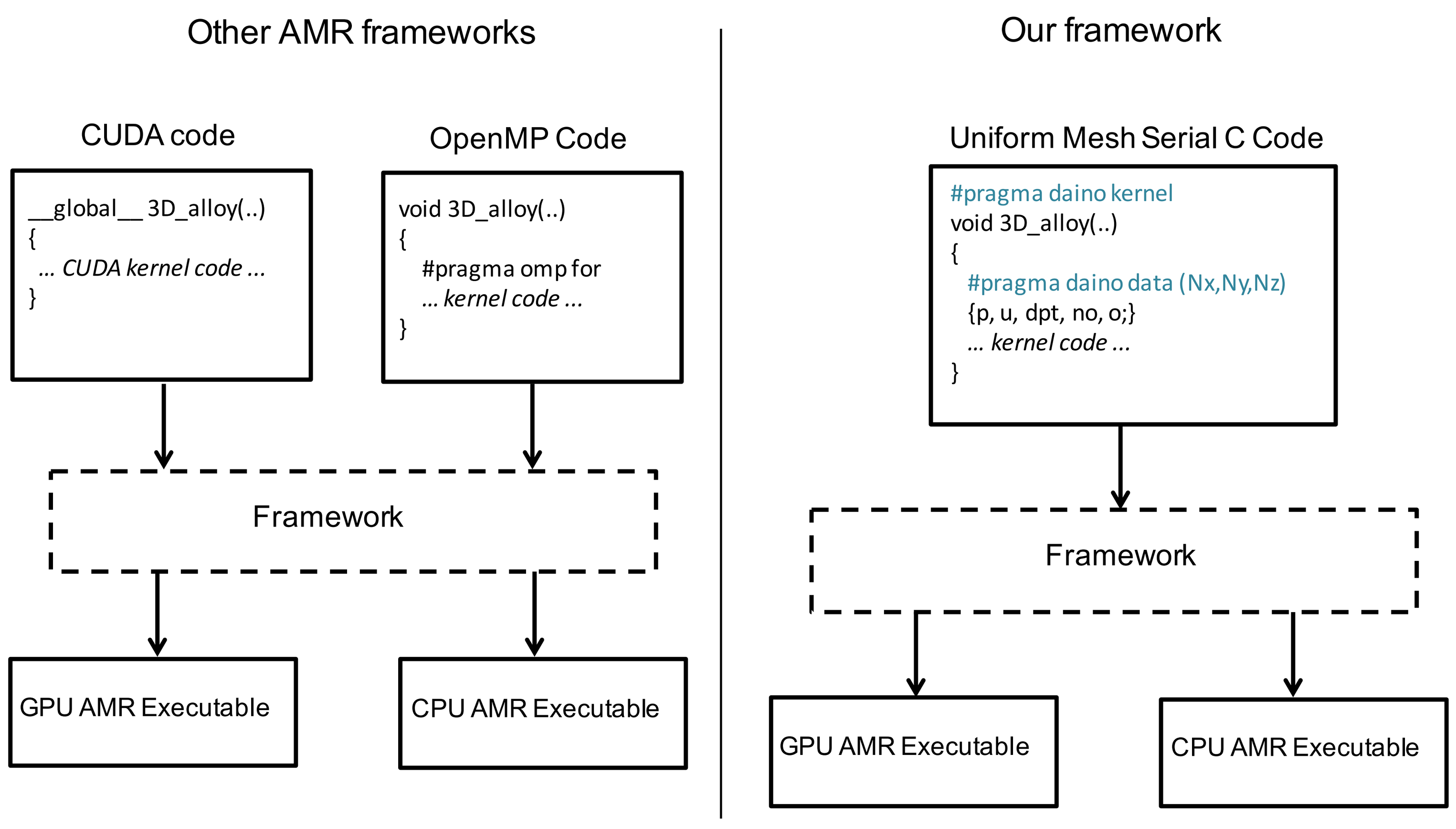

Figure 2: An illustration of the architecture-neutral interface used in the framework

