Runtime Analysis Tools for Parallel Scientific Applications

Oleg Korobkin\textsuperscript{1},
Gabrielle Allen\textsuperscript{1,2}, Eloisa Bentivegna\textsuperscript{3}, Steven Brandt\textsuperscript{1,2}, Peter Diener\textsuperscript{1,4}, Jinghua Ge\textsuperscript{1,5}, Frank Löffler\textsuperscript{1}, Erik Schnetter\textsuperscript{1,4,6}, Jian Tao\textsuperscript{1}

\textsuperscript{1}Center for Computation and Technology, Louisiana State University

\textsuperscript{2}Department of Computer Science, Louisiana State University

\textsuperscript{3}Max-Planck-Institute for Gravitational Physics, Potsdam, Germany

\textsuperscript{4}Department of Physics and Astronomy, Louisiana State University

\textsuperscript{5}Computing Services Center, Louisiana State University

\textsuperscript{6}Perimeter Institute for Theoretical Physics, Waterloo, Canada
Overview of the talk

- Introduction
- The Cactus framework
- Runtime application interface
- Use cases
- Conclusion
Challenges for designing large-scale applications

- Execution on large number of cores (> 10k).
- Assembling application from hundreds of components, developed in parts independently.
- Tuning individual components of the application with hundreds of parameters.
- Rapidly changing HPC environments.
Modern scientific applications are becoming increasingly complex.

Several abstraction levels are involved:

- mathematical equations;
- meta-code (high-level code);
- low-level code (i.e. C or Fortran code);
- individual components;
- application level.

With the new abstraction levels, new classes of errors emerge.

Tools are required to detect these errors and ensure application correctness.

Runtime analysis tools target the application level.
Why (parallel) debugger is not enough?

- Only designed to target low-level code issues.
- Is agnostic about high-level application (distributed) data structures.
- Applications compiled with debugging flag can be slower.
- Commercial licenses can limit node counts.
Cactus is

- a framework for developing portable, modular applications
- focusing, although not exclusively, on high-performance simulation codes
- designed to allow experts in different fields to develop modules based upon their experience and to use modules developed by experts in other fields with minimal knowledge of the internals or operation of the other modules
- website: http://www.cactuscode.org
Cactus applications

Cactus is used to perform large-scale parallel simulations in these areas:

- numerical relativity;
- relativistic astrophysics;
- quantum gravity;
- cosmology;
- computational fluid dynamics.
Cactus consists of two parts:

- **The Flesh**
  - The core part of Cactus
  - Independent of other parts of Cactus
  - Make system
  - Rule-based scheduling
  - Basic service library

- **The Thorns**
  - Separate libraries (modules)
  - Encapsulate the implementation of some functionality
  - Can specify dependencies on other thorns
The Driver thorn:

- Special thorn which handles distributed data structures and parallelisation.
- Only one active for a run, choose at start time.
- The only code (almost) dealing with parallelism.
- Underlying parallel layer transparent to application thorns.
- Examples:
  - PUGH: unigrid, part of the Cactus computational toolkit.
The Cactus Scheduler

- The flesh provides basic schedule bins.
- Thorns schedule their functions into these bins, using schedule rules.
- Schedule rules impose partial ordering within schedule bins.
- Thorns can create custom (nested) schedule groups.
- The flesh compiles schedule requests into a schedule tree.
- The final structure of the scheduled function calls is determined at runtime (by a parameter file).
The Cactus Framework  
Cactus scheduler

Schedule tree example

- Output grid variables
- do loop over timesteps
  - [CCTK_PREREGRID]
    - Change grid hierarchy
  - [CCTK_POSTREGRID]
    - Rotate timelevels
    - iteration = iteration+1
    - t = t+dt
  - [CCTK_PRESTEP]
- [CCTK_EVOL]

  WaveToy_Evolution  WaveToyC   Evolution of 3D wave equation
    - WaveBinaryC       WaveBinarySoi    Provide binary source during evolution (C)
    - WaveToy_Boundary   WaveToyC       Boundaries of 3D wave equation
    - GROUP WaveToy      WaveToyC       Apply boundary conditions
      - GROUP Boundary   Boundary          Execute all boundary conditions
        - Boundary_Apply    Boundary   Apply all requested local physical boundary conditions
        - CartGrid3D_Ap     CartGrid3D   Apply symmetry boundary conditions
        - Boundary_ClearS  Boundary   Unselect all grid variables for boundary conditions
  - Evolve finer grids recursively
  - Restrict from finer grids
- [CCTK_POSTRESTRICT]

Mode: local
mglevel: 0 [1], reflevel: 0 [2]
map: 0 [1], component: 0 [1]

[continue] [next function] [next iteration]
Overview of the Runtime Application Interface

Runtime interface is a set of thorns:
- HTTP server
- CLI server
- low-level socket server

Users can interact with a running simulation using a web browser or using a terminal.
An interactive session example

**HTTPS**

**CLI**

- The left image shows a web interface displaying groups and grid variables for a simulation.
- The right image illustrates a command-line interface with a graphical output.

