Multi-Physics Coupling of Einstein and Hydrodynamics Evolution: A Case Study of the Einstein Toolkit

Erik Schnetter
Center for Computation & Technology and Department of Physics & Astronomy
Louisiana State University, Baton Rouge, LA, USA
schnetter@cct.lsu.edu

ABSTRACT

Cactus is a software framework for high-performance computing which sees widespread use in the numerical relativity community and other fields. The Einstein Toolkit is a set of Cactus components providing infrastructure and basic functionality for, and enabling interoperability between, different general relativistic applications codes.

In particular, the Einstein Toolkit provides an efficient coupling mechanism between spacetime (Einstein) and relativistic hydrodynamics (Euler) evolution components. This provides coupling in the volume, where both Einstein and Euler equations are integrated simultaneously everywhere in the domain on co-located grids. Several independent but interoperable Einstein and hydrodynamics codes have been built on this toolkit by different research groups over the past decade.

Below we introduce the Einstein Toolkit and describe the coupling mechanism. We discuss certain trade-offs regarding simplicity, continuity with historical coincidence, performance, and memory consumption. We briefly mention the process which lead to the current design and outline future plans. Where appropriate, we draw parallels between solving the Einstein and the Maxwell (electrodynamics) equations, in effect outlining a possible design of an equivalent toolkit for coupling the Maxwell and the hydrodynamics equations.

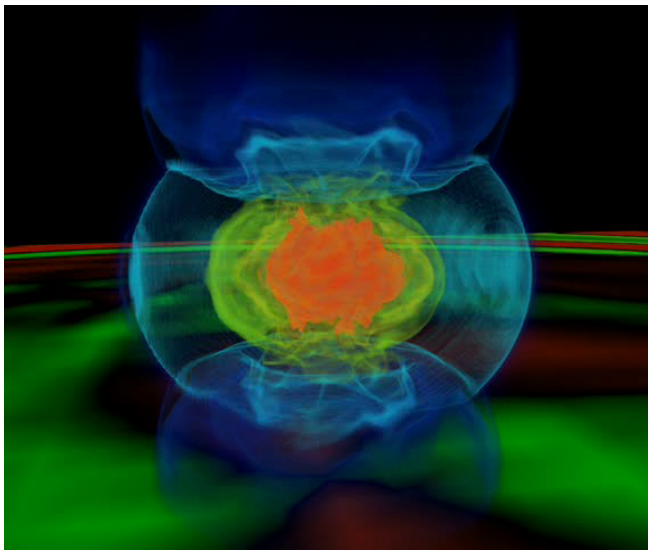

Figure 1: Rotationally deformed protoneutron star formed in the iron core collapse of an evolved massive star. Shown are volume rendering of the rest-mass density and a 2D rendition of outgoing gravitational waves. Image by R. Kähler.

1. INTRODUCTION

Multi-physics simulations are a rapidly growing field of interest, spurred by maturing single-physics simulation abilities as high-performance computing capabilities continue to increase exponentially. This is true in many fields of science and engineering, most prominently maybe in the area of fluid–structure interaction, but increasingly also in computational biology, coastal modelling, weather simulation, solid state physics, and computational astrophysics. Multi-physics can appear at various levels of interaction, ranging from simple sequential simulations to tightly coupled interaction everywhere in the simulation domain.

We present here a case study rooted in computational astrophysics. Gamma-Ray Bursts (GRBs) are intense narrowly-beamed flashes of γ-rays (very high-energy photons) of cosmological origin, and the riddle concerning their central engines and emission mechanisms is one of the most complex and challenging problems of astrophysics today [44, 25, 28]. For this, the Einstein equations and the hydrodynamics equations both need to be solved everywhere in space, as opposed to e.g. a fluid–structure interaction where there is a boundary between the fluid and the structure domains. Figure 1 shows the result of such a coupled simulation which was reported in [27], examining the collapse of a stellar iron core and the consequent formation of a protoneutron star.

To manage the complexity of a large simulation code and of the coupling mechanism we employ the Cactus framework. This allows us to separate cleanly the issues of evolution systems, public interfaces, data flow, time stepping methods, grid structures, and parallelism.

After introducing the Cactus framework in section 2, we first describe a physics framework for solving the Einstein

CBHPC 2008, October 14–17, 2008, Karlsruhe, Germany.

equations in vacuum in section 3. We describe a hydrodynamics framework in section 4, where we also discuss the coupling mechanism in detail. We conclude in section 5.

It should be noted that the Einstein and the Euler equations are only weakly coupled, in the sense that the coupling is only via lower-order terms, so that issues of stability can be examined separately. However, since the mechanism described here implements a generic tight coupling, they are not restricted to such case. The coupling mechanism could be translated to other physical systems, e.g. to coupling the Maxwell and the (non-relativistic) compressible fluid equations.

2. CACTUS FRAMEWORK

The Cactus Framework [20, 1] is an open-source, modular, portable programming environment for HPC computing. It was designed and written specifically to enable scientists and engineers to develop and perform the large-scale simulations needed for modern scientific discovery across a broad range of disciplines. Cactus is well suited for use in large, international research collaborations.

Work started on Cactus in 1997 at the Albert-Einstein-Institut in Potsdam, Germany, where it was developed with researchers at Washington University in St. Louis and at the NCSA as the core simulation code for numerical relativity for an international collaboration. An early version of this code was involved in the NASA grand challenge neutron-star modelling project and was delivered to NASA as the `GR3D` code [2]. Cactus is now used by over two dozen numerical relativity groups for their cutting edge research.

