A survey of high level frameworks in block-structured adaptive mesh refinement packages

Anshu Dubey [a,*], Ann Almgren [a], John Bell [a], Martin Berzins [j], Steve Brandt [f,g], Greg Bryan [k], Phillip Colella [a,l], Daniel Graves [a], Michael Lijewski [a], Frank Löffler [f], Brian O'Shea [c,d,e], Erik Schnetter [h,i,f], Brian Van Straalen [a], Klaus Weide [b]

[a] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[b] Flash Center for Computational Science, The University of Chicago, Chicago 60637, USA
[c] Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, USA
[d] Lyman Briggs College, Michigan State University, East Lansing, MI 48824, USA
[e] The Institute for Cyber-Enabled Research, Michigan State University, East Lansing, MI 48824, USA
[f] Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803, USA
[g] Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA
[h] Perimeter Institute for Theoretical Physics, Waterloo, ON N2L 2Y5, Canada
[i] Department of Physics, University of Guelph, Guelph, ON N1G 2W1, Canada
[j] Mathematics and School of Computing, University of Utah, Salt Lake City, UT 84112, USA
[k] Department of Astronomy, Columbia University, New York, NY 10025, USA
[l] Computer Science Department, University of California, Berkeley, CA 94720, USA

HIGHLIGHTS

- A survey of mature openly available state-of-the-art structured AMR libraries and codes.
- Discussion of their frameworks, challenges and design trade-offs.
- Directions being pursued by the codes to prepare for the future many-core and heterogeneous platforms.

ARTICLE INFO

ABSTRACT

Over the last decade block-structured adaptive mesh refinement (SAMR) has found increasing use in large, publicly available codes and frameworks. SAMR frameworks have evolved along different paths. Some have stayed focused on specific domain areas, others have pursued a more general functionality, providing the building blocks for a larger variety of applications. In this survey paper we examine a representative set of SAMR packages and SAMR-based codes that have been in existence for half a decade or more, have a reasonably sized and active user base outside of their home institutions, and are publicly available. The set consists of a mix of SAMR packages and application codes that cover a broad range of scientific domains. We look at their high-level frameworks, their design trade-offs and their approach to dealing with the advent of radical changes in hardware architecture. The codes included in this survey are BoxLib, Cactus, Chombo, Enzo, FLASH, and Uintah.

Published by Elsevier Inc.

1. Introduction

Block-structured adaptive mesh refinement (SAMR) [8,7] first appeared as a computational technique almost 30 years ago; since then it has been used in many individual research codes and, increasingly over the last decade, in large, publicly available code frameworks and application codes. The first uses of SAMR focused almost entirely on explicit methods for compressible hydrodynamics, and these types of problems motivated the building of many of the large code frameworks. SAMR frameworks have evolved along different paths. Some have stayed focused on specific domain areas, adding large amounts of functionality and problem-specific

University of Utah Institutional Repository
Author Manuscript

ARTICLE IN PRESS

2   *A. Dubey et al. / J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮*

physics modules that are relevant to those applications. Examples of these include AstroBEAR [26], CRASH [91], Cactus [42,15], Enzo [14,38], FLASH [40,32], Overture [76], PLUTO [68,89], and Uintah [77,78]. Other frameworks have pursued a more general functionality, providing the building blocks for a larger variety of applications while enabling domain-specific codes to be built using that framework. As an example, while almost every SAMR framework can be used to solve systems of hyperbolic conservation laws explicitly, not all frameworks include the functionality to solve elliptic equations accurately on the entire hierarchy or a subset of levels. Examples of frameworks constructed specifically for solving hyperbolic conservation laws include AMROC [29] and AMRClaw [5], both based on the wave propagation algorithms of R. LeVeque. Extensions of AMRClaw include GeoClaw [41], the widely used tsunami simulation tool. BoxLib [13], Chombo [23], Jasmine [73] and SAMRAI [17,47] are more general in that they supply full functionality for solving equation sets containing hyperbolic, parabolic and elliptic equations, and facilitate the development of codes for simulating a wide variety of different applications. PARAMESH [39] supplies only the mesh management capability and as such is equation-independent. A more comprehensive list of codes that use SAMR, and other useful adaptive mesh refinement (AMR) resources can be found at [46].

SAMR codes all rely on the same fundamental concept, viz. that the solution can be computed in different regions of the domain with different spatial resolutions, where each region at a particular resolution has a logically rectangular structure. In some SAMR codes the data is organized by level, so that the description of the hierarchy is fundamentally defined by the union of blocks at each level; while others organize their data with unique parent–child relationships. Along with the spatial decomposition, different codes solving time-dependent equations make different assumptions about the time stepping, i.e., whether blocks at all levels advance at the same time step, or blocks advance with a time step unique to their level. Finally, even when frameworks are used to solve exactly the same equations with exactly the same algorithm, the performance can vary due to the fact that different frameworks are written in different languages, with different choices of data layout, and also differ in other implementation details. However, despite their differences in infrastructure and target domain applications, the codes have many aspects that are similar, and many of these codes follow a set of very similar software engineering practices.

In this survey paper we examine a representative set of SAMR packages and SAMR-based codes that: (1) have been in existence for half a decade or more, (2) have a reasonably sized and active user base outside of their home institutions, and most importantly, (3) are publicly available to any interested user. In selecting the codes we have taken care to include variations in spatial and temporal refinement practices, load distribution and meta-information management. Therefore, we have octree and patch based SAMR, no subcycling and subcycling done in different ways, load distribution on a level by level or all levels at once, and globally replicated meta-data or local view. Additionally, the set covers a broad range of scientific domains that use SAMR technology in different ways. We look at their high-level frameworks, consider the trade-offs between various approaches, and the challenges posed by the advent of radical changes in hardware architecture. The codes studied in detail in this survey are BoxLib, Cactus, Chombo, Enzo, FLASH, and Uintah. The application domains covered by a union of these codes include astrophysics, cosmology, general relativity, combustion, climate science, subsurface flow, turbulence, fluid–structure interactions, plasma physics, and particle accelerators.

2. Overview of the codes

BoxLib is primarily a framework for building massively parallel SAMR applications. The goals of the BoxLib framework are twofold:

first, to support the rapid development, implementation and testing of new algorithms in a massively parallel SAMR framework; and second, to provide the basis for large-scale domain-specific simulation codes to be used for numerical investigations of phenomena in fields such as astrophysics, cosmology, subsurface flow, turbulent combustion, and any other field which can be fundamentally described by time-dependent PDE's (with additional source terms, constraints, etc.). In accordance with both goals, the core remains relatively agile and agnostic, and is not tied to a particular time-stepping or spatial discretization strategy, or to a particular set of physics packages.

That said, while BoxLib itself supplies a very general capability for solving time-dependent PDE's on an adaptive mesh hierarchy, there are a number of large, domain-specific BoxLib-based application codes in scientific use today. The most widely used include CASTRO [3,93,94] for fully compressible radiation-hydrodynamics; MAESTRO [74], for low Mach number astrophysical flows; Nyx [4] for cosmological applications; LMC [28] for low Mach number combustion; and the structured grid component of the Amanzi code for modeling subsurface flow [79].

Chombo is an offshoot of the BoxLib framework, having branched off from BoxLib in 1998. As a result Chombo shares many features with BoxLib, including the hybrid C++/Fortran approach, and the separation of concerns between the parts that are best handled in C++ (abstractions, memory management, I/O, flow control) and Fortran dialects (loop parallelism, stencil computations). Chombo articulates its APIs explicitly for easier plugin by the client application code. It has also diverged from BoxLib in the design of its data containers. Chombo currently supports applications in a range of disciplines, including the following: MHD for tokamaks using all-speed projection methods, and large eddy simulations of wind turbines at KAUST; compressible CFD + collision-less particle cosmology simulations (CHARM code, [70,71]); physics of the solar wind and its interaction with the interstellar medium using compressible hyperbolic CFD + electromagnetic and kinetic effects (MS-FLUKSS code [61,81,80]); general astrophysics modeling (PLUTO code [69]); astrophysical MHD turbulence (ORION code [56]); SF Bay and Delta hydrology modeling – shallow water (Realm code [6]); plasma-wakefield accelerators – compressible viscous flow at LBNL; blood flow in cerebral arteries – fluid/solid coupling (UCB ParLab project [30]); pore-scale subsurface reacting flow (Chombo-Crunch code, [72]); conjugate heat transfer in nuclear reactors [25]); 4D gyrokinetic models of tokamak edge plasmas (COGENT code, [31,18]); land ice model for climate simulation(BISICLES code, [24]); and atmospheric models for climate simulation—low-Mach number CFD at University of Michigan.

