Title: The Einstein Toolkit

Date: Jun 22, 2011 03:00 PM

URL: http://pirsa.org/11060085

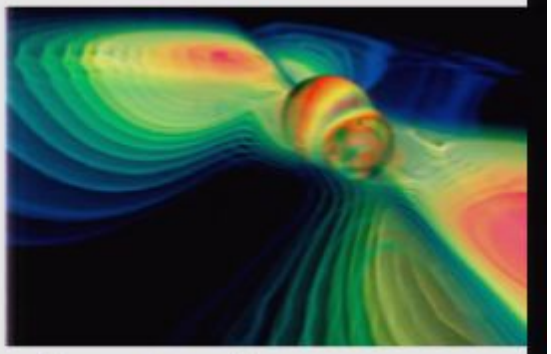
Abstract: The Einstein Toolkit Consortium is developing and supporting open software for relativistic astrophysics. Our aim is to provide the core computational tools that can enable new science, broaden the community, facilitate interdisciplinary research, and take advantage of emerging petascale computers and advanced cyberinfrastructure.
The Einstein Toolkit currently consists of an open set of over 100 components for computational relativity along with associated tools for simulation management and visualization. The toolkit includes a vacuum spacetime solver, a relativistic hydrodynamics solver, along with components for initial data, analysis, and computational infrastructure. These components have been developed and improved over many years by many different contributors.

einstein toolkit

Erik Schnetter

MICRA 2011, Waterloo, June 2011

Goals



- High-quality code base for CRA (Computational Relativistic Astrophysics)
- Extensible, open source, can be used in collaborations / by new students / for teaching
- Supports 3D simulations; efficient, parallel, runs on modern hardware

3D Components

Built-in Initial Data

BH NS BBH BNS BHNS

Initial Data Importers

EOS

Einstein Gravity (BSSN)

GR(M)HD (Valencia formulation)

Other Analysis

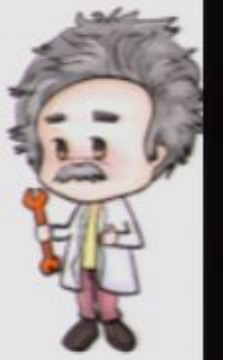
Gravitational Wave Extraction

Horizon finders

Computational Infrastructure (AMR, MPI, parallel I/O)

Community

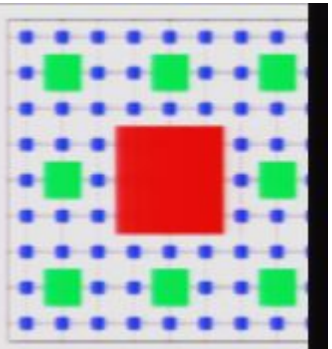
- Is a toolkit: Most users use some parts, then add their own to build their individual codes
- Managed by Einstein Toolkit Consortium (open; currently 63 members from 24 groups)
- Funded by NSF (LSU, GA Tech, RIT, Caltech, AEI)



Cactus Framework

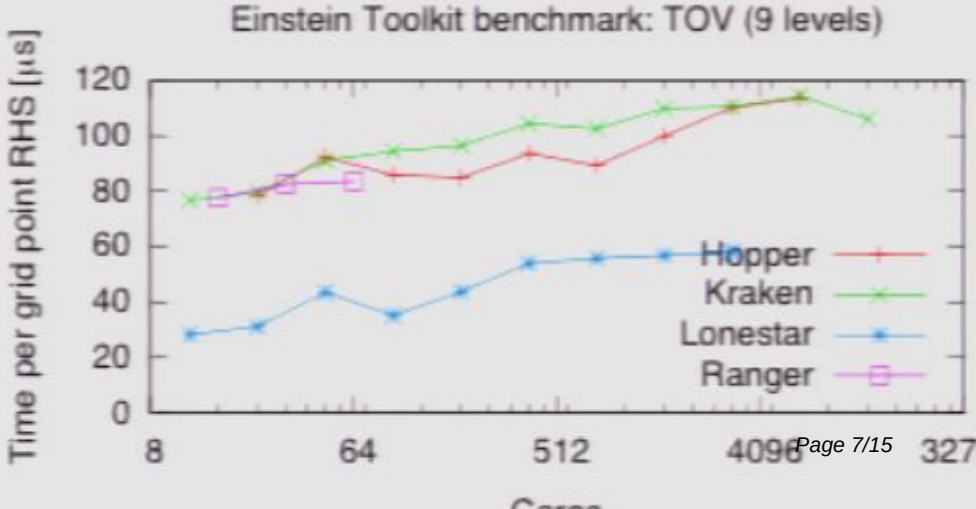
- Framework for high performance computing: support for code development, simulation control, data analysis, visualization, etc.
- Manage increased complexity of parallel programming with high level abstractions, e.g. for inter-node communication, multi-core, ...
- Active user community, 13+ years old, used by >15 research groups worldwide
- Enables collaborative development
- See http://www.cactuscode.org/

Carpet: Scalable Adaptive Mesh Refinement



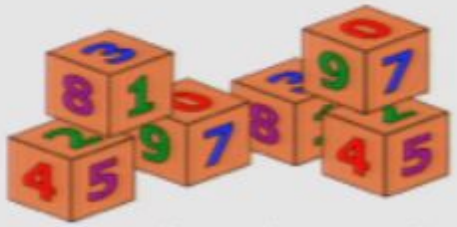
- Berger-Oliger adaptive mesh refinement (AMR) with subcycling in time
- Higher order methods require up to five ghost zones (large memory overhead)
- Hybrid parallelisation (MPI + OpenMP)