As with most frameworks, the Cactus code base is structured as a central part, called the *flesh* that provides core routines, and components, called *thorns*. The flesh is independent of all thorns and provides the main programme which parses the parameters and activates the appropriate thorns, passing control to thorns as required. By itself, the flesh performs no science; to do any computational task the user must compile in thorns and activate them at runtime.

A thorn is the basic working component within Cactus. All user-supplied code goes into thorns, which are by and large independent of each other. Thorns communicate with each other via calls to the flesh API or, more rarely, via custom APIs of other thorns. The Cactus component model is based upon tightly coupled subroutines working successively on the same data, although recent changes have broadened this to allow some element of spatial workflow.

The connection from a thorn to the flesh or to other thorns is specified in configuration files that are parsed at compile time and used to generate glue code that encapsulates the external appearance of a thorn. This glue code consists mostly of calls to registration routines in the flesh to populate databases describing the grid variables (their names, types, sizes, etc.), user-defined functions (their names, signatures, and available implementations), the execution schedule (specifying which routines need to be executed at what time of the simulation, and certain details about how they need to be called), and the control parameters (their names, types, and allowed value ranges). At runtime, the executable reads a parameter file that details which thorns are to be active and specifies values for the control parameters for these thorns. The glue code also defines certain C-style macros that are used to declare grid variables in user code and to access user-defined functions and control parameters.

Finally, there exists a special `include source` mechanism to generate include files from sections of user code, allowing inlining of code between different thorns. This mechanism arguably breaks the separation between thorns, but can lead to large performance gains if automatic, compiler-dependent cross-file inlining is not available or not reliable. The `include source` mechanism supports arbitrary languages, including Fortran, and the inlined code is surrounded by guards making it possible to enable or disable the inlined code at run time. One particular disadvantage of this mechanism is namespace pollution, similar to manually inlined code.

User thorns are generally stateless entities; they operate only on data which are passed to them. The data flow is managed by the flesh. This makes for a very robust model where thorns can be tested and validated independently, and can be combined at run-time in the manner of a functional programming language. Furthermore, thorns contain test cases for unit testing. Parallelism, communication, load balancing, memory management, and I/O are handled by a special component called *driver* which is not part of the flesh and which can be easily replaced. The flesh (and the driver) have complete knowledge about the state of the application, allowing inspection and introspection through generic APIs.

The stateless nature of thorn is a key enabling factor for tightly coupled multi-physics simulations. All data which a thorn stores has to be declared to the framework and are ultimately managed by the driver. These declarations make it possible for other thorns to access and/or modify this data as well, if appropriate permissions are given. Since all data is managed by the driver, it can be stored in a uniform manner allowing efficient access from all interested thorns. The driver "owns" the data, and passes it by reference (as subroutine arguments) for efficiency reasons. This also allows routines to be called multiple times to work on different data subsets, e.g. different grid blocks for mesh refinement. Load balancing is also performed by the driver and can thus take these access patterns into account.

One of the key concepts for thorns is the concept of an *implementation* [12], which is similar to an abstract class in Java. Relationships among thorns are all based upon relationships among the implementations they provide. In principle it should be possible to swap one thorn providing an implementation with another thorn providing that implementation, without affecting any other thorn.

An implementation defines a group of variables (and parameters) which are used to implement some functionality. For example, the thorn `MoL` provides the implementation `MethodOfLines` by implementing certain time integration algorithms. Another thorn can also implement `MethodOfLines`, and both thorns can be compiled in at the same time. At runtime, the user decides which thorn providing `MethodOfLines` should be used. No other thorn should be affected by this choice (except that different time integration algorithms would be used).

When a thorn needs access to a variable (or a parameter) provided by another thorn, it defines a relationship between itself and the other thorn's implementation, not explicitly with the other thorn: thorn *a* is said to *inherit* from implementation *B*. This allows the transparent replacement, at compile or run time, of one thorn with another thorn providing the same kind of functionality.

Each thorn declares the variables which it uses. These are private by default, but can be made publicly accessible. All

Table 1: Sample inheritance relationship between three thorns providing two implementations. Thorns $a1$ and $a2$ provide the same implementation A. Thorn b inherits from A and can thus access A's variables.

Thorn	Implementation	Inherits from	Public variables	Accessible variables
$a1$	A	—	x, y	x, y
$a2$	A	—	x, y	x, y
$b2$	B	A	z	x, y, z

thorns providing the same implementation need to define the same set of public variables. Other thorns can inherit from this implementation to gain access to these variables. Table 1 shows a small example for two implementations A and B.

It should be noted that there is no explicit definition of the implementations themselves; implementations are only defined implicitly through the thorns which implement them. This way of specifying interfaces can be called *ad hoc supertyping*, if one views thorns and implementations as part of an object oriented type system.

Thorns in Cactus can be written in C, C++, or Fortran. Storage for the thorns' variables is managed by the driver, ensuring that they can be accessed equally from all languages.

Certain thorns offer a generic numerical infrastructure. For example, thorn `MoL` provides a set of time integration algorithms, or thorn `Noise` can post-process initial conditions by adding noise to perform stability tests [39]. A thorn `Dissipation` interacts with `MoL` by adding artificial dissipation to evolved variables, i.e., filtering out high spatial frequencies. These thorns do not explicitly depend on the variables of any particular other implementation; instead, they use the flesh API to access variables of other thorns and remain thus completely generic.

The driver thorn determines also the grid structure used by the application. The simulations described in this case study all use block-structured grids, as these can be implemented very efficiently on current hardware. One Cactus driver `PUGH` offers a highly efficient uniform grid implementation, another Cactus driver `Carpet` [34, 3] offers also adaptive mesh refinement (AMR) and multi-block [32] capabilities. Both drivers are parallelised for distributed memory architectures and have been benchmarked on more than 10,000 processors.