Cactus [42,15] was designed as a general-purpose software framework for high-performance computing with AMR as one of its features. The first set of applications that used the framework was astrophysical simulations of compact objects involving general relativity (GR) such as black holes and neutron stars. These scenarios require high resolution inside and near the compact objects, and at the same time need to track gravitational waves propagating to large distances of several hundred times the radii of the compact objects. This leads very naturally to the use of AMR. In GR, gravitational effects are described via hyperbolic (wave-type) equations, propagating at the speed of light. This removes the need for an elliptic solver that is necessary in Newtonian gravity to calculate the gravitational potential. While the Cactus framework is generic, its most prominent user today is the Einstein Toolkit [55,95,37], a large set of physics modules for relativistic astrophysics simulations. The Einstein Toolkit includes modules for solving the Einstein equations and relativistic magneto-hydrodynamics, as well as modules for initial conditions, analysis, and so on.

Enzo is a standalone application code [14,38] that was originally designed to simulate the formation of large-scale cosmological structure, such as clusters of galaxies and the intergalactic medium. Since the formation of structures in the Universe is a

University of Utah Institutional Repository
Author Manuscript

A. Dubey et al. / J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

3

process that is driven by gravitational collapse, the study of this phenomenon naturally requires high resolution in both space and time, making AMR a logical choice. Since the first version of Enzo was written in 1996, the user base has grown to include roughly 100 researchers studying a variety of astrophysical phenomena, including galaxies, galaxy clusters, the interstellar and intergalactic media, turbulence, and star formation in the early universe and in our own Galaxy. Because of this growth, a wide range of capabilities have been added to the Enzo code, including a range of hydrodynamic and magnetohydrodynamic solvers, implicit flux-limited radiation diffusion and explicit radiation transport with a ray-casting method, optically-thin and thick radiative cooling, prescriptions for active particles and passive tracer particles, and a wide variety of problem types that users can expand upon to pursue their own interests.

FLASH [32,35,40] was originally designed for simulating astrophysical phenomena dominated by compressible reactive flows. The target applications also had multiple physical scales and therefore required AMR, which was provided by the octree-based PARAMESH package [39]. Though PARAMESH supports subcycling in time FLASH does not. FLASH underwent three iterations of infrastructure refactoring, and the resultant architecture has enabled the code to be readily extensible. As a result, capabilities have been added to the code to make it useful for such disparate communities such as cosmology, astrophysics, high-energy-density physics, computational fluid dynamics, and fluid–structure interactions. FLASH's capabilities include solvers for hydrodynamics, magnetohydrodynamics, self-gravity, flux-limited radiation, several specialized equations of state, several source terms including nuclear burning and laser driver, material properties, tracer and active particles, and immersed boundaries.

The Uintah software was initially a direct outcome from the University of Utah DOE Center for the Simulation of Accidental Fires and Explosions (C-SAFE) [78] that focused on providing state-of-the-art, science-based tools for the numerical simulation of accidental fires and explosions. The Uintah framework allows chemistry and engineering physics to be fully coupled with nonlinear solvers and visualization packages. The Uintah open-source (MIT License) software has been widely ported and used for many different types of problems involving fluid, solid, and fluid–structure interaction problems. The present status of Uintah is described by [10].

Uintah presently contains four main simulation algorithms, or components: (1) the ICE [51,52] compressible multi-material finite-volume CFD component, (2) the particle-based Material Point Method (MPM) [88] for structural mechanics, (3) the combined fluid–structure interaction (FSI) algorithm MPM-ICE [44,45,43], and (4) the Arches turbulent reacting CFD component [87,86] that was designed for simulation of turbulent reacting flows with participating media radiation. Arches is a three-dimensional, Large Eddy Simulation (LES) code that uses a low-Mach number variable density formulation to simulate heat, mass, and momentum transport in reacting flows. Uintah has been used, and is being used, for many different projects such as angiogenesis, tissue engineering, heart injury modeling, blast-wave simulation, semiconductor design, and multi-scale materials research [10].

3. Frameworks

There are a number of similarities between the six codes/software frameworks (which from now on we will call "codes") described in this paper. Each of these codes provides some generic support for SAMR applications as well as more specialized support for specific applications. Since the codes detailed in the survey come from different disciplines, groups, and scientific domains they each use various terms in their own different ways. In order to facilitate the discussion we override individual code's usage and adhere to the following terminology:

- **cell**: the smallest unit of discretized domain
- **mesh/grid**: generic way of describing the discretized domain
- **block**: logically rectangular collection of cells
- **patch**: a collection of contiguous cells of the same size (at the same refinement level), patches may be subdivided into blocks
- **active cells**: cells in a block that are updated when an operator is applied to the block
- **guard cells**: halo of cells surrounding the active cells that are needed for computation of an operator, but are not updated by the operator
- **level**: union of blocks that have the same cell size
- **framework**: the infrastructure backbone of the code
- **component**: an encapsulated stand-alone functionality within the code

All codes perform domain decomposition into blocks. BoxLib, Cactus, Chombo and Uintah are perhaps the most general frameworks, in that the bulk of the software capability is not tied to a particular application. Enzo is perhaps the most specific in that it is designed for astrophysical and cosmological applications. As such, the Enzo release contains numerous modules for specific processes such as star formation and feedback. FLASH lies between BoxLib/Cactus/Chombo/Uintah and Enzo; it not only has extensive physics-independent infrastructure, but also includes physics-specific modules and solvers for a variety of applications in its releases. All of the codes support finite difference/finite volume methods in 1, 2 or 3 dimensions in Cartesian coordinates and all codes except Cactus support particle and particle/mesh algorithms, with both active and passive particles. Support is provided for data that lives on cell centers, faces, edges, or nodes. Except FLASH, all other codes support subcycling in time. The original parallelization model in these codes, as in most codes of similar vintage, was distributed memory with MPI, though now they have various degrees of hybrid parallelization (see Table 1).

In all of the codes, explicit hyperbolic solvers act upon individual blocks with no knowledge of other blocks once the guard cells have been filled from blocks at the same or coarser levels as appropriate. Explicit refluxing occurs at coarse–fine boundaries to correctly update the solution. The implicit and semi-implicit solvers place more demands on the communications and different codes handle them differently. An interesting observation is that the original implicit and semi-implicit solvers came in the form of geometric multigrid in those codes that had any. As the codes started supporting capabilities more demanding of such solvers they started to provide interfaces to the readily available capabilities from libraries such as PETSc and Hypre. Note that because of AMR, and therefore the presence of fine–coarse boundaries, these interfaces are non-trivial and involve considerable effort to do correctly. Chombo, Flash and Uintah have support for fluid/structure interaction—in Chombo, embedded boundaries are used to represent the fluid/solid interface; FLASH uses an immersed boundary representation, and Uintah uses the particle-based Material Point Method (MPM) for structural modeling. The following sections give more details about individual codes and Table 1 summarizes the important framework features and capabilities in various codes.

3.1. BoxLib

The main branch of BoxLib is a combination of C++/Fortran90 software; a newer branch of BoxLib is written in pure Fortran90 but currently supports only non-subcyling algorithms. BoxLib contains extensive support for both explicit and implicit grid-based operations as well as particles on hierarchical adaptive meshes. Single-level and multi-level multigrid solvers are included for cell-based and node-based data. The fundamental parallel abstraction in both the C++ and Fortran90 versions is the MultiFab, which holds the data on the union of blocks at a level. A MultiFab is composed of

Table 1

A summary of features in the SAMR codes and frameworks. In the above table GMG stands for "Geometric Multigrid", and FSI stands for "Fluid–Structure Interaction". The "Spherical" and "Cylindrical" AMR columns specify whether the AMR structure understands something other than Cartesian geometry. The entry "multipatch" denotes that Cactus can cover spherical or cylindrical regions by piecing together distorted but logically rectangular regions.

Feature	BoxLib	Cactus	Chombo	Enzo	FLASH	Uintah
Subcycling	Optional	Optional	Optional	Required	None	Required
Time step ratio	Same as refinement	Independent	Same as refinement	Independent		Same as refinement
Elliptic solver	PETSc/Hypre/Trilinos native GMG	User supplied	PETSc/native GMG	Hypre native GMG	Hypre/native GMG	PETSc/Hypre
GMG with AMR	Sub or whole mesh		Sub or whole mesh	Single level	Whole mesh	Sub or whole mesh
Spherical AMR	1D	Multipatch	1D		1, 2, or 3D	1D
Cylindrical AMR	2D	Multipatch	2D		1, 2, or 3D	2D
Mixed dimensions			Yes			
Highest dimension		4D	Up to 6D			
Block size	Variable	Variable	Variable	Variable	Fixed	Variable
Refine factor	2/4	2	2/4	Any integer	2	Any integer
Parent blocks	Not unique	Not unique	Not unique	Unique	Unique	Unique
Regridding level	Tag cells per level	Tag cells per level	Tag cells per level	Tag cells per level	Tag blocks all at once	Tag cells all at once
Space filling curve	Morton		Morton	Piano-Hilbert	Morton	Hilbert + fast sorting
OpenMP	Per block and loops	Dynamically tuned loops	Per block	Per block and loops	Per block and loops	
Accelerators		CUDA OpenCL		CUDA		CUDA
Parallel I/O	native	HDF5	HDF5	HDF5	HDF5 PnetCDF	HDF5
Viz	VisIt/yt	VisIt/yt	VisIt	VisIt/yt	VisIt/yt	VisIt
FSI			Embedded boundary		Immersed boundary	MPM Method
Framework language	C++/Fortran	C/C++	C++	C++	Fortran	C++
User module language	Fortran	C/C++ Fortran	Fortran	Fortran	Fortran	

multiple FABs (Fortran Array Boxes); each FAB is an array of data on a single block. During each MultiFab operation the FABs composing that MultiFab are distributed among the nodes; MultiFabs at each level of refinement are distributed independently. Each node holds *meta-data* that is needed to fully specify the geometry and processor assignments of the MultiFabs. The meta-data can be used to dynamically evaluate the necessary communication patterns for sharing data amongst processors in order to optimize communication patterns within the algorithm.

