e.g., [83, 84, 85, 86, 87, 88, 89]), and non-axisymmetric instabilities in rapidly rotating polytropic NS models [84, 89, 90].

Simultanious to the advances in both our physical understanding of relativistic dynamics and the numerical techniques required to study them, a set of computational tools and libraries has been developed with the aim of providing a computational core that can enable new science, broaden the community, facilitate interdisciplinary research and take advantage of emerging petascale computers and advanced cyberinfrastructure: the Cactus computational toolkit [91]. While it was developed in large part by computer scientists, its development was driven by direct input from other fields, especially numerical relativity, and has succeeded in applying expertise in computer science directly to problems in numerical relativity.

This success prompted usage of the `Cactus` computational toolkit in other areas, such as ocean forecast models [92] and chemical reaction simulations [93]. At the same time, the growing number of results in numerical relativity increased the need for commonly available utilities such as comparison and analysis tools, typically those specifically designed for astrophysical problems. Including them within the `Cactus` computational toolkit was not felt to fit within its rapidly expanding scope. This triggered the creation of the Einstein Toolkit [94]. Large parts of the Einstein toolkit presently do make use of the `Cactus` toolkit, but this is not an requirement, and other contributions are welcome, encouraged and have been accepted in the past.

edits above this line will be carried out by myself (once Frank has sorted out conflicts).

2. Requirements

Why do we need this section? We don't even implement MHD nor neutrino transport, nor high order schemes. Isn't that something that should go into the conclusions/ outlook? The reader will be quite confused by this.

2.1. Scientific

While the aforementioned studies collectively represent breakthrough simulations that have significantly advanced the modeling of relativistic astrophysical systems, all simulations are presently missing one or more critical physical ingredients and are lacking the numerical precision to accurately and realistically model the large-scale and small-scale dynamics of their target systems simultaneously.

One of the aims of the Einstein Toolkit is to provide or extend some of these missing ingredients in the course of its development. The three most important of all possible additions, as deemed by the Einstein Toolkit members, are listed below:

the reader won't know what an ET member is

- **MHD**. Many studies, in particular those concerned with massive star collapse, NS-NS or BH-NS binaries and rotational non-axisymmetric instabilities, are still performed in pure GRHD. Without a doubt, these systems must be simulated with GRMHD to capture the effects of magnetic fields which in many cases will alter the simulation outcome on a qualitative level and may be the driving mechanisms behind much of the observable EM signature from GRBs (e.g., [95]) and magneto-rotationally exploding core-collapse supernovae (e.g., [96]). In addition, all simulations that have taken magnetic fields into account are still limited to the ideal MHD approximation, which assumes perfect

Legend:
A -- awkward; rephrase
E -- English issue
T -- tenses issue

conductivity. Non-ideal GRMHD schemes are just becoming available (see, e.g., [97, 98]), but have yet to be implemented widely in many branches of numerical relativity.

- **Equation of state (EOS), microphysics, and radiation transport**. Most presently published 3D GR(M)HD simulations, with the exception of recent work on massive star collapse (see, e.g., [80]) and binary mergers (see, e.g., [48]), relied on simple zero-temperature descriptions of ~~NS stellar structure~~, with many assuming simple polytropic forms. Such EOSs are computationally efficient, but are not necessarily a good description for matter in relativistic astrophysical systems. The inclusion of finite-temperature EOSs, derived from the microphysical descriptions of high-density matter, will lead to qualitatively different and much more astrophysically reliable results (see, e.g., [80]). In addition, most GR(M)HD studies neglect transport of neutrinos and photons and their interactions with matter. Neutrinos in particular play a crucial role in core-collapse supernovae and in the cooling of NS-NS merger remnants, thus they must not be left out when attempting to accurately model such events. Few studies have incorporated neutrino and/or photon transport and interactions in approximate ways (see, e.g., [80, 99, 48]).
- **High-order schemes and AMR**. Numerical accuracy is a central issue in long-term GR(M)HD simulations and must be addressed by a combination of (1) adaptive mesh refinement (AMR), which is used to focus grid points on regions where finer resolution is needed, and (2) high-order numerical techniques.
  Several AMR codes, including the `Carpet` driver [100] included in the Einstein Toolkit, are publicly available. An important task going forward is to facilitate the coupling of existing and future GRMHD codes with AMR to avoid under-resolving the dynamics in the systems under investigation. AMR methods are often much more complicated than uniformly distributed mesh methods, and require sophisticated algorithms to make use of massively parallel systems efficiently.
  While AMR can increase resolution near regions of interest within the computational domain, it does not increase the convergence order of the underlying numerical methods. Simulations of BHs can easily make use of high-order numerical methods, with eighth-order convergence commonly seen at present. However, most GRMHD schemes, while implementing high-resolution shock-capturing methods, are still limited to 2nd-order numerical accuracy in the hydrodynamic/MHD sector while performing curvature evolution with 4th-order accuracy or more. Higher order GRMHD schemes are in use in fixed-background simulations (e.g., [101]), but still await implementation in fully dynamical simulations.