3. EINSTEIN EQUATIONS

The *Einstein Toolkit* [4] contains a set of components dealing with the numerical implementation of the Einstein equations. This ranges from time evolution methods, initial data generators and file readers to various analysis methods. This provides a common set of basic tools for the numerical relativity community as well as standard interfaces for other, potentially non-public tools which can be built on this toolkit. The key feature is its extensibility — although several components in the Einstein Toolkit are state of the art, this is an open toolkit which can be used and extended by other groups.

In the numerical relativity community, there are currently (2008) seven major research groups employing Cactus, which corresponds to roughly half of those in the community performing three-dimensional time-dependent simulations on supercomputers. These competing groups all use the Einstein Toolkit to be able to share and collaborate on certain components, while not sharing but competing in other components.

Table 2: List of ADM variables (basic variables for the Einstein equations), and their EM (electrodynamic) counterparts for comparison. The ADM variables are used to provide a common, public interface to the state of the simulation; they are not necessarily the variables which are actually evolved in time.

ADM	Name	EM	Name
γ	metric	A	vector potential
K	(extrinsic) curvature	$\dot{A}$	
α, $\dot{\alpha}$	lapse	Φ, $\dot{\Phi}$	electric potential
β, $\dot{\beta}$	shift	—	

The central part of the Einstein Toolkit is a component *ADMBase* which contains a standard set of variables for the Einstein equations. This set of variables is called the *ADM variables*, named after Arnowitt, Deser, and Misner who introduced a variant of this set in 1962 [15]. These variables are widely agreed upon in the relativity community, and they are customarily used to analyse, exchange, or visualise data sets. Table 2 lists these variables together with their electrodynamic (EM) equivalents to which they correspond in a certain sense.

Within a simulation, the ADM variables are typically used to set up the initial configuration and to perform run-time analysis of the simulation result, such as e.g. evaluating constraints, locating horizons, or calculating the gravitational wave signature. Using the ADM variables for these tasks has the advantage that mathematical prescriptions or routines for these tasks can be exchanged between groups, such as e.g. the thorns `TwoPunctures` or `AHFinderDirect` mentioned in figures 4 and 6 below.

While these ADM variables are used to import and export data, they are not necessarily viable for evolving the system in time. The choice of evolution variables depends on many other factors, especially numerical stability and accuracy, but also depending on preferences within research groups. Different codes use different evolution systems. It is therefore important to acknowledge the difference between the variables used to exchange data and the variables which are evolved in time, since this allows both a community standard for (runtime) data interchange as well as complete flexibility for the evolution systems. Of course, for all evolution systems there have to exist conversion prescriptions from and to the ADM variables.

3.1 Time Evolution

In numerical relativity, two commonly used classes of evolution systems are the BSSN systems and the harmonic systems. BSSN systems, named after Baumgarte, Shapiro, Shibata, and Nakamura, introduce five additional variables (see e.g. [14, 13]), which can be calculated directly from the ADM variables. These additional variables greatly aid stability. Harmonic systems (see e.g. [23, 41]) may have even more variables. The particular systems used in numerical relativity are not relevant for this case study and shall here not be

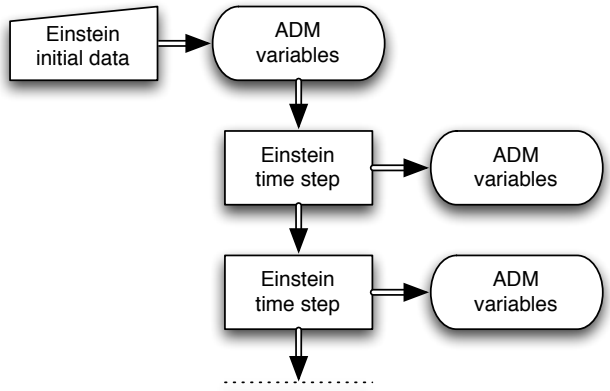

Figure 2: Data flow in and out of the ADM variables during initialisation and time evolution. Initial data are provided in the ADM variables and then converted to the evolved variables. During evolution, the ADM variables are recalculated after every time step e.g. for analysis purposes.

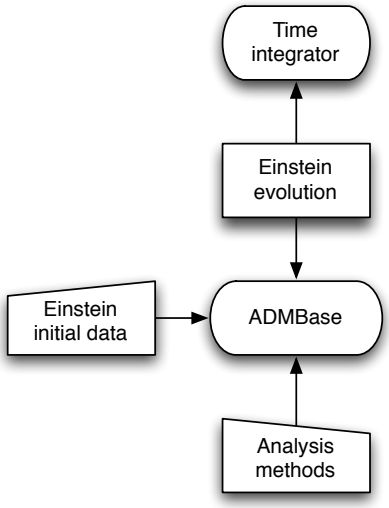

Figure 3: Schematic use of `ADMBase` in the Einstein Toolkit. Rounded boxes represent infrastructure thorns, rectangular boxes symbolise application ("user") thorns. Arrows $a \rightarrow b$ indicate that a inherits from (i.e., knows about and relies upon) b.

described further.

Time evolution proceeds as follows. The initial configuration is provided in terms of the ADM variables. From these the evolved variables are initialised. After each time step, the ADM variables are calculated from the evolved variables for analysis purposes. The evolved variables are never recalculated from the ADM variables during evolution. This scheme is depicted in figure 2.

The ADM variables and a set of schedule groups orchestrating initial data calculation are defined in a thorn *ADMBase*. All thorns calculating initial configurations are supposed to do so at a time decided by ADMBase, and provide the result in terms of ADM variables. Figure 3 depicts this interaction graphically. For efficiency purposes, ADMBase offers some flexibility for which variables are allocated, where non-allocated variables have to be treated as zero. This saves some memory, but adds some additional complexity to all thorns using ADMBase. As described below, reducing memory usage had a very high priority when ADMBase was designed, but changes in typical HPC hardware configurations have reduced this priority today.