O. Korobkin et al

Runtime Analysis Tools

July 19, 2011
The HTTP server

- Displays schedule-tree in a browser window
- Allows to pause and restart the simulation
- Allows single-stepping through individual function calls
- Pause on condition or when a warning of certain level of criticality is issued
- Ability to steer simulation parameters
- Provides basic viewport for the simulation grid functions
Simulation "home page"

Simulation: Wave propagation on a Minkowski background

This browser is connected to a Cactus simulation which contains a web server thorn. This thorn provides information and control for the simulation.

You are logged into this simulation as user Oleg Korobkin with monitoring and steering privileges.

Available options:
- Simulation Control
- Parameters
- Thorns
- Groups and Variables

Simulation:
- Parallel Cactus job using 4 MPI processes and 1 thread per process, process 0 running on host numrel09.cct.lsu.edu
- Started by user korobkin (Oleg Korobkin, 285-B Nicholson Hall, +1 225 578 1218)
- Time since start up:
  - 27 minutes
  - 57 seconds
- Estimated time per iteration
Check/Modify Parameters

Thorn CarpetIOASCII

Shown below are all parameters of thorn *CarpetIOASCII*, separated into tables of *steerable* and *fixed* parameters. Steerable parameters are those which can be modified during a simulation. To change steerable parameters, just edit the values in the form and then click the submit button. Parameters which have *not* been modified are *greyed out* and shown with their default values.

### Steerable Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out0D_criterion</td>
<td>time</td>
<td>Criterion to select 0D ASCII output intervals, overrides out_every</td>
</tr>
<tr>
<td>out0D_dir</td>
<td>scratch</td>
<td>Name of 0D ASCII output directory, overrides IO::out_dir</td>
</tr>
<tr>
<td>out0D_dt</td>
<td>20.0</td>
<td>How often to do 0D ASCII output, overrides IO::out_dt</td>
</tr>
<tr>
<td>out0D_every</td>
<td>-2</td>
<td>How often to do 0D ASCII output, overrides out_every</td>
</tr>
<tr>
<td>out0D_point_x</td>
<td>0</td>
<td>x coordinate for 0D points</td>
</tr>
<tr>
<td>out0D_point_xi</td>
<td>0</td>
<td>x-index (counting from 0) for 0D points</td>
</tr>
</tbody>
</table>
Simulation control page

Application-Level Debugging Page

This page allows you to pause your simulation and single-step through individual function calls.

Run Control

The simulation is currently paused.

The next scheduled function is **RuntimeVisit:Visit_Listen** at time bin **CCTK_POSTSTEP**.

<table>
<thead>
<tr>
<th>Control Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>continue the simulation</td>
</tr>
<tr>
<td>next function</td>
<td>single-step to the next function call</td>
</tr>
<tr>
<td>next iteration</td>
<td>single-step to the next iteration</td>
</tr>
<tr>
<td>run until</td>
<td>trigger a checkpoint after the next iteration</td>
</tr>
<tr>
<td>terminate</td>
<td>terminate the simulation</td>
</tr>
</tbody>
</table>

Pause on warnings

- Pause on warn level 0 (CCTK_WARN_ABORT)
- Pause on warn level 1 (CCTK_WARN_ALERT)
- Pause on warn level 2 (CCTK_WARN_COMPLAIN)
- Pause on warn level 3 (CCTK_WARN_PICKY)
- Pause on warn level 4 (CCTK_WARN_DEBUG)
The CLI server

- Simulation with activated CLI server opens a socket (UNIX domain or IPC socket) on a local filesystem of the root node.
- To interact with the simulation, a user needs to login to the root node.
- User can send commands to the simulation through the socket, and the simulation can return moderate amount of data back.
- On the user side, communication is implemented in Python.
- Data stream in both directions is encoded using the Base64 encoding.
CLI: sample session

```python
$ python
>>> import sim  # import sim module
>>> sim.init("/tmp/comm_sock")  # initialize module
>>> sim.pause()  # pause simulation
'Paused at CCTK_POSTSTEP in Server::Server_Work'
>>> sim.next()
'Next scheduled call at CCTK_CHECKPOINT is CarpetIOHDF5::CarpetIOHDF5_EvolutionCheckpoint'
>>> sim.CarpetIOHDF5.out_every.value
10
>>> sim.CarpetIOHDF5.out_criterion.value = "time"
>>> sim.CarpetIOHDF5.out_dt.value = 10.0
>>> sim.vars.phi.vtk_Render2D("wt2.png")
```
Module `sim` creates an object hierarchy, which reflects the high-level structures of the application, and facilitates access to them.

```
sim
  +- thorns
  |   +- ADMBase
  |   +- Carpet
  |     |   +- groups
  |     |     |   +- timing  # sample group
  |     |     |     +- timing[0] # sample variable
  |     |     |     +- timing[1] # another variable
  |     |
  |     |   +- verbose # sample parameter
  |     |     +- type # parameter type: [BRISK]
  |     |     +- range # range of the parameter
  |     |     +- desc # description
  |     |     +- value # Python 'property'
  |   +- CarpetLib
  |   +- LoopControl
< ... >
```
Module `sim` creates an object hierarchy, which reflects the high-level structures of the application, and facilitates access to them.

```python
< ... >
  | | | +- timing[0] # sample variable
  | | | +- timing[1] # another variable
  | | |
  | | +- verbose # sample parameter
  | | +- type # parameter type: [BRISK]
  | | +- range # range of the parameter
  | | +- desc # description
  | | +- value # Python 'property'
  |
  +- vars # shortcut bypassing thorns/groups
    | +- ... # grid functions
    |
  +- gh # grid hierarchy
    +- ... # information about grid functions
```
List of currently implemented commands