The scaling behavior of BoxLib depends strongly on the algorithm being implemented. Solving only a system of hyperbolic conservation laws, CASTRO has achieved excellent weak scaling to 196K cores on the Jaguar machine at the Oak Ridge Leadership Computing Facility (OLCF) using only MPI-based parallelism. Good scaling of linear solves is known to be much more difficult to achieve. Recent scaling results of the Nyx code, which relies on multigrid solves of the Poisson equation for self-gravity, demonstrate excellent scaling up to 49K cores on the Hopper machine at NERSC.

3.2. Chombo

Many features in Chombo, such as hybrid language model, are similar to BoxLib. Chombo's union of blocks is called a BoxLayout (with a specialization being a DisjointBoxLayout). These maintain the mapping of blocks to compute elements (nodes, or cores). This *meta-data* is replicated across all processors redundantly. In cases with extreme block counts ($O(10^6)$ blocks) this meta-data is compressed [92]. Meta-data is shared at the thread-level. Cell-based refinement codes have a different parameter space to operate in and can have significantly higher meta-data costs in return for higher floating-point efficiency.

Chombo keeps the FAB (Fortran Array Box) data member from BoxLib, but it is templated on data type and data centering. Chombo has a hierarchy of templated data holders (LayoutData, BoxLayoutData, and LevelData). An important reason for the

templated data holder design is to provide a common code base that also supports the EBCellFAB, a cell-centered finite volume data holder that supports embedded boundary algorithms [20–22,19], and the BinFAB to support Particle-In-Cell algorithms. Chombo also supports mapped multiblock domains and mixed-dimension domains up to 6D.

Chombo has built-in diagnostics to track time in functions (serial and parallel), memory leak tracking, memory tracing, sampling profilers, and the ability to trap and attach a native debugger to a running job as part of the default build environment without requiring third party packages. Chombo has demonstrated scaling behavior of both its hyperbolic gas dynamics codes and its multigrid elliptic solver to 196K cores on the Jaguar machine at the Oak Ridge Leadership Computing Facility (OLCF) using flat MPI-based parallelism [92].

3.3. Cactus/carpet

Cactus modules consist of routines targeting blocks, where the core Cactus framework manages when and how these routines are called. The framework itself does not provide parallelism or AMR; these are instead implemented via a special *driver* component. These days, *Carpet* [84,83,16] is the only widely-used driver. In principle, it would be possible to replace Carpet by an alternative driver that provides, e.g., a different AMR algorithm; if that new driver adhered to the existing interfaces, neither the framework nor existing physics modules would need to be modified.

Carpet supports two ways to define the grid hierarchy. Application components can explicitly describe locations, shapes, and depths of refined regions, which is useful, e.g., when tracking black holes or stars. Alternatively, the application can mark individual cells for refinement, and Carpet will then employ a tiling method (implemented in parallel) to find an efficient grid structure that encloses all marked points.

Cactus relies on a domain-specific language (DSL) describing its modules [1]. This DSL describes the schedule (workflow) of tasks in a Cactus computation, the grid functions (a distributed data struc-

University of Utah Institutional Repository
Author Manuscript

ARTICLE IN PRESS

A. Dubey et al. / J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

5

ture which contains the values of a field on every point of the grid), and parameter files. The framework provides an API that lets infrastructure components (e.g., the driver) query this information. One distinguishing feature of Cactus is that the components self-assemble—the information provided via this DSL is rich enough that components can simply be added to an existing simulation without explicitly specifying inter-component data flow. (Note that components have to be designed with this in mind.) This framework design requires Cactus applications to modularize functionality to a high degree.

Cactus is designed such that the executable is almost always compiled by the user, from the source code of Cactus and its modules, and, optionally, additional private modules. While the source code of all modules is typically stored in standard revision control systems, they may be of varying type, hosted by various research groups, spread across the world. Cactus provides convenience tools to automatically assemble a complete source tree from a given list of modules [85], and to compile on a large list of known supercomputers and standard environments [90].

This component-based design makes it possible to provide fairly interesting high level features which include, e.g., (1) externalizing the driver into a component as described above, (2) a generic time integration module that provides high-order coupling between independently-developed physics modules, or (3) an integrated web server for monitoring simulations that offers functionality similar to a debugger [53] or Web 2.0 integrations [2].

Kranc [49,54] is a Mathematica-based tool that generates complete Cactus components from equations specified in Mathematica syntax. In particular, Kranc allows tensorial equations to be written in a compact way employing abstract index notation. Kranc is able to apply a set of high-level optimizations that compilers are typically unable to perform, especially for large compute kernels (loop fission/fusion, SIMD vectorization). Kranc generates both C++ and OpenCL code.

3.4. Enzo

Enzo's SAMR supports arbitrary block sizes and aspect ratios (although blocks must be rectangular solids, and there are some practical limitations on their sizes and locations). Adaptive time-stepping is used throughout the code, with each level of the grid hierarchy taking its own time step. This adaptive time-stepping is absolutely critical to the study of gravitationally-driven astrophysical phenomena, since the local timescale for evolution of physical systems typically scales as $\rho^{-0.5}$, where ρ is the local density.

Similar to BoxLib and Chombo, Enzo uses C++ for the overall code infrastructure and memory management, and typically uses Fortran-90 for computationally-intensive solvers. Within the code, the fundamental object is the "grid patch", or block, and individual patches are composed of a number of baryon fields, as well as particles of a variety of types. The non-local solvers for gravity and implicit flux-limited diffusion require substantial amounts of communication—gravity solves currently use a FFT-based method on the root patch, and a multigrid relaxation-based method on subpatches. Threading is supported at block-level and deeper level, but the improvement in performance has been marginal with deeper threading compared to block-level parallelism. In addition, several of the hydrodynamic and magnetohydrodynamic solvers have been ported to graphics processing units (GPUs), with substantial speedup seen in situations where this physics dominates the computational time (e.g., driven compressible MHD turbulence). Using the hybrid-parallelized (MPI + OpenMP threads) version of Enzo, nearly-perfect weak scaling up to 130K cores on a Cray XK5 has been observed when the code is used in its unigrid (non-AMR) mode, and reasonable scaling of up to 32K cores has been observed on Blue Waters (a Cray XK7 machine) using AMR with adaptive time-stepping.

A noteworthy feature of the Enzo code is its development process, which has become completely open source and community-driven. The Enzo project was originally developed by the Laboratory for Computational Astrophysics at UIUC and then UC San Diego, but is now developed and maintained by a distributed team of scientists. Code improvements are funded at the individual PI level by a wide variety of sources, and user contribution to the source code is heavily encouraged. To facilitate this process, Enzo uses the distributed version control system Mercurial [67], which allows for extremely simple forking, branching, and merging of code bases. Coupled with an extensive answer testing framework [14] and a formalized system of peer review (managed by explicit "pull requests" from user forks to the main Enzo codebase), this enables the code to remain stable while at the same time directly incorporating feedback from non-core developers. User contribution is further encouraged by the code structure, which enables the straightforward addition of new physics modules, particularly those that are purely local in their impact (i.e., plasma chemistry, radiative cooling, star formation and feedback routines, etc.).

3.5. FLASH

FLASH combines two frameworks, an Eulerian discretized mesh and a Lagrangian framework [34] into one integrated code. Though most of the other codes discussed in the paper support Lagrangian particles in some form, a general purpose framework for Lagrangian data is unique to FLASH. The backbone of the overall code infrastructure is component-based, similar to Cactus. Instead of relying upon F90 object-oriented features for architecting the code, FLASH imposes its own object-oriented framework built using a very limited DSL and a Python-based configuration tool that together exploit the Unix directory structure for scoping and inheritance. FLASH's *Grid* unit manages the Eulerian mesh and all the associated data structures. Support exists for explicit stencil-type solvers using a homegrown uniform grid package, and for AMR using PARAMESH and Chombo. FLASH's explicit physics solvers are completely oblivious to the details of the underlying mesh and can switch between packages during application configuration. The elliptic and parabolic solvers need closer interaction with the mesh, therefore applications that require those solvers usually default to using PARAMESH, which has the most comprehensive solver coverage in FLASH.

The physics units that rely on their solvers interacting closely with the mesh are split into two components; the mechanics of the solvers that need to know the details of the mesh, but are agnostic to the physics, become sub-units within the Grid unit, while the sections that are physics-specific exist in their own separate physics units. Within the Grid unit, a unified API is provided for the various underlying solvers, some of which are interfaces to libraries such as Hypre. The unified API exploits the provision for co-existence of multiple alternative implementations of a given functionality to facilitate the use of the most appropriate solver for a specific application. This feature also allows re-use of the solvers for other purposes as needed.