In the example chosen here, a BSSN evolution system is implemented in a thorn `BSSN_MoL`, which is part of the CCATIE code. This thorn provides the variables containing the evolved variables (the state vector), the routines converting from and to the ADM variables, and the calculation of the right hand side (RHS) of the state vector which the time integrator requires to integrate the system in time.

A thorn `MoL` provides a standard abstraction as well as several concrete implementations for time integration. This abstraction is standard in numerical analysis; a Fortran implementation of such a standard is described and implemented e.g. in the Numerical Recipes [30]. In particular, `MoL` requires a routine calculating the RHS, which needs to be provided by the thorn containing the evolution system.

3.2 Examples

Figure 4 shows a more realistic interaction diagram between various components used in a typical Einstein code. Compared to figure 3, there are many more components; in particular, there are several thorns setting up initial data, a component modifying the initial configuration before the evolution begins, there are two different time evolution systems, a thorn filtering out high frequency components during time evolution, and there are also analysis components. Existing Einstein codes contain many more such thorns. Which of these thorns are active (used) is decided at run time by the parameter file.

The existing components were developed by different people in different research groups in geographically different locations over the past decade. The two main components which enable their seamless interaction are ADMBase, a community standard for exchanging solutions of the Einstein equations, and `MoL`, providing a standard abstraction for time integration. Table 3 lists several Einstein evolution components and their origin which rely on ADMBase and `MoL`.

3.3 Development Process

The current form of ADMBase was designed in 2000 by interested contributors in the numerical relativity community, in close collaboration with the Cactus group. Most discussions were held on public mailing lists, with several phone conferences and face-to-face meetings. Certain design decisions necessarily reflect the then-current available hardware and software infrastructure (such as typical node configurations and compilers), and perceived future research directions.

Since then, hardware and software have changed; compute nodes have more memory, and typical HPC configurations contain many more compute nodes. There is at least one good, free Fortran 90 compiler available, which has made Fortran 77 programming obsolete. In the field of numerical relativity, the main direction of research has changed twice since then, first by settling on two mainstream for-

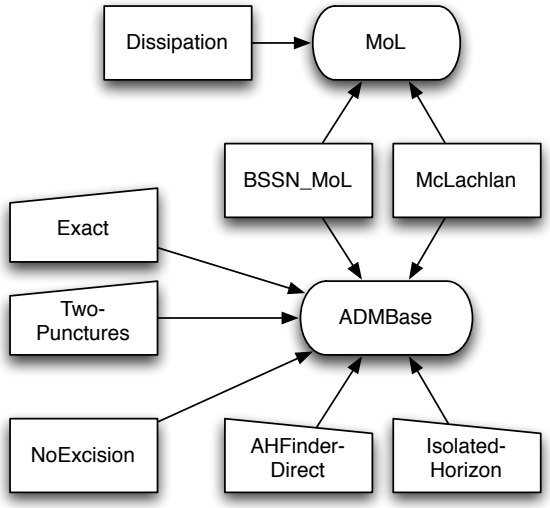

Figure 4: Example use of ADMBase in the numerical relativity code `CCATIE`. `MoL` provides the time integrator. There are several initial data (`Exact`, `TwoPunctures`), time evolution (`BSSN_MoL`, `McLachlan`), and analysis (`AHFinderDirect`, `IsolatedHorizon`) components. `NoExcision` modifies initial data before time evolution begins, and `Dissipation` modifies the evolution equations by applying a filter.

Table 3: Several production-level (having been used in refereed research publications) Einstein evolution components for Cactus and their origin. Some codes have been superseded by newer components.

Name	Group	Location
Abigel	AEI [41]	Potsdam, Germany (outdated)
ADM	AEI [4]	Potsdam, Germany (outdated)
"BSSN"	PSU [42]	State College, PA, USA
"BSSN"	RIT [47]	Rochester, NY, USA
"BSSN"	UIUC [18]	Illinois, IL, USA
CCATIE	AEI/CCT [14, 13]	Potsdam, Germany / Baton Rouge, LA, USA
"harmonic"	AEI [40]	Potsdam, Germany
LEAN	FSU [37]	Jena, Germany
Maya	PSU [38]	State College, PA, USA (outdated)
McLachlan	CCT [5]	Baton Rouge, LA, USA
TGR	TAT [31]	Tübingen, Germany (outdated)
Quilt	CCT [33]	Baton Rouge, LA, USA

Table 4: List of components of $T_{\mu\nu}$, and their approximate non-relativistic hydrodynamics counterparts for comparison. These variables are used to provide a common, public interface to the energy, momentum, and stress of the matter; these are not the hydrodynamics variables which are evolved in time.

$T_{\mu\nu}$	Hydro	Name
T_{00}	ρ, ϵ	density, internal energy
T_{0i}	ρv^i	momentum
T_{ii}	p	pressure
T_{ij}	τ_{ij}	shear stress

mulations (BSSN and harmonic) out of many, and then by break-through successes in handling black hole singularities. Both rendered some of the designed flexibility in the Einstein Toolkit unnecessary, so that certain parts of its design appear over-engineered today. (These parts play no fundamental role in the coupling mechanism.)

4. EINSTEIN–HYDRODYNAMICS COUPLING

The Einstein toolkit offers standard mechanisms to couple the evolution of the Einstein equations and of arbitrary matter fields. On one hand, the evolution of the matter fields depends on the geometry of the spacetime; on the other hand, the evolution of the geometry also depends on the matter fields which are present. A generic description of the spacetime geometry is already available via the ADM variables in ADMBase. Here we describe a somewhat analogous interface for the treatment of matter.