2.2. Academic and Social

A primary concern for research groups is securing reliable funding to support graduate students and postdoctoral researchers. This goal is easier to achieve if it can be shown that science goals can be attacked directly with fewer potential infrastructure problems, one

what about reproducibility and physics benchmarking?

add ref to users list on webpage

of the goals of the Einstein Toolkit.

While the Einstein Toolkit does have a large group of users, many of them are not directly collaborating on science problems, and some can even be seen as competitors. However, all groups agree that sharing the development of the underlying infrastructure is mutually beneficial for every group and the wider community as well. This is achieved by lifting much of the otherwise necessary burden of creating such an infrastructure off the research groups' shoulders, while at the same time increasing the amount of code review, and by doing so, code quality.

this is proposal bullshit language ; please

In addition, the Einstein Toolkit provides computer scientists an ideal platform to perform state-of-the-art research, which directly benefits research in other areas of science and provides an immediate application of their research. One of the most prominent examples within the Einstein Toolkit is the `Cactus` computational toolkit, a framework developed by computer scientists and now used by researchers is many other fields.

3. Design and Strategy

The mechanisms for the development and support of the Einstein Toolkit are designed to be open, transparent and community-driven. The complete source code, documentation and tools included in the Einstein Toolkit are distributed under open-source licenses. The Einstein Toolkit maintains a version control system (`svn.einsteintoolkit.org`) with open access that contains software supported by the Einstein Toolkit, the toolkit web pages, and documentation. An open wiki for documentation (`docs.einsteintoollkit.org`) has been established where the community can contribute either anonymously or through personal authentication. Almost all discussions about the toolkit take place on an open mail list (`users@einsteintoolkit.org`). The regular weekly meetings for the Einstein Toolkit are open and the community is invited to participate. Meeting minutes are recorded and publicly available as well. The Einstein Toolkit blog requires users to first request a login, but then allows for posting at will. Any user can post comments to entries already on the blog. The community makes heavy use of an issue tracking system (`trac.einsteintoolkit.org`), with submissions also open to the public.

Despite this open design, some actions naturally have to be restricted to a smaller group of maintainers. This is true for, e.g., administrative tasks like the setup and maintenance of the services themselves, or to avoid large amounts of spam. One of the most important tasks of an Einstein Toolkit maintainer is to review and apply patches sent by users in order to ensure a high software quality level. Every substantial change or addition to the toolkit has to be reviewed by another Einstein Toolkit maintainer, and is generally open for discussion on the users mailing list. This convention, despite not being technically enforced, works well in practice, and is at the same time promoting active development.

this entire paragraph needs improvement. DO NOT expect the reader to know what you are talking about. It's great that simfactory simplifies supercomputer usage, but the reader won't know what the hell it is in the first place. Not a good idea

4. Core Technologies

The Einstein Toolkit consists of many components which, combined by the user, are utilized together to perform a full simulation. While all of these components have important tasks, a few stand out, and four are described in more detail below: the `Cactus` framework, providing the underlying infrastructure of which many other components make use, adaptive mesh refinement drivers, without which most results could not be obtained in reasonable time, the Simulation Factory, because it simplifies the necessary supercomputer usage, and Kranc, which can generate code in a computer language from a high-level description in Mathematica.

mention roots in development at the AEI and NCSA.

4.1. Cactus Framework

The `Cactus` Framework [91, 102, 103] is an open source, modular, portable programming environment for collaborative HPC computing, primarily developed at Louisiana State University. The `Cactus` computational toolkit consists of general modules providing parallel drivers, coordinates, boundary conditions, interpolators, reduction operators, and efficient I/O in different data formats. Generic interfaces are used, making it possible to use external packages and improved modules which are immediately available to its users. `Cactus` is involved in the NSF Blue Waters consortium for petascale computing, and has funding from NSF SDCI to develop new application-level tools for performance and correctness.

this is not a proposal

(the "flesh")

The structure of the `Cactus` framework is completely modular, with only a very small core providing the interfaces between modules, both at compile- and run-time. The `Cactus` modules, called "thorns", may, and typically do, specify inter-module dependencies, e.g., to share or extend configuration information, common variables, or runtime parameters. Modules compiled into an executable can remain dormant at run-time. This usage of modules and a common interface between them, enables researchers to 1) easily use modules written by others without the need to understand all details of their implementation, and 2) to write their own modules without the need to change the source code of other parts of a simulation, in the (supported) programming language of their choice. The number of active modules within a typical `Cactus` simulation ranges from tens to hundreds, often with an extensive set of inter-module dependencies.

