The Next Generation of Astrophysical Simulations of Compact Objects

Christian Reisswig

Caltech

Einstein Symposium 2014
Motivation: Compact Objects

Astrophysical phenomena with strong dynamical gravitational fields

Compact object coalescence:
- Binary black holes,
- Binary neutron stars,
- Black hole - neutron star binaries

Black hole formation:
collapse of massive and supermassive stars

Reisswig+, Phys. Rev. D, 2009

Sources for powerful gravitational waves!

Central Engines for IGRBs

- Rapidly rotating massive evolved star
- Protoneutron star + stalled shock
- Still not clear how IGRB central engine forms and operates!
Central Engines for IGRBs

Rapidly rotating massive evolved star

Simulations of collapsars stop here

Black hole formation + hyperaccretion (Protomagnetar?)

Protoneutron star + stalled shock

~100 ms

Central Engines for IGRBs

Simulations of IGRBs start here

~100 ms

Rapidly rotating massive evolved star

Protoneutron star + stalled shock

~100 ms

Black hole formation + hyperaccretion (Protomagnetar?)

Disk + jet formation

Simulations of collapsars stop here

Simulations of IGRBs start here

(e.g. Milosavljevic+ 2012, Lindner+ 2010, Bucciantini+ 2009, Proga+ 2003, Zhang+ 2004)
Central Engines for IGRBs

Goal: Self-consistent 3D simulations of stellar collapse → disk / jet formation
Supermassive Star Collapse

Radiation pressure dominated, $10^4 < M < 10^8 M_{\text{sol}}$

Cools and contracts until onset of general relativistic collapse

Possible pathway for supermassive BH formation at $z>7$!
Supermassive Star Collapse

Cools and contracts until onset of general relativistic collapse

Depending on rotation, mass, metallicity

Formation of first supermassive black holes at $z>7$

Radiation pressure dominated, $10^4 < M < 10^8 M_{\text{sol}}$

Extremely energetic supernova explosion ($\sim 10^{55}$ erg)

(e.g. Reisswig+ 2013, Saijo+ 2009, Zink+ 2007, Shibata+ 2002)

(e.g. Chen+ 2014, Montero+ 2012, Linke+ 2001, Fuller+ 1986)
Supermassive Star Collapse

Cools and contracts until onset of general relativistic collapse

Depending on rotation, mass, metallicity

Radiation pressure dominated, $10^4 < M < 10^8 \, M_{\odot}$

Formation of first supermassive black holes at $z>7$

Extremely energetic supernova explosion ($\sim 10^{55} \text{erg}$)

EM signals visible to NASA's JWST, WFIRST, and ESA's Euclid!

(e.g. Whalen+ 2013)

GWs detectable by eLISA
Supermassive Star Collapse

Cools and contracts until onset of general relativistic collapse

Depending on rotation, mass, metallicity

Formation of first supermassive black holes at z>7

Radiation pressure dominated, $10^4 < M < 10^8 \, M_{\text{sol}}$

Extreme energetic supernova explosion ($\sim 10^{55}$ erg)

EM signals visible to NASA's JWST, WFIRST, and ESA's Euclid! (e.g. Whalen+ 2013)

Goal: Self-consistent models of collapse / explosion dynamics; predict observable signals

GWs detectable by eLISA
Multiscale Multiphysics Simulations

Core-collapse Supernovae  
Ott+ 2013, Abdikamalov+ 14

Black Hole Formation  
Reisswig+ 2013, Ott+ 2011

Binary Neutron Stars  
Rezzolla+11

Extremely computationally complex systems!  
All four forces of nature at work!

- Magnetohydrodynamics  (dynamics of fluid)
- Non-linear gravity  (neutron stars, black holes, gravitational waves)
- Complex microphysics  (Equation of state, nuclear reaction networks)
- Radiation transport  (neutrinos, photons)
Multiscale Multiphysics Simulations

Core-collapse Supernovae  Black Hole Formation  Binary Neutron Stars

Ott+ 2013, Abdikamalov+ 14  Reisswig+ 2013, Ott+ 2011  Rezzolla+11

Extremely computationally complex systems!
All four forces of nature at work!

- Multiple scales  (black holes, accretion disks / ejecta, gravitational wave-zone)
- Intrinsically Multi-D  (hydrodynamic instabilities, turbulence, rotation)
Multiscale Multiphysics Simulations

Current state-of-the-art simulations fall short in multiple ways!

Trade-offs in: Gravity,
   Radiation transport,
   Microphysical complexity,
   Dimensionality
Multiscale Multiphysics Simulations

Current state-of-the-art simulations fall short in multiple ways!

Trade-offs in: Gravity,
Radiation transport,
Microphysical complexity,
Dimensionality

Correct dynamics not captured!
Limited signal predictions!
Multiscale Multiphysics Simulations

Current state-of-the-art simulations fall short in multiple ways!

Trade-offs in: Gravity,
   Radiation transport,
   Microphysical complexity,
   Dimensionality

Correct dynamics not captured!
Limited signal predictions!

Just use larger computers??

Extremely challenging for current computer simulations!

Limited scaling
   (need to run on 100,000+ cores)
Algorithmic complexity
   (need to combine different discretization schemes)

e.g. LRZ SuperMUC: O(100,000) cores
Multiscale Multiphysics Simulations

May require different coupled discretization schemes

- Multiblock adaptive-mesh refinement (e.g. forests of oct-trees)
- Particle-in-cell methods (→ Monte-Carlo radiation transport)
- Extra grids (e.g. GW extraction, apparent horizon finding)
- Smoothed-particle hydrodynamics (for very low density material)
- Moving voronoi meshes?
Future (and current) machines achieve higher computational power via many cores!