Matter fields influence the geometry via the *stress-energy tensor* $T_{\mu\nu}$ (pronounced tee-mu-nu) [6]. $T_{\mu\nu}$ describes the location, density, velocity, pressure, and shear stress of the matter. $T_{\mu\nu}$ plays a role somewhat analogous to, but slightly different from the ADM variables for the spacetime itself. While it is by definition always possible to calculate $T_{\mu\nu}$ from the matter fields, it is in general not possible to infer the matter fields from $T_{\mu\nu}$ alone; matter fields require in general additional variables. (These details do not matter here and are ignored in the following.) Table 4 lists the components of $T_{\mu\nu}$ and their equivalent interpretation in non-relativistic hydrodynamics.

A component *TmunuBase* contains a standard set of variables for $T_{\mu\nu}$. TmunuBase contains also a set of schedule groups orchestrating when $T_{\mu\nu}$ is to be computed. There can be several different matter fields active at the same time; if so, their contributions to $T_{\mu\nu}$ add up. Different from ADMBase, TmunuBase does not handle initialisation; as described above, this is not possible since there is no standard set of variables for all possible matter fields. (Such a standard set for the special case of compressible single-fluid flows is under development.)

4.1 Coupled Time Evolution

Time evolution of a coupled Einstein–hydrodynamics evolution proceeds as follows. The spacetime is initialised as described in section 3 above, and the hydrodynamics variables are initialised under control of the hydrodynamics evolution component. Time stepping is performed by `MoL` for both the

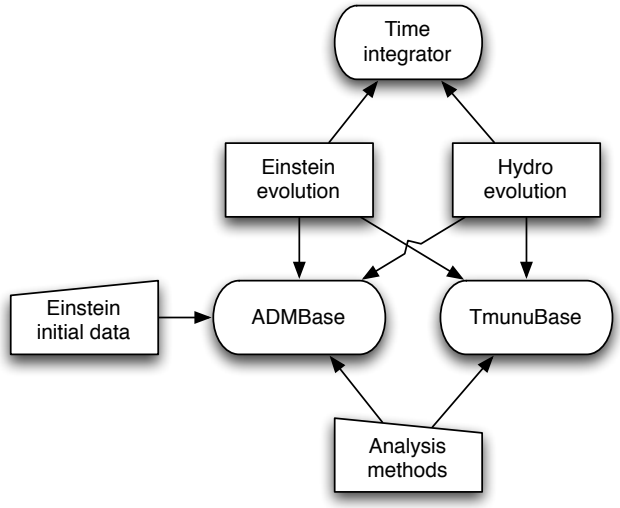

Figure 5: Schematic view of ADMBase and TmunuBase combined. ADMBase contains the Einstein interface, TmunuBase the hydro interface. Einstein and hydro evolution methods are provided by separate thorns, relying on a single, common time integrator. Both Einstein and hydro evolution methods access the respective other's variables via the public interfaces ADMBase and TmunuBase. Compare figure 3, which describes the Einstein interface in isolation.

Einstein and the hydrodynamics variables simultaneously. The right hand sides of the Einstein equations are calculated by the Einstein evolution component, accessing TmunuBase as necessary. The right hand sides of the hydrodynamics equations are calculated by the hydro evolution component, accessing ADMBase as necessary. This is schematically depicted in figure 5. (Compare to figure 3 for an illustration of the Einstein interface alone.)

Using a common time integrator for both the Einstein and the hydrodynamics equations is crucial for an efficient and accurate coupling. `MoL` was designed with this kind of coupling as one of the primary design goals; before its development, time integration used to be performed by the Einstein evolution component itself. By using a common time integrator, one automatically achieves a high order of accuracy in the coupling — for example, a fourth order accurate Runge-Kutta integrator couples Einstein and hydrodynamics equations also to fourth order. This eliminates the need for explicit coupling mechanisms which would otherwise be necessary in the individual components' time integration algorithms, e.g. by performing sequential, fractional time steps in a carefully chosen order.

4.2 Performance Trade-Offs

The Einstein toolkit offers two different mechanisms for spacetime–matter coupling with different performance trade-offs. These differences exist among other reasons because these interfaces were designed at different times, and supercomputing hardware has changed somewhat in the mean time. It used to be the case that solving the Einstein equations requires much memory compared to the amount of memory available on compute nodes, hence saving memory had a very high priority. The amount of memory per compute node has generally increased in the past years, and the emphasis has shifted to increasing CPU performance (see e.g. [45]), especially also by increasing parallel scalability (see e.g. [36, 35, 29, 7, 3]).

The hydrodynamical quantities $T_{\mu\nu}$ are required to evolve the spacetime variables in time. As described above, *TmunuBase* (the more recent interface) stores $T_{\mu\nu}$ as variables. The second interface is *CalcTmunu*, which does not require any storage of its own. Instead, the components of $T_{\mu\nu}$ are (re-)calculated for each grid point as necessary. In order to achieve good performance, CalcTmunu inlines this recalculation into the inner loops of the spacetime evolution via automatically generated include files.

Generating these include files is managed by the Cactus framework via the `include source` mechanism (see section 2 above), so that user code is only exposed to rather generic mechanism and is shielded from many low level details. A hydrodynamics thorn provides code segments for declaring local variables and for calculating the components of $T_{\mu\nu}$. These segments are combined by Cactus into include files that can be used by other thorns where $T_{\mu\nu}$ is required. Overall, this leads to an efficient mechanism with zero storage overhead. This mechanism is as elegant as is possible within the constraints imposed by Fortran 77, which was as of several years ago still one of the main languages to express equations in physics.