Architecture-independent interface

The framework is composed of a compiler and runtime. The input to the framework is serial code applying stencil operations on a uniform grid. The user adds directives to identify the stencil functions and relevant data arrays.

```
#pragma dno kernel
void func(float ***a, float ***b, ..) {
#pragma dno data domName(i, j, k)
a, b;
#pragma dno timeloop
for(int t; t< TIME_MAX;t++) {
for(int i; i<NX; i++)
 for(int j; i<NY; j++) {
    ... // comput. not related to a and b
   for(int k; k<NZ; k++) {
    a[i][j][k] = c*(b[i-1][j][k] + \
     b[i+1][j][k] + b[i][j][k])+ \
     b[i][j+1][k] + b[i][j-1][k]);
} } } }
```

Figure 3: Minimal example directives

Data-centric AMR

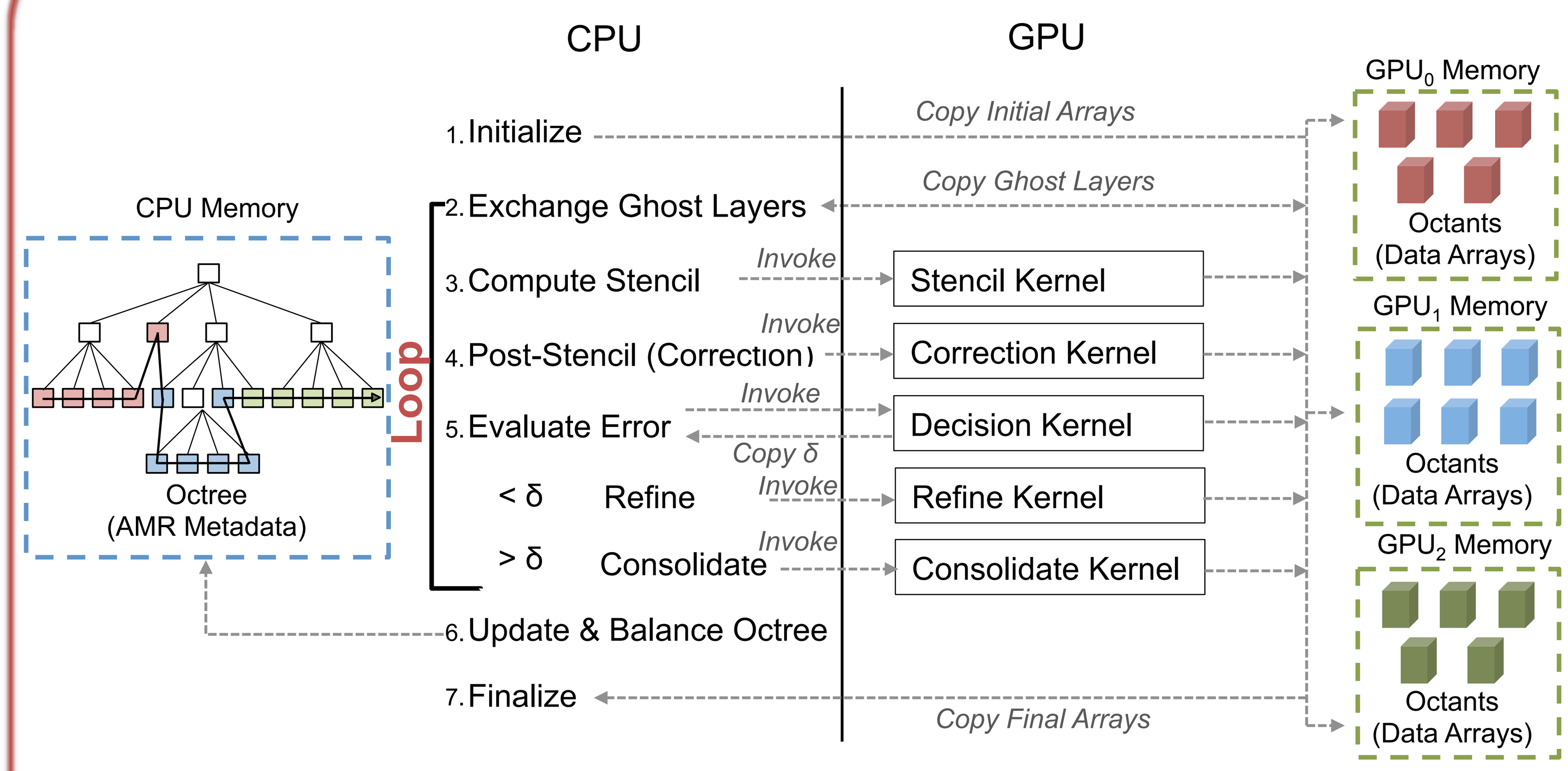

Figure 4: Conceptual overview of the proposed model. All operations touching the data arrays are done by GPU kernels. Accordingly, CPU specializes in operations applied on the octree while GPU specializes in operations applied on the data arrays.