The `Cactus` Framework was developed by the numerical relativity community, and although it is a general component framework that supports different application domains its core user group has remained from numerical relativity. The `Cactus` team has traditionally developed and supported a set of core modules for numerical relativity, as part of the `CactusEinstein` arrangement. Over the last few years however, the relevance of many of the modules has declined, and more and more of the basic infrastructure for numerical relativity has been provided by open modules distributed by research groups within the community. The Einstein Toolkit now collects the widely used parts of CactusEinstein, combined with contributions from the community.

E

I don't think such purely historical code admin information is needed. It's a waste of space and will confuse the reader who may not care about CactusEinstein.

way outdated; can't we just not say what computer it was?

4.2. Adaptive Mesh Refinement

In `Cactus`, infrastructure capabilities such as memory management, parallelization, time evolution, mesh refinement, and I/O are delegated to a set of special *driver* components. This helps separate physics code from infrastructure code; in fact, a typical physics component (implementing, e.g., the Einstein or relativistic MHD equations) does not contain any code or subroutine calls having to do with parallelization, time evolution, or mesh refinement. The information provided in the interface declarations of the individual components allows a highly efficient execution of the combined program.

The Einstein Toolkit offers two drivers, `PUGH` and `Carpet`. `PUGH` provides domains consisting of a uniform grid with Cartesian topology, and is highly scalable (up to more than 130,000 cores on a Blue Gene/P [104].) `Carpet` [105, 106, 100] provides multi-block methods and adaptive mesh refinement (AMR). Multi-block methods cover the domain with a set of (possibly distorted) blocks that exchange boundary information via techniques such as interpolation or penalty methods.‡ The AMR capabilities employ the standard Berger-Oliger algorithm [107] with subcycling in time.

AMR implies that resolution in the simulation domain is dynamically adapted to the current state of the simulation, i.e., regions that require a higher resolution are covered with blocks with a finer grid (typically by a factor of two); these are called *refined levels*. Finer grids can be also recursively refined again. At regular intervals, the resolution requirements in the simulation are re-evaluated, and the grid hierarchy is updated; this step is called *regridding*.

Since a finer grid spacing also requires smaller time steps for hyperbolic problems, the finer grids perform multiple time steps for each coarse grid time step, leading to a recursive time evolution pattern that is typical for Berger-Oliger AMR. If a simulation uses 11 levels, then the resolutions (both in space and time) of the the coarsest and finest levels differ by a factor of $2^{11-1} = 1024$. This non-uniform time stepping leads to a certain complexity that is also handled by the `Carpet` driver; for example, applying boundary conditions to a fine level requires interpolation in space and time from a coarser level. Outputting the solution at a time in between coarse grid time steps also requires interpolation. These parallel interpolation operations are implemented efficiently in `Carpet`, and are applied automatically as specified in the execution schedule, i.e. without requiring function calls in user code. Figure 1 describes some details of the Berger-Oliger time stepping algorithm. More details are described in [105].

`Carpet` is the main driver used today for `Cactus`-based astrophysical simulations. `Carpet` offers hybrid MPI/OpenMP parallelization and is used in production on up to several thousand cores. Figure 2 shows a weak scaling test of `Carpet`, where `McLachlan` (see section 5.3 below) solves the Einstein equations on a grid structure with nine levels of mesh refinement. This demonstrates excellent scalability up to more than ten thousand cores. (In production simulations, smaller and more complex grid structures and other necessary tasks

add refs.

sentence too general; get rid of parentheses

‡ Although multi-block methods are supported by `Carpet`, the Einstein Toolkit itself does not yet contain any multi-block coordinate systems.

currently

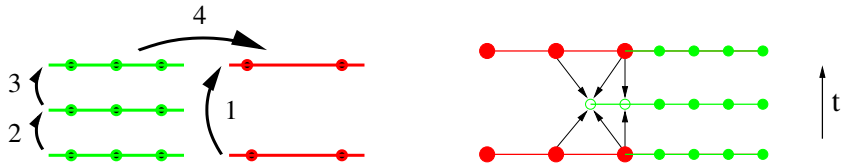

Figure 1. Berger-Oliger time stepping details, showing a coarse and a fine grid, with time advancing upwards. **Left:** Time stepping algorithm. First the coarse grid takes a large time step, then the refined grid takes two smaller steps. The fine grid solution is then injected into the coarse grid where the grids overlap. **Right:** Fine grid boundary conditions. The boundary points of the refined grids are filled via interpolation. This may require interpolation in space and in time.