It goes without saying that the explicit inlining via include files has a number of disadvantages, and it is unfortunate that current compiler technology for Fortran is still not sufficiently far advanced to guarantee inlining across different source files. Among the more serious disadvantages of CalcTmunu's design are:

- It is necessary to have `if` statements in the code included into inner loops, checking which hydrodynamics thorn is active; this may prevent optimisations.
- The local variables in the included code segments poison the namespace where they are included, even if the code segments are not active.
- Different code segments are required for C, free-format Fortran, and Fortran 77. For example, the hydrodynamics code `Whisky` provides only a Fortran 77 interface, requiring Einstein thorns which use it to be (partly) written in Fortran 77 as well.
- There is no way to output $T_{\mu\nu}$ with standard Cactus mechanisms.
- It is not possible to use generic, global interpolation operators to evaluate $T_{\mu\nu}$ at arbitrary locations between grid points.

These limitations were well known when CalcTmunu was designed, but were deemed acceptable at the time given the need for performance and the available main memory.

The newer interface TmunuBase maintains full backwards compatibility with CalcTmunu by taking CalcTmunu contributions into account when $T_{\mu\nu}$ is calculated and stored in the TmunuBase variables, and likewise by contributing its $T_{\mu\nu}$ variables when CalcTmunu is used. This makes it possible to use either interface to access $T_{\mu\nu}$, independent of which

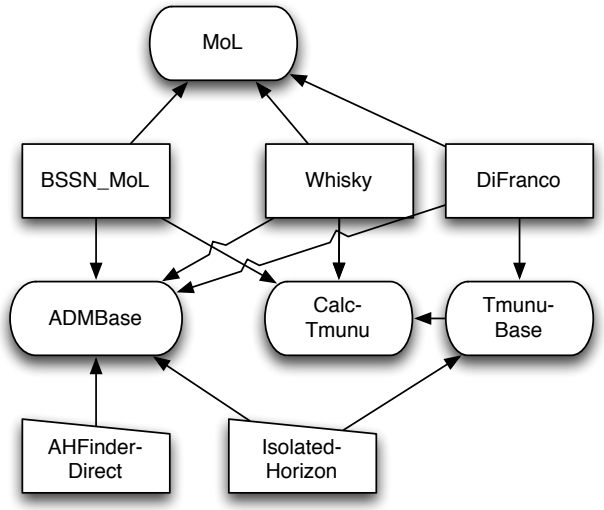

Figure 6: Example use of ADMBase with CalcTmunu and TmunuBase combined. There are two different hydrodynamics codes `Whisky` and `DiFranco`, using the matter interfaces CalcTmunu and TmunuBase, respectively. The analysis thorn `IsolatedHorizon` accesses both the ADMBase and TmunuBase variables. Note that it thus implicitly has also access to the $T_{\mu\nu}$ of `Whisky` via the compatibility interface between CalcTmunu and TmunuBase. Compare figure 4 which describes a vacuum Einstein code.

Table 5: Several Cactus-based hydrodynamics evolution components and their origin.

Name	Group	Location
DiFranco	CCT [46]	Baton Rouge, LA, USA
Pizza	TAT [22]	Tübingen, Germany
Whisky	AEI [16]	Potsdam, Germany
WhiskyMHD	AEI [19]	Potsdam, Germany
—	UIUC [18]	Illinois, IL, USA

interface is provided by a particular hydrodynamics component.

Languages like C++ could offer interfaces which are both efficient and maintain encapsulation. However, it is not possible to rely on C++ interfaces in a real-world toolkit which has to support multiple languages, large bodies of existing code, and code developers which are not proficient in C++.

4.3 Examples

Figure 6 shows a realistic interaction diagram between various components used in a typical coupled Einstein–hydrodynamics application. Compared to figure 5 there are both matter coupling mechanisms CalcTmunu and TmunuBase present, each used by a different hydrodynamics component. At most one of these hydrodynamics components would be activated at run time.

Table 5 lists several hydrodynamics evolution components and their origin. In particular, `Whisky` [17, 9] solves the compressible (relativistic) Euler equations using high-resolution shock-capturing (HRSC) methods.

Implementing a framework-based application solving the time-dependent Maxwell equations could be designed analogously to ADMBase (see table 2). The discretisation scheme (staggering of grid points, choice of derivative operators) is not prescribed by Cactus, but is under control of the component implementing the equations. Similarly, designing a coupled (non-relativistic) Maxwell–hydrodynamics evolution code could be based on the design of TmunuBase. If HRSC methods are to be used in the hydrodynamics component, then it may be necessary to calculate explicitly the characteristic speeds of the coupled system, which would partly break the encapsulation of the respective sub-systems. Whether this is necessary depends on the choice of HRSC method.

5. CONCLUSION

The Einstein Toolkit contains an efficient interface ADMBase for coupling components providing Einstein solvers, initial data generators, and analysis methods. This interface is based upon a community-agreed standard set of variables, and also orchestrates when these variables can be accessed or need to be re-calculated. ADMBase is used by more than ten large production codes in several independent research groups, which have resulted in more than 100 science publications and more than 25 student theses over the past ten years [1] in numerical relativity.

The Einstein Toolkit contains also two efficient interfaces CalcTmunu and TmunuBase for coupled Einstein–hydrodynamics simulations, based upon different performance/storage trade-offs. Several relativistic hydrodynamics codes use these interfaces to couple to existing Einstein solvers. This coupling is in the volume, so that both Einstein and hydrodynamics equations are solved at every grid point, relying on a common time integrator.