Framework implementation

The implementation is based on LLVM compiler infrastructure.

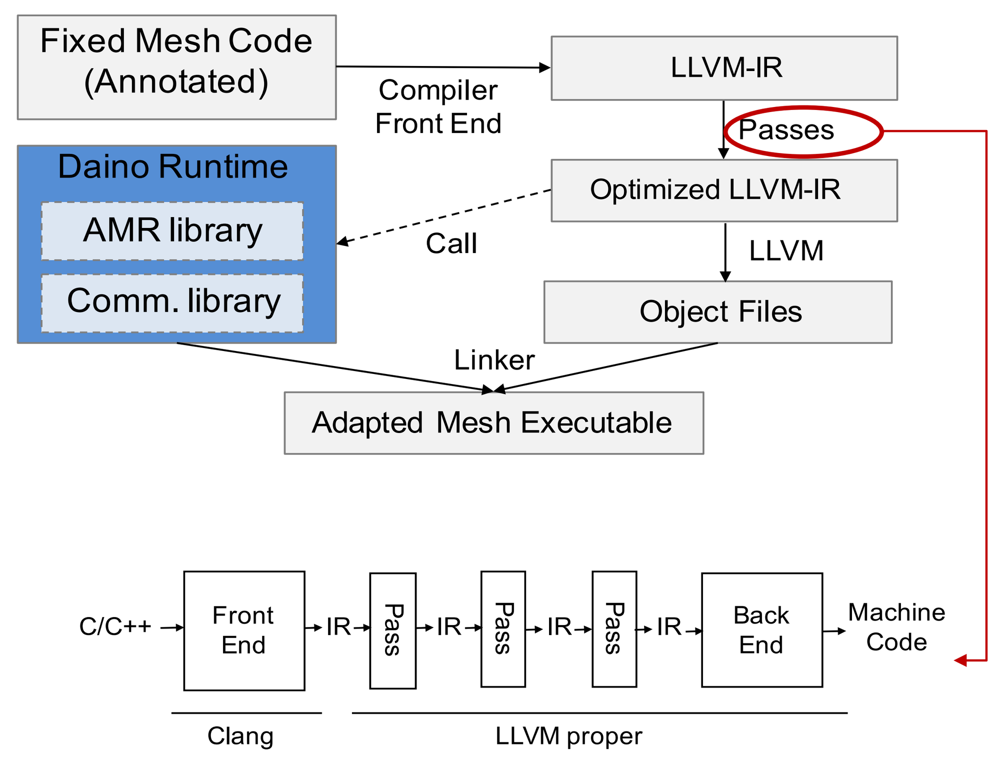

Figure 5: Apply translations and optimizations as compiler passes

Results

Applications

Hydrodynamics Solver: We model a hydrodynamics application using Euler equations extending the GAMER implementation [2].

Shallow-water Solver: We model shallow water simulations by depth-averaging the Navier–Stokes equations.

Phase-field Simulation: We evaluate an AMR version of a phase-field simulation for modeling 3D dendritic growth [3].