The Lagrangian capabilities of FLASH have evolved into a framework of their own because they can be used in multiple ways, both for modeling physical particles directly and for providing mechanisms used by other code units. The mechanics of Lagrangian data movement is employed for implementing a *laser drive* for simulating laser-driven shock experiments, in addition to their original use for active and passive tracer particles. Similarly, an immersed boundary method for fluid–structure interaction makes use of mapping and data movement mechanics to couple the structure with the fluid through Lagrangian markers. FLASH has demonstrated scaling up to 130K cores in production runs and a million way parallelism in benchmarking on the BG/P and BG/Q platforms [33,27] respectively.
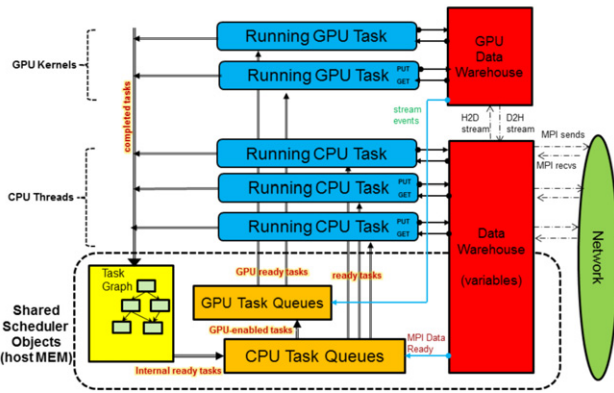
Fig. 1. Schematic of the Uintah nodal runtime system [64].

3.6. Uintah

Uintah consists of a set of parallel software components and libraries in a framework that can integrate multiple simulation components, analyze the dependencies and communication patterns between them, and efficiently execute the resulting multi-physics simulation. Uintah uses task-graphs in a similar way to Charm++ [50], but with its own unique algorithms. For example Uintah uses a "data warehouse" through which all data transfer takes place, and which ensures that the user's code is independent of the communications layer. Uintah's task-graph structure of the computation makes it possible to improve scalability through adaptive self-tuning without necessitating changes to the task graph specifications themselves. This task-graph is used to map processes onto processors and to make sure that the appropriate communication mechanisms are in place.

Uintah has been used increasingly widely since 2008, and a concerted effort to improve its scalability has been undertaken [66] by building upon the visionary design of Parker [77]. Particular advances made in Uintah are advanced scalable AMR [59,60,57] coupled to challenging multiphysics problems [58,11], and a load balancing data assimilation and feedback approach which outperforms traditional cost models. Uintah originally utilized the widely-used Berger–Rigoutsos algorithm [9] to perform regridding. However, at large core counts this algorithm does not scale [57], requiring a redesigned regridder for such cases. The regridder in Uintah defines a set of fixed-sized tiles throughout the domain. Each tile is then searched, in parallel, for refinement flags without the need for communication. All tiles that contain refinement flags become patches. This regridder is advantageous at large scale because cores only communicate once at the end of regridding when the patch sets are combined. This approach immediately led to a substantial increases in the scalability of AMR by a factor of 20× [58]. The load balancer makes use of profiling-based cost estimation methods that utilize time series analysis. These methods provide highly accurate cost estimations that automatically adjust to the changing simulation based upon a novel feedback mechanism [58].

A key factor in improving performance is the reduction in wait time through the dynamic and even out-of-order execution of task-graphs [66,11]. Uintah reduces its memory footprint through the use of a nodal shared memory model in which there is one MPI rank and one global memory (a Uintah data warehouse) per multicore node, with a thread-based runtime system used to exploit all the cores on the node [63]. The task-based runtime system is designed around the premise of asynchronous task execution and dynamic task management and includes (see Fig. 1) the following features. Each CPU core and GPU accelerator uses decentralized execution [66] and requests its own work from task queues. A Shared Memory Abstraction through Uintah's data warehouse is used to achieve lock-free execution by making use of atomic operations supported by modern CPUs so as to allow scheduling of tasks to not only CPUs but also to multiple GPUs per nodes. Support is provided for multiple accelerators per node and for ensuring that task queues are hosted by the accelerator [48,64]. The nodal runtime system that makes this possible is shown in Fig. 1. Two queues of tasks are used to organize work for CPU cores and accelerators in a dynamic way. This architecture has now also been used successfully on Intel Xeon Phi accelerators. All of these optimizations have resulted in the benchmark fluid–structure interaction. AMR application's demonstrated scalability to 500K cores and beyond on the Blue Gene Mira at the DOE's Argonne National Laboratory [65].

4. Performance challenges

Use of SAMR provides an effective compression mechanism for the solution data by keeping high resolution only where it is most needed. This data compression comes with certain costs; the management of mesh is more complex with a lot more meta-data, and good performance (scaling) is harder to achieve. The design space is large as observed from the variations found in the SAMR codes. Some of the performance challenges are inherent in using SAMR, for example even load distribution among computational resources. Some others such as memory consumption are artifacts of design choices. In this section we discuss the impact of design choices on code manageability and performance.

The dominant design difference between SAMR codes is in the way the grid hierarchies are managed. Logically the simplest to manage is the tree with clear parent–child relationship, with more flexibility proportionately increasing the complexity. The tree structure combined with the constraint of having identical number of cells in each block makes FLASH's meta-data the easiest to manage and compress. Not surprisingly, FLASH was among the first SAMR codes to eliminate redundant meta-data replication. Local tree views are easy to construct and do not consume too much memory. The disadvantage is that compression of solution data is less efficient with many more blocks being at higher resolution that they need to be. Other frameworks are more aggressive in limiting unnecessary refinement, though the refinement efficiency is usually a tunable parameter to allow greater flexibility in resource usage. The disadvantage is that devising distributed meta-data, or even compression of meta-data is more difficult. Enzo constrains its refinement by insisting on a single parent at the coarser level for each patch in the finer level, thereby allowing some simplifying assumptions in the management of meta-data. Other codes such as Chombo and BoxLib do not place such constraints in the interest of allowing maximum flexibility in tagging cells for refinement. In general, more flexibility in refinement options translates to more complex meta-data. Chombo, for example, has a mechanism for compressing the meta-data, but that imposes restrictions on the size of individual boxes [92]. Replicating meta-data makes management of the mesh cheaper in communication costs, but more expensive in memory cost. The scaling limitations of replicating meta-data are well known, though Chombo team's contention is that distributed meta-data has been a premature optimization for patch-based frameworks. This is because MPI rank count and local fast memory are growing proportionately on all vendor roadmaps. However, the straight meta-data replication is unlikely to persist in the current form in future versions of these codes. The code that has gone the farthest in overcoming this limitation is Uintah, which is also ahead of other codes in embracing newer programming models.

Achieving balanced load distribution is difficult for all of these codes not only because of AMR, but also because they support multiphysics, where different solvers have different demands on the communication and memory infrastructure. For example, if specialized physics such as evaluation of reaction networks is very

localized, some blocks will have much more computational work than others. Sometimes weighting blocks by the amount of work helps, but not if different solvers dictate conflicting weighting on the same blocks. Similarly, with subcycling finer grids do much more work, therefore appropriately weighting the blocks and distributing work becomes harder. One possible solution is to distribute the load on a per level basis on all the available processors. This achieves load balance at the possible cost of somewhat longer range communications for filling the guard cells, which is not a huge cost on moderate processor count. When there is no subcycling, weighting for load per block is easier and a global all-levels load distribution gives reasonable results. Here the disadvantage is that coarser levels do redundant work because the time step is usually dictated by the finest level. For all of these reasons performance and scaling have been major concerns for SAMR codes and all frameworks adopt some tunability for performance. For example "grid-efficiency" in patch based meshes allows for trade-off between efficiency of solution data compression and load balance. Similarly parameters like maximum block size, the frequency of regridding etc. give some tools to the users to optimize their science output within given resource constraints. A case study of this kind of optimization can be found in [33].

5. Future directions

As we anticipate the move to architectures with increasingly more cores and heterogeneous computational resources per node, both algorithms and software need to evolve. Uintah is perhaps ahead of all the other codes in exploiting newer approaches in programming abstractions. The future plans for SAMR and other multiphysics codes are trending towards removing flexibility from some parts while adding flexibility to other parts. For example a common theme among many patch based codes is to move towards a fixed block size to allow for easier distributed meta-data management. Chombo is experimenting with this approach, while Enzo is considering a transition to octree based code similar to FLASH to avoid memory fragmentation, difficulty in optimizing solvers, and difficulty in load balancing.