- AMR tracks physics features, refining around black holes or neutron stars
- See http://www.carpetcode.org/



Weak scaling benchmark,
9 levels of mesh refinement,
good parallel scaling

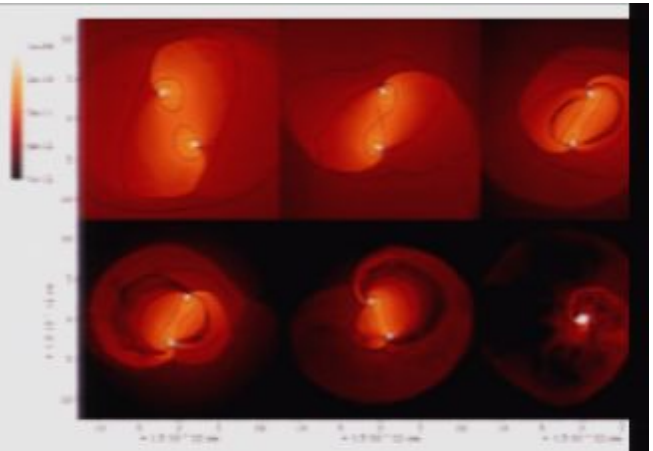
Kranc



Kranc Assembles Numerical Code

- Einstein equations (BSSN formulation) contain about 5,000 terms, very tedious to code manually
- Want to experiment with different formulations
- Compilers often do not optimise well; need to optimise explicitly (e.g. loop splitting, cache tiling)
- Some optimisations are hardware dependent (e.g. vectorisation, accelerators); don't want to repeat this manually for every new system
- See http://kranccode.org/

General Thoughts

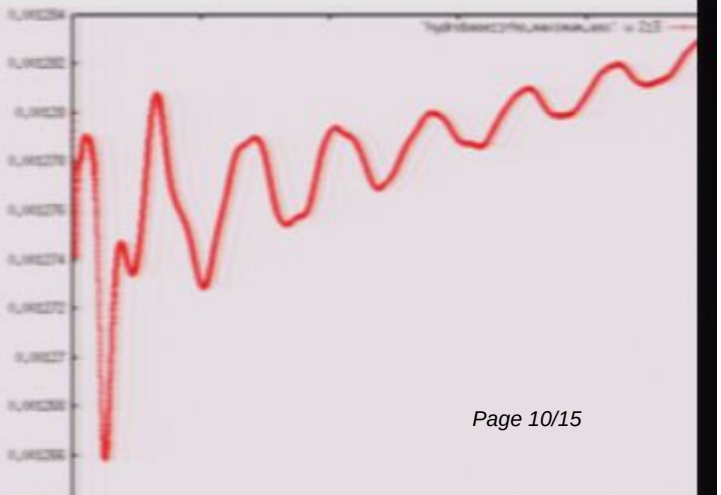


- Don't want to provide "best code"; rather want to offer pieces from which people can choose
- Should be very high quality, but not cutting edge
- Providing an open-source tool means real work; comparable to writing a paper: should be citable, should count on CV

Tutorial for New Users

http://docs.einsteintoolkit.org/et-docs/Tutorial_for_New_Users

1. Get account on Queen Bee (fill in web form)
2. Download (4 shell commands)
3. Configure (3 commands) [need user name, email address, allocation]
4. Build (1 command)
5. Run simulation (1 command)



Summary

- The Einstein Toolkit provides a code / components for a code for CRA simulations
- Open source, supported by community
- Additions/improvements are welcome!

SAGE/Cython: Speeding up Python via type annotations

Regular

```
def f(x): return sin(x**2)

def integrate_f(a,b,N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s*dx
```

```
timeit('integrate_f(0,1,1000)')
```

```
5 loops, best of 3: 4.42 s per loop
```

Annotated

```
%cython

from math import sin
def f(double x): return sin(x**2)

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s*dx
```

__Users_es...7_code_sage13_spyx.c __Users_es...ode_sage13_spy

```
timeit('integrate_f(0,1,1000)')
```

Summary

- The Einstein Toolkit provides a code / components for a code for CRA simulations
- Open source, supported by community
- Additions/improvements are welcome!

SAGE/Cython: Speeding up Python via type annotations

Regular

```
def f(x): return sin(x**2)

def integrate_f(a,b,N):
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s*dx
```

```
timeit('integrate_f(0,1,1000)')
```

```
5 loops, best of 3: 4.42 s per loop
```

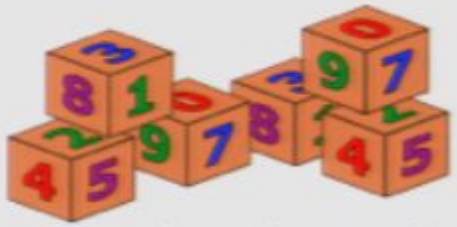
Annotated

```
%cython

from math import sin
def f(double x): return sin(x**2)

def integrate_f(double a, double b, int N):
    cdef int i
    cdef double s, dx
    s = 0
    dx = (b-a)/N
    for i in range(N):
        s += f(a+i*dx)
    return s*dx
```

__Users_es...7_code_sage13_spyx.c __Users_es...ode_sage13_spy

```
timeit('integrate_f(0,1,1000)')
```

Kranc



Kranc Assembles Numerical Code

- Einstein equations (BSSN formulation) contain about 5,000 terms, very tedious to code manually
- Want to experiment with different formulations
- Compilers often do not optimise well; need to optimise explicitly (e.g. loop splitting, cache tiling)
- Some optimisations are hardware dependent (e.g. vectorisation, accelerators); don't want to repeat this manually for every new system
- See http://kranccode.org/