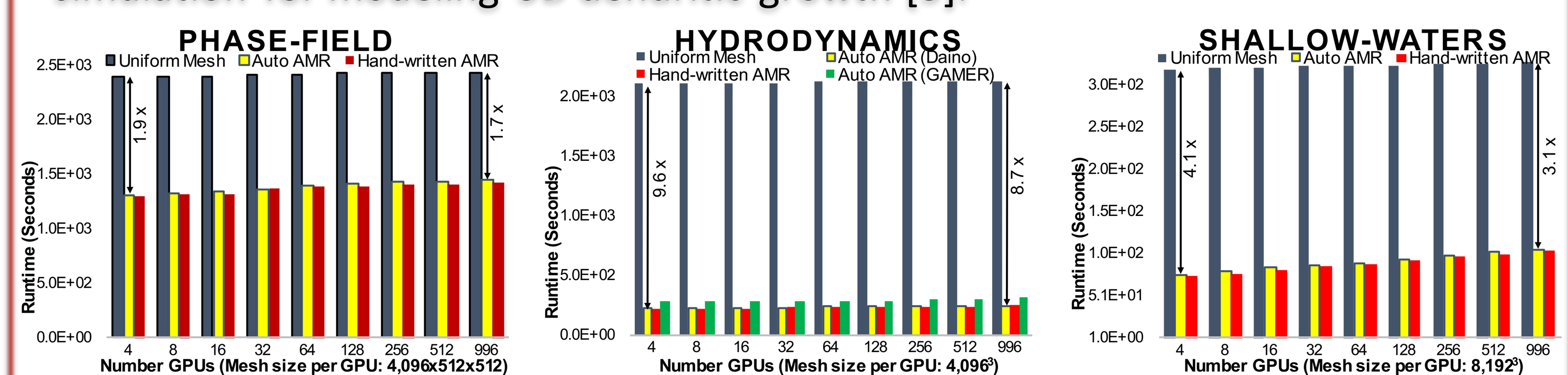

Figure 6: Weak scaling of uniform mesh, hand-written and automated AMR (GAMER-generated AMR included in hydrodynamic)

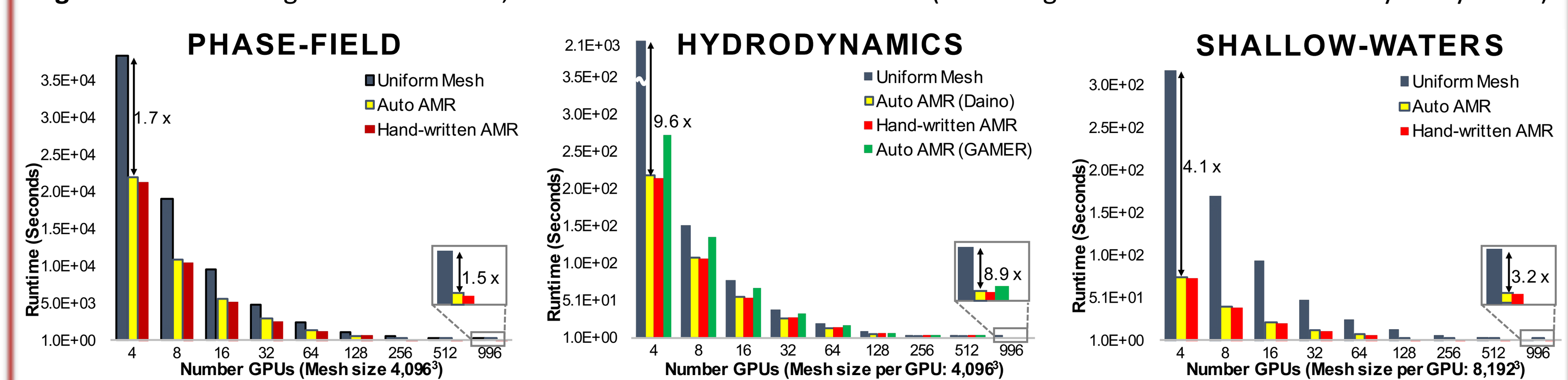

Figure 7: Strong scaling of uniform mesh, hand-written and automated AMR (GAMER-generated AMR included in hydrodynamic)

Table 1: Phase-field runtime breakdown (%) for Daino AMR

GPUs	Stencil (GPU)	AMR				Total
		Ld. Blc.	Remesh	Ghost	2:1 Blc.	
32	88.7%	8.8%	<1%	1.8%	<1%	100%
128	87.2%	9.7%	<1%	2.7%	<1%	100%
512	84.3%	11.1%	<1%	3.2	1.2	100%

Table 2: Hydrodyn. runtime breakdown (%) for Daino AMR

GPUs	Stencil (GPU)	AMR				Total
		Ld. Blc.	Remesh	Ghost	2:1 Blc.	
32	94.1%	4.5%	<1%	1.2%	<1%	100%
128	91.8%	5.5%	<1%	1.9%	<1%	100%
512	88.0%	8.3%	<1%	2.4	1.1	100%

Table 3: Shallow. runtime breakdown (%) for Daino AMR

GPUs	Stencil (GPU)	AMR				Total
		Ld. Blc.	Remesh	Ghost	2:1 Blc.	
32	93.7%	4.7%	<1%	1.2%	<1%	100%
128	92.0%	6.1%	<1%	1.6%	<1%	100%
512	90.5%	7.3%	<1%	1.9	<1%	100%