Another common theme is more fine-grained specification of tasks through some form of tiling (in FLASH, Chombo and BoxLib) and higher level specification of computational tasks or workflow items in the schedule (Cactus and Uintah) to improve both parallel efficiency, as well as safety by reducing programming errors. Dynamic task scheduling already exists in Uintah, others such as Chombo, Cactus and Boxlib are actively working with researchers in runtime systems to bring it into their frameworks. The use of domain specific languages is also under consideration, and is in different degrees of adoption by different packages. Cactus and Uintah already deploy some, and are looking at further expansion. Chombo uses one at C++/Fortran interface, while FLASH uses one only for configuration purposes. Code transformation and autotuning are under investigation as well, with some deployment in Cactus and Uintah.

Cactus is using *Kranc* [49,54] to convert high-level mathematical expressions and discretized operators into code (C++, CUDA, or OpenCL), automatically deriving dependency information. A DSL for Kranc called *EDL* (Equation Description Language) exists [12,82], but is not widely used yet. Uintah proposes to use the Wasatch approach proposed by Sutherland [75], in which the programmer writes pieces of code that calculate various mathematical expressions, explicitly identifying what data the code requires and produces/calculates. This code is used to create both dependency and execution graphs. Uintah will use an embedded DSL called Nebo to achieve abstraction of field operations, including application of discrete operators such as interpolants and gradients [36] in addition to the directed acyclic graph expressions that expose the dependency and flow of the calculation.

Several codes are moving to higher order methods as a part of their strategy for dealing with future architectures. For example, most Cactus-based physics components employ high-order finite difference or finite volume methods for numerical calculations, which can readily be implemented via Cactus's block-structured grid functions. This is being extended to support Discontinuous Galerkin finite element methods. Similarly, almost all applications using Chombo are now moving to 5th or higher-order accuracy. For example, a new method of Local Correction potential theory elliptical solver based on an extension of the original MLC solver described in [62], but extended to higher order and exploiting SIMD and many-core parallelism, has been implemented in Chombo. Transitioning to higher order methods is also a part of Boxlib's and FLASH's future plan.

BoxLib and Enzo developers are considering a fundamental redesign of their time-stepping algorithms, replacing traditional block-structured AMR with region-based AMR. BoxLib's region-based AMR, of which a prototype has been implemented in the BoxLib framework, replaces the concept of a "level" of data with the concept of a "region" of data. Regions, like levels, are defined by their spatial refinement; the difference is that while there is only one level at any spatial resolution, there may be multiple regions within a domain that have the same spatial resolution but different time steps. This enables more efficient time-stepping, as different physical areas of the flow may require the same spatial resolution but do not require the same time-step. In some ways this resembles a tree code, in that one can track the regions in a tree structure. The use of regions also allows more flexible job scheduling of different parts of the calculation. Enzo's plan is to take steps to go from global time-steps to semi-local time steps, and will restrict the possible time steps in such a way as to make bookkeeping more straightforward. BoxLib is also replacing communication-intensive algorithms with communication-avoiding or communication-hiding algorithms and exploring the use of resilient temporal integration strategies.

6. Summary and conclusions

The application codes and the infrastructure packages described in this survey provide a snapshot of high level frameworks utilized by multiphysics simulations when using block structured SAMR techniques. The selected set does not claim to be comprehensive; there are many more AMR-based codes and infrastructure packages that are in active use by different communities. Rather, it is representative of the different approaches, capabilities and application areas served by AMR. The codes described here share many common characteristics. They have all been in development for several years, and are publicly available (see Table 2 for the download sites). The codes also all have active user communities—their users can either contribute back to the code base for inclusion in distribution, or develop their own applications within the framework.

All the releases have core infrastructure support that provides a layer of abstraction between the physics solver capabilities and the nitty-gritty of housekeeping details such as domain decomposition, mesh management, and IO. They have interfaces to math solver libraries for providing more capabilities. Additionally, they all provide varying degrees of customizability through their frameworks. All of the codes have been used on multiple generations of high end HPC platforms and have demonstrated good performance at scale, as described in their respective sections.

The codes included in this survey share a common concern with other similarly extensive codes bases—namely, how to position themselves with regard to the future platform architectures. They are large enough that customizing them to a specific machine, let alone a class of machine architecture, is not a realistic option. Furthermore, the developers of the codes have collectively seen the advantages of optimizations that are achieved through better engineering of their frameworks over platform-specific optimizations,

Table 2
Where and how to access the different frameworks.

Release	For More Info/Where to access	How to access	Registration required?
BoxLib	ccse.lbl.gov/BoxLib ccse.lbl.gov/Downloads	git	N
Cactus	cactuscode.org/download, einsteintoolkit.org/download	svn, git	N
Chombo	commons.lbl.gov/display/chombo	svn	Y
Enzo	enzo-project.org	hg	N
FLASH	flash.uchicago.edu/site/flashcode	web	Y
Uintah	uintah.utah.edu	svn	N

which tend to have shorter life-span in usability. The codes are therefore moving towards restructuring their frameworks through more abstractions and design simplifications. They are in various stages of interfacing with the abstractions that have developed since the time that their frameworks were originally built. Though Uintah is ahead in deployment of runtime systems, other codes are not far behind since future architectures dictate the need to eliminate the bulk synchronous model that most codes currently employ. The convergence of application needs in the face of a more challenging HPC landscape and the maturation of technologies such as task-graph-based runtime, embedded DSLs and code transformation is set to transform the landscape of multiphysics application code frameworks in the next few years. The codes described in this survey, because they are critical research tools for many scientific domains, are likely to be among the first to attempt and to successfully go through this transformation.

Acknowledgments

References

[1] G. Allen, T. Goodale, F. Löffler, D. Rideout, E. Schnetter, E.L. Seidel, Component Specification in the Cactus Framework: The Cactus Configuration Language, in: Grid2010: Proceedings of the 11th IEEE/ACM International Conference on Grid Computing, 2010.

[2] G. Allen, F. Löffler, T. Radke, E. Schnetter, E. Seidel, Integrating Web 2.0 technologies with scientific simulation codes for real-time collaboration, in: Cluster Computing and Workshops, 2009. Cluster '09. IEEE International Conference, ISSN 1552-5244, 2009. pp. 1–10. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=5289130. http://dx.doi.org/10.1109/CLUSTR.2009.5289130.

[3] A.S. Almgren, V.E. Beckner, J.B. Bell, M.S. Day, L.H. Howell, C.C. Joggerst, M.J. Lijewski, A. Nonaka, M. Singer, M. Zingale, CASTRO: a new compressible astrophysical solver. I. hydrodynamics and self-gravity, Astrophys. J. 715 (2010) 1221–1238.

[4] A.S. Almgren, J.B. Bell, M. Lijewski, Z. Lukic, E.V. Andel, Nyx: a massively parallel AMR code for computational cosmology, Astrophys. J. 765 (2013) 39.

[5] AMRCLAW, 2009. http://depts.washington.edu/clawpack/users/amrclaw.

[6] E. Ateljevich, P. Colella, D. Graves, T. Ligocki, J. Percelay, P. Schwartz, Q. Shu, CFD Modeling in the San Francisco Bay and Delta, in: Proceedings of the Fourth SIAM Conference on Mathematics for Industry MI09, 2009, 2010.

[7] M. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1) (1989) 64–84.

[8] M. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (3) (1984) 484–512.

[9] M. Berger, I. Rigoutsos, An algorithm for point clustering and grid generation, IEEE Trans. Syst. Man Cybern. 21 (5) (1991) 1278–1286.

[10] M. Berzins, Status of Release of the Uintah Computational Framework, Tech. Rep. UUSCI-2012-001, Scientific Computing and Imaging Institute, 2012. http://www.sci.utah.edu/publications/SCITechReports/UUSCI-2012-001.pdf.

[11] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C. Wight, J. Peterson, Uintah - A Scalable Framework for Hazard Analysis, in: TG '10: Proc. of 2010 TeraGrid Conference, ACM, New York, NY, USA, ISBN: 978-1-60558-818-6, 2010.

[12] M. Blazewicz, I. Hinder, D.M. Koppelman, S.R. Brandt, M. Ciznicki, M. Kierzynka, F. Löffler, E. Schnetter, J. Tao, From physics model to results: an optimizing framework for cross-architecture code generation, Sci. Program 21 (1–2) (2013).

[13] BoxLib, 2011. https://ccse.lbl.gov/BoxLib.

[14] G.L. Bryan, M.L. Norman, B.W. O'Shea, T. Abel, J.H. Wise, M.J. Turk, D.R. Reynolds, D.C. Collins, P. Wang, S.W. Skillman, B. Smith, R.P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A.G. Kritsuk, E. Tasker, S. Skory, C.M. Simpson, O. Hahn, J.S. Oishi, G.C. So, F. Zhao, R. Cen, Y. Li, The Enzo Collaboration, ENZO: An adaptive mesh refinement code for astrophysics, Astrophys. J. Suppl. 211, 19. http://dx.doi.org/10.1088/0067-0049/211/2/19.

[15] Cactus developers, Cactus Computational Toolkit, 2013 http://www.cactuscode.org/.

University of Utah Institutional Repository
Author Manuscript

A. Dubey et al. / J. Parallel Distrib. Comput. ▊ (▊▊▊▊) ▊▊▊–▊▊▊

9

[16] Carpet developers, Carpet: Adaptive Mesh Refinement for the Cactus Framework, 2013. http://www.carpetcode.org/.