All three interfaces ADMBase, CalcTmunu, and TmunuBase are independent of the particular driver which is used to provide memory management, parallelism, mesh refinement, or a multi-block infrastructure, and work seamlessly in all these cases. They are also independent of the boundary or symmetry conditions imposed during time evolution. In particular, these interfaces do not allocate memory or handle parallelism on their own, but instead rely on Cactus flesh APIs to delegate these tasks to the driver component.

The `include source` mechanism was designed specifically for the purpose of the CalcTmunu Einstein–hydrodynamics coupling in order to overcome memory limitations of machines that were current at that time. It has since then not been used for any other major interface in Cactus-based applications. This may be due to several reasons:

- Other developers may have emphasised code clarity over performance, e.g. in educational environments such as for the Cactus CFD Toolkit [8].
- Newer HPC systems provide more main memory per node (…sometimes because they were designed with Cactus applications in mind; see e.g. http://supercomputers.aei.mpg.de/).
- More sophisticated numerical methods such as higher order finite differencing or adaptive mesh refinement require fewer grid points for the same accuracy, reducing memory requirements.

These points emphasise that software interfaces depend on performance which in turn depends on contemporary hardware properties. Interfaces and software engineering methods that are suitable today may not be so any more ten years from now.

In the future, automatic code generation using e.g. Kranc [24, 21, 10] will make it possible to couple different physics systems even more tightly. Kranc generates complete Cactus components from Mathematica equations, potentially applying optimisations or other transformations on the fly. This enables compile-time optimisations reducing the overhead of coupling while retaining a flexible and structured source code, in this case written in Mathematica. Stencil-based optimisations could automatically adapt e.g. to the total number of variables in the simulation [26, 43].

Several Einstein and hydrodynamics components have been extensively benchmarked on many HPC platforms [7], and in no case have these interfaces been found to be a performance bottleneck. ADMBase and TmunuBase continue to provide a highly efficient community standard for relativistic multi-physics simulations.

6. ACKNOWLEDGMENTS

The Einstein Toolkit and its interfaces which are described in this paper were developed and made public by many people in the numerical relativity community over a time span of more than a decade. We thank Christian D. Ott for helpful suggestions after reading an early version of the manuscript. This research employed the resources of the Center for Computation & Technology at Louisiana State University, which is supported by funding from the Louisiana legislature's Information Technology Initiative. We acknowledge support for the XiRel project [5] via the NSF PIF award 0701566, for the Alpaca project [11] via the NSF SDCI award 0721915, and for the ParCa project via a phase II NASA STTR award.

7. REFERENCES

[1] Cactus Computational Toolkit home page, http://www.cactuscode.org/.

[2] GR3D: A multi-purpose 3D code for compuational general relativistic astrophysics, http://wugrav.wustl.edu/research/codes/GR3D.

[3] Mesh refinement with Carpet, http://www.carpetcode.org/.

[4] CactusEinstein Toolkit home page, http://www.cactuscode.org/Community/NumericalRelativity/.

[5] XiRel: Next Generation Infrastructure for Numerical Relativity, http://www.cct.lsu.edu/xirel/.

[6] Stress-Energy Tensor, http://en.wikipedia.org/wiki/Stress-energy_tensor.

[7] Cactus benchmarking results, http://www.cactuscode.org/Benchmarks/.

[8] CFD Toolkit for Cactus, http://www.cactuscode.org/Community/CFDToolkit/.

[9] Whisky, EU Network GR Hydrodynamics Code, http://www.whiskycode.org/.

[10] Kranc: Automated Code Generation, http://numrel.aei.mpg.de/Research/Kranc/.

[11] Alpaca: Tools for Application-Level Profiling and Correctness Analysis, http://www.cactuscode.org/Development/alpaca/.

[12] Cactus users' guide. http://www.cactuscode.org/.

[13] M. Alcubierre, B. Brügmann, P. Diener, M. Koppitz, D. Pollney, E. Seidel, and R. Takahashi. Gauge conditions for long-term numerical black hole evolutions without excision. *Phys. Rev. D*, 67:084023, 2003.

[14] M. Alcubierre, B. Brügmann, T. Dramlitsch, J. A. Font, P. Papadopoulos, E. Seidel, N. Stergioulas, and R. Takahashi. Towards a stable numerical evolution of strongly gravitating systems in general relativity: The conformal treatments. *Phys. Rev. D*, 62:044034, 2000.

[15] R. Arnowitt, S. Deser, and C. W. Misner. The dynamics of general relativity. In L. Witten, editor, *Gravitation: An introduction to current research*, pages 227–265. John Wiley, New York, 1962.

[16] L. Baiotti, I. Hawke, P. Montero, and L. Rezzolla. A new three-dimensional general-relativistic hydrodynamics code. In R. Capuzzo-Dolcetta, editor, *Computational Astrophysics in Italy: Methods and Tools*, volume 1, page 210, Trieste, 2003. Mem. Soc. Astron. It. Suppl.

[17] L. Baiotti, I. Hawke, P. J. Montero, F. Löffler, L. Rezzolla, N. Stergioulas, J. A. Font, and E. Seidel. Three-dimensional relativistic simulations of rotating neutron star collapse to a Kerr black hole. *Phys. Rev. D*, 71:024035, 2005.

[18] J. A. Faber, T. W. Baumgarte, Z. B. Etienne, S. L. Shapiro, and K. Taniguchi. Relativistic hydrodynamics in the presence of puncture black holes. *Phys. Rev. D*, 76(10):104021–+, Nov. 2007.

[19] B. Giacomazzo and L. Rezzolla. WhiskyMHD: a new numerical code for general relativistic magnetohydrodynamics. *Class. Quantum Grav.*, 24:S235–S258, 2007.

[20] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*, Berlin, 2003. Springer.

[21] S. Husa, I. Hinder, and C. Lechner. Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations. *Comput. Phys. Comm.*, 174:983–1004, 2006.