Also: GPUs, Intel Xeon Phi

We need to distribute the computational load across many processing units!
Multiscale Multiphysics Simulations

We need to **distribute** the computational load **across** many processing units!

Distributed memory!

Internode communication: Network, e.g. via Message Passing Interface (MPI)
Multiscale Multiphysics Simulations

We need to *distribute* the computational load *across* many processing units!

Shared memory!

Intranode parallelization:  Threads
Multiscale Multiphysics Simulations

We need to **distribute** the computational load **across** many processing units!

**Ideal world**: Problem size is big / want more performance

→ just use bigger computer (more cores)

You want twice as much speed, simply use twice as many cores!

**Scaling**
Multiscale Multiphysics Simulations

Must use highly parallel algorithms!
(1,000 → 100,000+? cores)

Problems

Simulation load is data dependent and can change unpredictably during simulation (AMR, particles)

Particles can cluster

Some grids may be located only on certain processors (GW extraction, AH finding)

→ Starvation

→ We require some sophisticated load-balancing scheme!

Data exchange between processes:

→ Communication overhead / latencies
Multiscale Multiphysics Simulations

Must use highly parallel algorithms!
(1,000 → 100,000+? cores)

Problems

Simulation load is data dependent and can change unpredictably during simulation (AMR, particles)

Particles can cluster

Some grids may be located only on certain processors (GW extraction, AH finding)

→ Starvation

→ We require some sophisticated load-balancing scheme!

Data exchange between processes:

→ Communication overhead / latencies
Orchestration of Simulation

Classical “static” execution model:

- Routines are executed in a predefined order
- Interprocess communication happens synchronously

![Diagram showing the execution model with blocks A, B, C, and D, and arrows indicating starvation and latency. The diagram also includes a timeline labeled Time.]
Orchestration of Simulation

Classical “static” execution model:

- Routines are executed in a predefined order
- Interprocess communication happens asynchronously
Orchestration of Simulation

Ideal execution model:

- **Routines are executed out of order**
- **Interprocess communication happens asynchronously**

Can be achieved by **task-based parallelism**!
Orchestration of Simulation

Ideal execution model:

- Routines are executed **out of order**
- Interprocess communication happens **asynchronously**

![Diagram of execution order]

iii) Asynchronized Out-of-order

**NOTE:** Starvation and Latencies can still occur!

Need enough "tasks" to execute: **task granularity**

Higher task granularity will cause additional "bookkeeping" overhead!

**Task granularity vs bookkeeping overhead**
Task-based parallelism

- Each computational routine represents a “task”
- Each task depends on input, and defines its output
- Task can only be executed once input is “ready”

Functional programming style! (E.g. Haskell, C++ template meta-programming)
Task-based parallelism

- Each computational routine represents a “task”
- Each task depends on input, and defines its output
- Task can only be executed once input is “ready”

\[ F(x_0, \ldots, x_N) \rightarrow (y_0, \ldots, y_M) \]

NOTE: Tasks do NOT just represent mapping grid functions onto others! They are more fine grained!
Implementation
(Examples)

Uintah: Fire and explosion simulations
   AMR + particle-in-cell
   Center for the Simulation of
   Accidental Fires and Explosions (C-SAFE)

With task-based parallelism:
   Strong scaling up to 250,000 cores!

Homegrown via MPI

High Performance ParalleX (HPX): Hartmut Kaiser et. al. (LSU)
   unified programming model for parallel and distributed applications

Not a simulation code.

“Replaces MPI”: don't worry about lower level parallelization paradigms
   like threads or message passing

Hadoop / MapReduce: Google, parallel database queries
SIMsalabim
A new framework using task-based parallelism

Forests of oct-tree grids
Particle-in-cell methods

General-relativistic magnetohydrodynamics
Finite volumes / finite differences
Smoothed-particle hydrodynamics
Monte-Carlo radiation transport
Forests of oct-trees: Logic in SIMsalabim

For each octant, we have a separate task wrapping some function $F$

**Advantage:** We have plenty of tasks that can execute in parallel!

**Disadvantage:** Manually define objects and tasks for each octant separately?? INSANE!!

**Solution:** Design a **high-level driver** that provides a **high-level user interface**!

\[
F : A \rightarrow B \quad \text{High level}
\]

\[
f_0 : a_0 \rightarrow b_0 \quad \ldots \quad f_n : a_n \rightarrow b_n
\]

Autom. **defined by “octreeForest driver”**
SIMsalabim: Load-balancing and Work-stealing

General strategy: - **Load-balancing** every few steps ([hierarchical] global operation)  
- **Work-stealing** for unpredictable imbalances (can be expensive)

**Work-stealing within one MPI process:**  
Handled automatically by Intel Threading Building Blocks

**Inter-process work-stealing strategy:**

When local scheduler idles:

1) Among all available processes, pick the one with the highest load (#ready tasks) and ask for a task.
2) Victim process sends task (+ all associated input data) with highest work/data ratio (or denies/doesn't answer → retry with another process).
3) Idling process executes task and sends back result to origin.
Oct-trees and task-based parallelism

Example: WaveToy2D

Nominal octant

Octant with GZs attached (temporary object)
**Current state**

WaveToy2D tested on a few number of nodes (multi-process, multi-threading)

Simple work-stealing test over multiple nodes. Good scaling within tested range.

Next step: Put hydro + spacetime finite-volume / finite difference solver into SIMsalabim
Summary

- Simulation of compact objects are demanding: we require tremendous computational power
- Future computers are massively parallel
- Need to overcome starvation and communication latency
- Asynchronous out-of-order scheduling: Task-based parallelism
  → shown to scale to >200,000 cores