[17] CASC, SAMRAI Structured Adaptive Mesh Refinement Application Infrastructure (https://computation.llnl.gov/casc/SAMRAI/), Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2007.

[18] P. Colella, M. Dorr, J. Hittinger, D. Martin, High-order finite-volume methods in mapped coordinates, J. Comput. Phys. 230 (2011) 2952–2976.

[19] P. Colella, D.T. Graves, B.J. Keen, D. Modiano, A Cartesian grid embedded boundary method for hyperbolic conservation laws, J. Comput. Phys. (ISSN: 0021-9991) 211 (1) (2006) 347–366. http://www.sciencedirect.com/science/article/pii/S0021999105002780. http://dx.doi.org/10.1016/j.jcp.2005.05.026.

[20] P. Colella, D. Graves, T. Ligocki, D. Modiano, B.V. Straalen, EBC hombo software package for Cartesian grid embedded boundary applications, 2003. http://davis.lbl.gov/APDEC/WWW/apdec/designdocuments/ebchombo.pdf.

[21] P. Colella, D. Graves, T. Ligocki, D. Modiano, B.V. Straalen, EBAMRTools: EBChombo's adaptive refinement library, 2003. http://davis.lbl.gov/APDEC/old/designdocuments/ebamrtools.pdf.

[22] P. Colella, D. Graves, T. Ligocki, D. Modiano, B.V. Straalen, EBAMRGodunov, 2003. http://davis.lbl.gov/APDEC/WWW/apdec/designdocuments/ebamrhscl.pdf.

[23] P. Colella, D. Graves, N. Keen, T. Ligocki, D. Martin, P. McCorquodale, D. Modiano, P. Schwartz, T. Sternberg, B. Van Straalen, Chombo Software Package for AMR Applications Design Document, Tech. Rep., Lawrence Berkely National Laboratory, Applied Numerical Algorithms Group, Computational Research Division, 2009.

[24] S. Cornford, D. Martin, D. Graves, D. Ranken, A. LeBrocq, R. Gladstone, A. Payne, E. Ng, W. Lipscomb, Adaptive Mesh, finite-volume modeling of marine ice sheets, J. Comput. Phys. 232 (1) (2013) 529–549.

[25] R.K. Crockett, P. Colella, D.T. Graves, A Cartesian grid embedded boundary method for solving the Poisson and heat equations with discontinuous coefficients in three dimensions, J. Comput. Phys. (ISSN: 0021-9991) 230 (2011) 2451–2469. http://dx.doi.org/10.1016/j.jcp.2010.12.017.

[26] A.J. Cunningham, A. Frank, P. Varnière, S. Mitran, T.W. Jones, Simulating magnetohydrodynamical flow with constrained transport and adaptive mesh refinement: algorithms and tests of the AstroBEAR code, Astrophys. J. Suppl. Ser. 182 (2) (2009) 519. http://stacks.iop.org/0067-0049/182/i=2/a=519.

[27] C. Daley, J. Bachan, S. Couch, A. Dubey, M. Fatenejad, B. Gallagher, D. Lee, K. Weide, Adding shared memory parallelism to FLASH for many-core architectures, in: TACC-Intel Highly Parallel Computing Symposium, 2012, (submitted Feb 22, 19:27 GMT) http://www.easychair.org/conferences/submission.cgi?a=a0b8e6e0ccc3;submission=967097.

[28] M. Day, J. Bell, Numerical simulation of laminar reacting flows with complex chemistry, Combust. Theory Modell. 4 (2000) 535–556.

[29] R. Deiterding, AMROC-blockstructured adaptive mesh refinement in object-oriented C++, 2002.

[30] T. Deschamps, P. Schwartz, D. Trebotich, P. Colella, D. Saloner, R. Malladi, Vessel Segmentation and Blood Flow Simulation Using Level-Sets and Embedded Boundary Methods, in: International Congress Series, vol. 1268, 2004, pp. 75–80.

[31] M.R. Dorr, R.H. Cohen, P. Colella, M.A. Dorf, J. A.F. Hittinger, D.F. Martin, Numerical Simulation of Phase Space Advection in Gyrokinetic Models of Fusion Plasmas, in: Proceedings of the 2010 Scientific Discovery through Advanced Computing (SciDAC Conference, Chattanooga, Tennessee, July 11–15, 2010, Oak Ridge National Laboratory, 2010, pp. 42–52. http://computing.ornl.gov/workshops/scidac2010/2010_SciDAC_Proceedings.pdf.

[32] A. Dubey, K. Antypas, M. Ganapathy, L. Reid, K. Riley, D. Sheeler, A. Siegel, K. Weide, Extensible component-based architecture for FLASH a massively parallel, multiphysics simulation code, Parallel Comput. 35 (10–11) (2009) 512–522.

[33] A. Dubey, A. Calder, C. Daley, R. Fisher, C. Graziani, G. Jordan, D. Lamb, L. Reid, D.M. Townsley, K. Weide, Pragmatic optimizations for better scientific utilization of large supercomputers, Internat. J. High Perform. Comput. Appl. 27 (3) (2013) 360–373.

[34] A. Dubey, C. Daley, J. ZuHone, P. Ricker, K. Weide, C. Graziani, Imposing a Lagrangian Particle Framework on an Eulerian Hydrodynamics Infrastructure in FLASH, ApJ. Supp. 201 (2) (2012) 27. http://stacks.iop.org/0067-0049/201/27. http://dx.doi.org/10.1088/0067-0049/201/2/27.

[35] A. Dubey, L. Reid, R. Fisher, Introduction to FLASH 3.0 with application to supersonic turbulence, Phys. Scripta T132 (2008) topical Issue on Turbulent Mixing and Beyond, results of a conference at ICTP, Trieste, Italy, August 2008.

[36] C. Earl, J. Sutherland, M. Might, Nebo: A Domain-Specific Language for High-Performance Computing, Technical Report UUCS-12-032, School of Computing, University of Utah, 2013.

[37] EinsteinToolkit maintainers, Einstein Toolkit: Open software for relativistic astrophysics, 2013. http://einsteintoolkit.org/.

[38] Enzo developers, Enzo astrophysical AMR code, 2013. http://enzo-project.org/.

[39] P. MacNeice, K. Olson, C. Mobarry, R. de Fainchtein, C. Packer, PARAMESH: a parallel adaptive mesh refinement community toolkit, Comput. Phys. Commun. 126 (3) (2000) 330–354.

[40] B. Fryxell, K. Olson, P. Ricker, F.X. Timmes, M. Zingale, D.Q. Lamb, P. MacNeice, R. Rosner, J.W. Truran, H. Tufo, FLASH: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes, Astrophys. J., Suppl. 131 (2000) 273–334. http://dx.doi.org/10.1086/317361.

[41] GeoClaw, 2009. http://depts.washington.edu/clawpack/users/geoclaw.

[42] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, J. Shalf, The Cactus framework and toolkit: design and applications, in: Vector and Parallel Processing – VECPAR'2002, 5th International Conference, in: Lecture Notes in Computer Science, Springer, Berlin, 2003. http://edoc.mpg.de/3341.

[43] J.E. Guilkey, T.B. Harman, B. Banerjee, An Eulerian-Lagrangian approach for simulating explosions of energetic devices, Comput. Struct. 85 (2007) 660–674.

[44] J.E. Guilkey, T.B. Harman, A. Xia, B.A. Kashiwa, P.A. McMurtry, An Eulerian–Lagrangian approach for large deformation fluid–structure interaction problems, Part 1: Algorithm development, in: Fluid Structure Interaction II, WIT Press, Cadiz, Spain, 2003.

[45] T.B. Harman, J.E. Guilkey, B.A. Kashiwa, J. Schmidt, P.A. McMurtry, An Eulerian–Lagrangian approach for large deformation fluid–structure interaction problems, Part 1: multi-physics simulations within a modern computational framework, in: Fluid Structure Interaction II, WIT Press, Cadiz, Spain, 2003.

[46] W.D. Henshaw, D.W. Schwendeman, Parallel computation of three-dimensional flows using overlapping grids with adaptive mesh refinement, J. Comput. Phys. 227 (16) (2008) 7469–7502.

[47] R. Hornung, S. Kohn, Managing application complexity in the SAMRAI object-oriented framework, Concurrency Comput.: Pract. Exp. 14 (5) (2002) 347–368.

[48] A. Humphrey, Q. Meng, M. Berzins, T. Harman, Radiation modeling using the uintah heterogeneous CPU/GPU runtime system, in: Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment XSEDE 2012, ACM, 2012. http://dx.doi.org/10.1145/23357552335791.

[49] S. Husa, I. Hinder, C. Lechner, Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations, Comput. Phys. Commun. 174 (2006) 983–1004.

[50] L.V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, G. Zheng, Programming petascale applications with Charm++ and AMPI, Petascale Comput.: Algorithms Appl. 1 (2007) 421–441.

[51] B. Kashiwa, E. Gaffney, Design basis for CFDLIB, Tech. Rep. LA-UR-03-1295, Los Alamos National Laboratory, 2003.