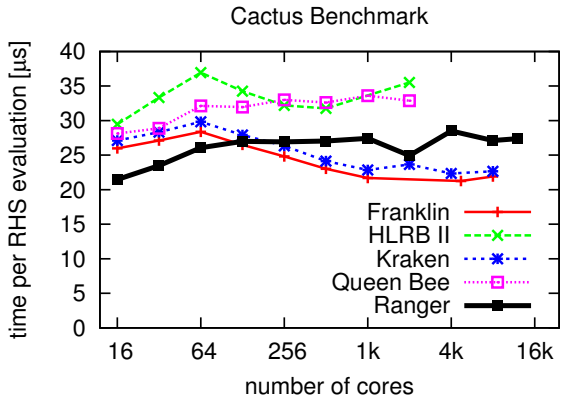

Figure 2. Results from weak scaling tests evolving the Einstein equations on a mesh refinement grid structure with nine levels. This shows the time required per grid point, where smaller numbers are better, and ideal scaling would be a horizontal line. This demonstrates excellect scalability to up to more than 10,000 cores. Including hydrodynamics approximately doubles calculation times without negatively influencing scalability.

up to

reduce scalability by ~~about a~~ factor of ten.)

We estimate that, in 2010, about 7,000 core years of computing time (45 million core hours) were used via `Carpet` by more than a dozen research groups world-wide. To date, more than 90 peer-reviewed publications and more than 15 student theses are based on `Carpet` [100].

have been

4.3. Simulation Factory

Today's supercomputers differ significantly in their hardware configuration, available software, directory structure, queueing system, queuing policy, and many other user-visible properties. In addition, the system architectures and user interfaces offered by supercomputers are very different to those offered by laptops or workstations. This makes performing large, three-dimensional time-dependent simulations a complex and time consuming task with a steep learning curve. However, most of these differences are only

superficial, and the basic capabilities of supercomputers are very similar; most of the complexity of managing simulations lies in menial tasks that require no physical or numerical insight.

complete

The Simulation Factory [108, 109] offers a set of abstractions for the tasks necessary to set up and successfully ~~finish~~ numerical simulations based on the `Cactus` framework. These abstractions hide tedious low-level management operations, they capture "best practices" of experienced users, and they create a log trail ensuring repeatable and well-documented scientific results. Using these abstractions, most operations are much simplified, many types of potentially disastrous user errors are avoided, allowing different supercomputers to be used in a uniform manner.

Using the Simulation Factory, we are able to offer a tutorial for the Einstein Toolkit [94] that lets new users download, configure, build, and run full simulations of the coupled Einstein/relativistic hydrodynamics equations on a supercomputer with a few simple commands. Users need no prior knowledge about either the details of the architecture of a supercomputer nor its particular software configuration. The same exact set of SimFactory commands can be used on all other supported supercomputers to run the same simulation there.

reformat to use less space

The Simulation Factory supports and simplifies three kinds of operations:

Remote Access The actual access commands and authentication methods differ between systems, as do the user names that a person has on different systems. Some systems are not directly accessible, and one must log in to a particular "trampoline" server first. The Simulation Factory hides this complexity.

Configuring and Building Building `Cactus` requires certain software on the system, such as compilers, libraries, and build tools. Many systems offer different versions of these, which may be installed in non-default locations. Finding a working combination that results in efficient code is extremely tedious and requires low-level system experience. The Simulation Factory provides a *machine database* that enables users to store and exchange this information. In many cases, this allows people to begin to use a new machine in a very short time with just a few, simple commands.

Submitting and Managing Simulations Many simulations run for days or weeks, requiring frequent checkpointing and job re-submission because of short queue run-time limits. Simple user errors in these menial tasks can potentially destroy weeks of information. The Simulation Factory offers safe commands that encapsulate best practices that prevent many common errors and leave a log trail.

The above features make running simulations on supercomputers much safer and simpler.

have not yet looked at text below this line.

4.4. Kranc

Kranc[110, 111, 112] is a Mathematica application which converts a high-level continuum description of a PDE into a highly optimized module for `Cactus`, suitable for running on