[22] W. Kastaun. *Developing a code for general relativistic hydrodynamics with application to neutron star oscillations*. PhD thesis, Universität Tübingen, Tübingen, Germany, 2007. URN: urn:nbn:de:bsz:21-opus-28031.

[23] L. E. Kidder, M. A. Scheel, and S. A. Teukolsky. Extending the lifetime of 3D black hole computations with a new hyperbolic system of evolution equations. *Phys. Rev. D*, 64:064017, 2001.

[24] C. Lechner, D. Alic, and S. Husa. From tensor equations to numerical code — computer algebra tools for numerical relativity. In *SYNASC 2004 — 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania*, 2004. , http://arxiv.org/abs/cs.SC/0411063.

[25] P. Mészáros. Gamma-ray bursts. *Rep. Prog. Phys.*, 69:2259–2321, 2006.

[26] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale. Scientific application performance on

candidate petascale platforms. In *International Parallel & Distributed Processing Symposium (IPDPS)*, 2007.

[27] C. D. Ott, H. Dimmelmeier, A. Marek, H.-T. Janka, I. Hawke, B. Zink, and E. Schnetter. 3d collapse of rotating stellar iron cores in general relativity including deleptonization and a nuclear equation of state. *Phys. Rev. Lett.*, 98:261101, 2007.

[28] C. D. Ott, E. Schnetter, G. Allen, E. Seidel, J. Tao, and B. Zink. A case study for petascale applications in astrophysics: Simulating Gamma-Ray Bursts. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, number 18 in ACM International Conference Proceeding Series, Baton Rouge, Louisiana, 2008. ACM. http://doi.acm.org/10.1145/1341811.1341831.

[29] C. D. Ott, E. Schnetter, G. Allen, E. Seidel, J. Tao, and B. Zink. A case study for petascale applications in astrophysics: Simulating Gamma-Ray Bursts. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, number 18 in ACM International Conference Proceeding Series, Baton Rouge, Louisiana, 2008. ACM. http://doi.acm.org/10.1145/1341811.1341831.

[30] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, England, 1986.

[31] E. Schnetter. *Gauge fixing for the simulation of black hole spacetimes*. PhD thesis, Universität Tübingen, Tübingen, Germany, 2003. URN: urn:nbn:de:bsz:21-opus-8191.

[32] E. Schnetter, P. Diener, E. N. Dorband, and M. Tiglio. A multi-block infrastructure for three-dimensional time-dependent numerical relativity. *Class. Quantum Grav.*, 23:S553–S578, 2006.

[33] E. Schnetter, P. Diener, N. Dorband, and M. Tiglio. A multi-block infrastructure for three-dimensional time-dependent numerical relativity. *Class. Quantum Grav.*, 23:S553–S578, 2006.

[34] E. Schnetter, S. H. Hawley, and I. Hawke. Evolutions in 3D numerical relativity using fixed mesh refinement. *Class. Quantum Grav.*, 21(6):1465–1488, 21 March 2004.

[35] E. Schnetter, C. D. Ott, G. Allen, P. Diener, T. Goodale, T. Radke, E. Seidel, and J. Shalf. Cactus Framework: Black holes to gamma ray bursts. In D. A. Bader, editor, *Petascale Computing: Algorithms and Applications*, chapter 24. Chapman & Hall/CRC Computational Science Series, 2007.

[36] J. Shalf, E. Schnetter, G. Allen, and E. Seidel. Cactus as benchmarking platform. Technical Report CCT-TR-2006-3, Louisiana State University, 2007.

[37] U. Sperhake. Binary black-hole evolutions of excision and puncture data. *Phys. Rev. D*, 76:104015, 2007.

[38] U. Sperhake, B. Kelly, P. Laguna, K. L. Smith, and E. Schnetter. Black hole head-on collisions and gravitational waves with fixed mesh-refinement and dynamic singularity excision. *Phys. Rev. D*, 71:124042, 2005.

[39] B. Szilágyi, R. Gomez, N. T. Bishop, and J. Winicour. Cauchy boundaries in linearized gravitational theory. *Phys. Rev. D*, 62:104006, 2000.

[40] B. Szilágyi, D. Pollney, L. Rezzolla, J. Thornburg, and J. Winicour. An explicit harmonic code for black-hole evolution using excision. *Class. Quantum Grav.*, 24:S275–S293, 2007.

[41] B. Szilágyi and J. Winicour. Well-posed initial-boundary evolution in general relativity. *Phys. Rev. D*, 68:041501, 2003.

[42] B. Vaishnav, I. Hinder, F. Herrmann, and D. Shoemaker. Matched filtering of numerical relativity templates of spinning binary black holes. *Phys. Rev. D*, 76(8):084020–+, Oct. 2007.

[43] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice Boltzmann simulation optimiszation on leading multicore platforms. In *International Parallel & Distributed Processing Symposium (IPDPS)*, 2008.

[44] S. E. Woosley and J. S. Bloom. The supernova gamma-ray burst connection. *Annual Rev. Astron. Astrophys.*, 44:507–556, 2006.

[45] B. Zink. A general relativistic evolution code on CUDA architectures. Technical Report CCT-TR-2008-1, Louisiana State University, 2007.

[46] B. Zink, E. Schnetter, and M. Tiglio. Multi-patch methods in general relativistic astrophysics – I. hydrodynamical flows on fixed backgrounds. *Phys. Rev. D*, 77:103015, 2008.

[47] Y. Zlochower, J. G. Baker, M. Campanelli, and C. O. Lousto. Accurate black hole evolutions by fourth-order numerical relativity. *Phys. Rev. D*, 72:024021, 2005.