[52] B. Kashiwa, M. Lewis, T. Wilson, Fluid–Structure Interaction Modeling, Tech. Rep. LA-13111-PR, Los Alamos National Laboratory, 1996.

[53] O. Korobkin, G. Allen, S.R. Brandt, E. Bentivegna, P. Diener, J. Ge, F. Löffler, E. Schnetter, J. Tao, Runtime analysis tools for parallel scientific applications, in: Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery TG '11, ACM, New York, NY, USA, ISBN: 978-1-4503-0888-5, 2011, pp. 22:1–22:8. http://dx.doi.org/10.1145/20167412016765.

[54] Kranc, Kranc: Kranc Assembles Numerical Code, 2013. http://krasncode.org/.

[55] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B.C. Mundim, C.D. Ott, E. Schnetter, G. Allen, M. Campanelli, P. Laguna, The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics, Classical Quantum Gravity 29 (11) (2012) 115001. http://dx.doi.org/10.1088/0264-9381/29/11/115001.

[56] P.S. Li, D.F. Martin, R.I. Klein, C.F. McKee, A stable, accurate methodology for high Mach number, strong magnetic field MHD turbulence with adaptive mesh refinement: resolution and refinement studies, Astrophys. J. 745 (2) (2012) article id. 139, 13 pp.

[57] J. Luitjens, M. Berzins, Scalable parallel regridding algorithms for block-structured adaptive mesh refinement, Concurr. Comput.: Pract. Exper. 23 (13) (2011) 1522–1537.

[58] J. Luitjens, M. Berzins, Improving the Performance of Uintah: A Large-Scale Adaptive Meshing Computational Framework, in: Proc. of the 24th IEEE Int. Parallel and Distributed Processing Symposium, IPDPS10, 2010. http://www.sci.utah.edu/publications/luitjens10/Luitjens_ipdps2010.pdf.

[59] J. Luitjens, M. Berzins, T. Henderson, Parallel space-filling curve generation through sorting, Concurr. Comput.: Pract. Exper. (ISSN: 1532-0626) 19 (10) (2007) 1387–1402.

[60] J. Luitjens, B. Worthen, M. Berzins, T. Henderson, Scalable parallel amr for the Uintah multiphysics code, in: Petascale Computing Algorithms and Applications, chap, Chapman and Hall/CRC, 2007.

[61] D.J. McComas, D. Alexashov, M. Bzowski, H. Fahr, J. Heerikhuisen, V. Izmodenov, M.A. Lee, E. Möbius, N. Pogorelov, N.A. Schwadron, G.P. Zank, The heliosphere's interstellar interaction: no bow shock, Science 336 (2012) 1291. http://dx.doi.org/10.1126/science.1221054.

[62] P. McCorquodale, P. Colella, G. Balls, S. Baden, A local corrections algorithm for solving Poisson's equation in three dimensions., Commun. Appl. Math. Comput. Sci. 2 (1) (2007) 57–81.

[63] Q. Meng, M. Berzins, Scalable large-scale fluid–structure interaction solvers in the Uintah framework via hybrid task-based parallelism algorithms, Concurrency Computat.: Pract. Exper. 26 (7) (2014) 1388–1407.

[64] Q. Meng, A. Humphrey, M. Berzins, The Uintah framework: a unified heterogeneous task scheduling and runtime system, in: Digital Proceedings of The International Conference for High Performance Computing, Networking, Storage and Analysis (SC12) - WOLFHPC Workshop, ACM, 2012.

[65] Q. Meng, A. Humphrey, J. Schmidt, M. Berzins, Preliminary experiences with the uintah framework on Intel Xeon Phi and stampede, in: Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE '13), ACM, New York, NY, USA, 2013, http://dx.doi.org/10.1145/2484762.2484779. Article 48, 8 pages.

[66] Q. Meng, J. Luitjens, M. Berzins, Dynamic task scheduling for the Uintah framework, in: Many-Task Computing on Grids and Supercomputers (MTAGS), 2010 IEEE Workshop on, 15-15 Nov. 2010, pp. 1–10, http://dx.doi.org/10.1109/MTAGS.2010.5699431.

[67] Mercurial developers, Mercurial distributed version control system, 2013. http://mercurial.selenic.com/.

[68] A. Mignone, C. Zanni, P. Tzeferacos, B. van Straalen, P. Colella, G. Bodo, The PLUTO code for adaptive mesh computations in astrophysical fluid dynamics, Astrophys. J. Suppl. Ser., 198, 7. http://dx.doi.org/10.1088/0067-0049/198/1/7.

[69] A. Mignone, C. Zanni, P. Tzeferacos, B. van Straalen, P. Colella, G. Bodo, The PLUTO code for adaptive mesh computations in astrophysical fluid dynamics, Astrophys. J. Suppl. Ser. 198 (7) (2012) http://dx.doi.org/10.1088/0067-0049/198/1/7.

[70] F. Miniati, P. Colella, Block structured adaptive mesh and time refinement for hybrid, hyperbolic + N-body systems, J. Comput. Phys. 227 (1) (2007) 400–430. http://www.sciencedirect.com/science/article/pii/S0021999107003385. http://dx.doi.org/10.1016/j.jcp.2007.07.035.

[71] F. Miniati, D.F. Martin, Constrained-transport Magnetohydrodynamics with Adaptive Mesh Refinement in CHARM, Astrophys. J. Suppl. Ser. 195 (1) (2011) 5. http://stacks.iop.org/0067-0049/195/i=1/a=5.

[72] S. Molins, D. Trebotich, C.I. Steefel, C. Shen, An investigation of the effect of pore scale flow on average geochemical reaction rates using direct numerical simulation, Water Resour. Res. 48 (3) (2012).

[73] Z. Mo, A. Zhang, X. Cao, Q. Liu, X. Xu, H. An, W. Pei, S. Zhu, JASMIN: a parallel software infrastructure for scientific computing, Frontiers Comput. Sci. China (ISSN: 1673-7350) 4 (4) (2010) 480–488. http://dx.doi.org/10.1007/s11704-010-0120-5.

[74] A. Nonaka, A. Almgren, J.B. Bell, M.J. Lijewski, C.M. Malone, M. Zingale, MAESTRO: an adaptive low mach number hydrodynamics algorithm for stellar flows, Astrophys. J., Suppl. 188 (2010) 358–383.

[75] P.K. Notz, R.P. Pawlowski, J.C. Sutherland, Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE Software, ACM Trans. Math. Software 39 (1) (2012) 1:1–1:21.

[76] Overture, An Object-Oriented Toolkit for Solving Partial Differential Equations in Complex Geometry. http://www.overtureframework.org.

[77] S.G. Parker, A component-based architecture for parallel multi-physics PDE simulation, Future Generation Comput. Syst. 22 (2006) 204–216.

[78] S.G. Parker, J. Guilkey, T. Harman, A component-based parallel infrastructure for the simulation of fluid–structure interaction, Eng. Comput. 22 (2006) 277–292.

[79] G. Pau, A. Almgren, J. Bell, M. Lijewski, A parallel second-order adaptive mesh algorithm for incompressible flow in porous media, Phil. Trans. R. Soc. A 367 (2009) 4633–4654.

[80] N.V. Pogorelov, S.N. Borovikov, G.P. Zank, L.F. Burlaga, R.A. Decker, E.C. Stone, Radial velocity along the voyager 1 trajectory: the effect of solar cycle, Astrophys. J. Lett., 750, L4. http://dx.doi.org/10.1088/2041-8205/750/1/L4.

[81] N.V. Pogorelov, S.T. Suess, S.N. Borovikov, R.W. Ebert, D.J. McComas, G.P. Zank, Three-dimensional features of the outer heliosphere due to coupling between the interstellar and interplanetary magnetic fields. IV. Solar cycle model based on ulysses observations, Astrophys. J., 772, 2. http://dx.doi.org/10.1088/0004-637X/772/1/2.

[82] E. Schnetter, Performance and Optimization Abstractions for Large Scale Heterogeneous Systems in the Cactus/Chemora Framework, 2013. http://arxiv.org/abs/1308.1343.

[83] E. Schnetter, P. Diener, E.N. Dorband, M. Tiglio, A multi-block infrastructure for three-dimensional time-dependent numerical relativity, Class. Quantum Grav. 23 (2006) S553–S578. http://dx.doi.org/10.1088/0264-9381/23/16/S14.

[84] E. Schnetter, S.H. Hawley, I. Hawke, Evolutions in 3-D numerical relativity using fixed mesh refinement, Classical Quantum Gravity 21 (2004) 1465–1488. http://dx.doi.org/10.1088/0264-9381/21/6/014.

[85] E.L. Seidel, G. Allen, S.R. Brandt, F. Löffler, E. Schnetter, Simplifying complex software assembly: the component retrieval language and implementation, in: Proceedings of the 2010 TeraGrid Conference, 2010. http://dx.doi.org/10.1145/18385741838592.

[86] P.J. Smith, R. Rawat, J. Spinti, S. Kumar, S. Borodai, A. Violi, Large eddy simulation of accidental fires using massively parallel computers, in: AIAA-2003-3697, 18th AIAA Computational Fluid Dynamics Conference, 2003.

[87] J. Spinti, J. Thornock, E. Eddings, P. Smith, A. Sarofim, Heat transfer to objects in pool fires, in transport phenomena in fires, in: Transport Phenomena in Fires, WIT Press, Southampton, UK, 2008.

[88] D. Sulsky, Z. Chen, H. Schreyer, A particle method for history-dependent materials, Comput. Methods Appl. Mech. Engrg. (ISSN: 0045-7825) 118 (1–2) (1994) 179–196. http://www.sciencedirect.com/science/article/pii/0045782594901120 http://dx.doi.org/10.1016/0045-7825(94)90112-0.

[89] R. Teyssier, Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES, Astron. Astrophys. 385 (2002) 337–364. http://dx.doi.org/10.1051/0004-6361:20011817.

[90] M. Thomas, E. Schnetter, Simulation Factory: Taming application configuration and workflow on high-end resources, in: Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, 2010, pp. 369–378. http://dx.doi.org/10.1109/GRID.2010.5698010.

[91] B. van der Holst, G. Tóth, I.V. Sokolov, K.G. Powell, J.P. Holloway, E.S. Myra, Q. Stout, M.L. Adams, J.E. Morel, S. Karni, B. Fryxell, R.P. Drake, CRASH: A Block-adaptive-mesh code for radiative shock hydrodynamics implementation and verification, Astrophys. J. Suppl. Ser. 194 (2) (2011) 23. http://stacks.iop.org/0067-0049/194/i=2/a=23.

[92] B. Van Straalen, P. Colella, D.T. Graves, N. Keen, Petascale block-structured AMR applications without distributed meta-data, in: Euro-Par 2011 Parallel Processing, Springer, 2011, pp. 377–386.

[93] W. Zhang, L. Howell, A. Almgren, A. Burrows, J. Bell, CASTRO: a new compressible astrophysical solver. II. Gray radiation hydrodynamics, Astrophys. J., Suppl. 196 (2011) 20.

[94] W. Zhang, L. Howell, A. Almgren, J. Burrows, A. Dolence, J. Bell, CASTRO: a new compressible astrophysical solver. II. Gray radiation hydrodynamics, Astrophys. J., Suppl. 204 (2013) 7.

[95] M. Zilhão, F. Löffler, An Introduction to the Einstein Toolkit, IJMPA 28 (22–23) (2013).

Anshu Dubey is a member of the Applied Numerical Algorithms Group at Lawrence Berkeley National Laboratory. Before joining LBL she was the Associate Director of the Flash Center for Computational Science at the University of Chicago. She received her Ph.D. in Computer Science (1993) from Old Dominion University and B.Tech. in Electrical Engineering from Indian Institute of Technology Delhi (1985). Her research interests are in parallel algorithms, computer architecture and software engineering applicable to high performance scientific computing,

Ann Almgren is a Staff Scientist in the Computing Research Department at Lawrence Berkeley National Laboratory. She received her B.A. in physics from Harvard University, and her Ph.D. in mechanical engineering from the University of California, Berkeley. Her research interests include numerical simulation of complex multiscale physical phenomena, with a current focus on low Mach number fluid dynamics, as well as adaptive mesh refinement techniques for evolving multicore architectures.

John Bell is a Senior Staff Scientist and leader of the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory. He received his B.S. in mathematics from MIT, and his Ph.D. in mathematics from Cornell University. His research focuses on the development and analysis of numerical methods for partial differential equations arising in science and engineering. He has made contributions in the areas of finite difference methods, numerical methods for low Mach number flows, adaptive mesh refinement, interface tracking and parallel computing. He has also worked on the application of these numerical methods to problems from a broad range of fields including combustion, shock physics, seismology, flow in porous media and astrophysics.

Martin Berzins is a Professor of Computer Science in the School of Computing at the University of Utah and a member of the Scientific Computing and Imaging Institute there. Martin obtained his B.S. and Ph.D. degrees from the University of Leeds in UK, where he founded the Computational PDEs Unit and became the Research Dean for Engineering. Martin's research interests lie in algorithms and software for the parallel solution of large scale science and engineering applications.

Steven R. Brandt currently holds a position as adjunct professor of computer science, and research staff (IT consultant) at the Center for Computation & Technology (CCT) at LSU. He received his Ph.D. at the University of Illinois at Urbana Champaign in 1996. His research interests lie in computational science and high performance computing. Steven R. Brandt co-leads the Cactus development team. His recent work includes adaptive mesh refinement, and GPU acceleration.
     Home page: https://www.cct.lsu.edu/~sbrandt/.

Greg Bryan is an Associate Professor in the Department of Astronomy at Columbia University. He received his B.Sc. in physics from the University of Calgary (Canada) and his Ph.D. from the University of Illinois. His interest is in theoretical and computational astrophysics —in particular, computational structure formation. He is the original creator of the Enzo code (http://enzo-project.org) and is currently one of the lead developers.

University of Utah Institutional Repository
Author Manuscript

A. Dubey et al. / J. Parallel Distrib. Comput. ▮ (▮▮▮▮) ▮▮▮–▮▮▮

11

Phil Colella is a Senior Scientist and leads the Applied Numerical Algorithms Group in the Computational Research Division at the Lawrence Berkeley National Laboratory, and a Professor in Residence in the Electrical Engineering and Computer Science Department at UC Berkeley. Received his A.B. (1974), M.A. (1976) and Ph.D. (1979) degrees from the University of California at Berkeley, all in applied mathematics. He has developed high-resolution and adaptive numerical algorithms for partial differential equations and numerical simulation capabilities for a variety of applications in science and engineering. He has also participated in the design of high-performance software infrastructure for scientific computing, including software libraries, frameworks, and programming languages.

Daniel Graves is a member of the Applied Numerical Algorithms Group at Lawrence Berkeley National Laboratory. He received his B.S. in Mechanical Engineering from University of New Hampshire and his Ph.D. in Mechanical Engineering from University of California, Berkeley. He is one of core developers of Chombo.

Michael Lijewski is a staff member of the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory and is one of the principal architects and developers of the BoxLib software framework. He received his B.S. in Mathematics from Illinois Wesleyan University, and his M.S. in Applied Mathematics from the University of Maryland, College Park. His areas of expertise include algorithm and software design for current and future multicore architectures.
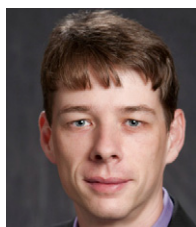
Frank Löffler currently holds a position as research staff (IT consultant) at the Center for Computation & Technology (CCT) at LSU. He received his Ph.D. at the Max-Planck-Institute Potsdam (Albert-Einstein-Institute), Germany in 2005. His research interests lie in computational science and applications in physics and chemistry. Löffler co-leads the Cactus development team at LSU and is the head maintainer of the Einstein Toolkit. His recent work includes adaptive mesh refinement improvements, simulations of single and binary neutron stars, and path integral calculations of atoms and molecules.
Home page: https://www.cct.lsu.edu/~knarf/index.html.

Brian O'Shea is an Assistant Professor of Physics and Astronomy at Michigan State University. He received his Ph.D. in physics at the University of Illinois in 2005. His research interests lie in theoretical and computational astrophysics, and he specializes in the study of galaxies and the intergalactic medium using high-dynamic-range, multiphysics cosmological simulations as well as in performance optimization of large AMR simulations of self-gravitating fluid flow. He is one of the lead developers of the Enzo astrophysics code (http://enzo-project.org).
Home page: http://www.pa.msu.edu/~osheabr/.

Erik Schnetter is Research Technologies Group Lead at the Perimeter Institute for Theoretical Physics in Waterloo, Ontario, Canada. He received his Ph.D. at the Eberhard-Karls-Universität Tübingen, Germany in 2003. His research interests lie in computational science, in using computers as tools to solve scientific and engineering problems. His recent work includes adaptive mesh refinement and multi-block methods for relativistic astrophysics simulations, software frameworks and automated code generation, as well as sustainable performance optimization for modern high-performance computing hardware architectures.
Home page: http://www.perimeterinstitute.ca/personal/eschnetter/.

Brian Van Straalen is a member of the Applied Numerical Algorithms Group at Lawrence Berkeley National Laboratory. He received his BAsc in Mechanical Engineering (1993) and his MMath in Applied Mathematics (1995) from the University of Waterloo, Canada and is a Ph.D. candidate at UC Berkeley in Computer Science. His research focuses on software engineering for high performance scientific computing.

Klaus Weide is a Research Professional at the Flash Center for Computational Science at the University of Chicago. He received Masters and doctoral degrees (1992) in physics from the University of Göttingen in Germany. He did research work on the molecular dynamics of small molecules first at the Max Planck Institute for Fluid Dynamics in Göttingen and, after 1993, at the University of Chicago. Klaus joined the Center for Thermonuclear Astrophysical Flashes at the University of Chicago in 2006 and since then has been working on extending, porting, maintaining, and supporting the Center's FLASH code.