- `sim.init()`: connect to a communication socket and retrieve metadata
- `sim.cctk_*`: wrappers for basic functions, provided by Cactus flesh
- `sim.THORN.PARAM`: parameters of various thorns implemented as Python properties.
- `sim.vars.GRIDFUNCTION`: provides r/w access to Cactus grid functions using the Python array notation: i.e.,
  ```python
  sim.vars.phi[3,-1,:] = 0.4159265358979
  ```
- `sim.gh`: replicates information about the grid hierarchy.
- `sim.vtk_*`: the commands which are registered with CLI by the VTK visualization thorn RuntimeVTK
Thorn RuntimeVTK

- registers image rendering functions, which can be accessed at any time during the simulation;
- uses API of the VTK library;
- calls internal Cactus interpolation routines to retrieve data from other processors.
The volume rendering of a scalar wave spreading from the center. using CUDA on a GPU, while the rendering parameters are steerable at runtime via the Cactus web server.
Instabilities in multi-block MHD

- **Test:** evolve stationary uniform magnetic field on a multi-block grid with curvilinear coordinates.
- **Problem:** instability, unbounded growth of the magnetic field.
Use cases

Case 1

Invoking web interface

- ADMBase_LapseStatic
- ADMBase_ShiftStatic
- ADMBase_Static
- NaNChecker_ResetCounter

- MoL_StartLoop
  - GROUP MoL_Evolution
    - MoL
      - A single Cactus evolution step using MoL
    - MoL/SetCounter
      - Set the counter for the ODE method to loop over
    - MoL/SetTime
      - Ensure the correct time and timestep are used
  - GROUP MoL_PreStep
    - MoL_Init Bulgaria
      - MoL
        - Initialise the step size control of the RHS functions
    - GROUP MoL_CalcRHS
      - MoL PatchGRMHD
        - MultiPatchGRMHD
          - Calculate the RHS
          - Physics thorns schedule the calculation of the discrete spatial operator in here
      - MoL/PostRHS
        - MoL PatchGRMHD
          - Modify RHS functions
          - Any 'final' modifications to the RHS functions (boundaries etc.)
      - MoL/Add
        - MoL PatchGRMHD
          - Updates calculated with the efficient Runge-Kutta 3 method
      - MoL/DecrementCounter
        - MoL PatchGRMHD
          - Decrease the counter number
      - MoL/ResetTime
        - MoL PatchGRMHD
          - If necessary, change the time
            - The group for physics thorns to schedule boundary calls etc.
  - GROUP MoL_PostStep
    - GROUP ADMBase_SetADM
      - MoL PatchGRMHD
        - MoL PatchGRMHD
          - Wrapper group, do not schedule directly into this group
        - MoL/OldStyleBoundary
          - MoL PatchGRMHD
            - Place old style boundary routines here
        - MoL/OldStyle_Restore
          - MoL PatchGRMHD
            - Restore old style boundary routines here
        - MoL/Pop2Prim
          - MoL PatchGRMHD
            - Transform to magnetic conserved variables on boundary
        - MoL/ResetDeltaTime
          - MoL PatchGRMHD
            - Control the step size
      - MoL_FinishLoop
        - MoL PatchGRMHD
          - Control the flow to the next iteration
    - GROUP MoL_PseudoEvolution
      - Group PseudoEvolution
        - MoL
          - Calculate pseudo-evolved quantities
        - MoL PatchGRMHD
          - MoL PatchGRMHD
            - Set the ADM variables before this group, and use them afterwards

O. Korobkin et al

Runtime Analysis Tools

July 19, 2011
Invoking web interface

- **MoL_ResetTime**: If necessary, change the time.
- **GROUP MoL_PostStep**: The group for physics thorns to schedule body evolution, and wrapper group, do not schedule directly into this group.
  - **GROUP ADMBase_SetADMVar**: Set the ADM variables before this group, and store and change dt.
  - **GROUP MoL_OldBdry_Wrap**: Place old style boundary routines here.
  - **MoL_OldBdry_ResetDt**: Reset dt.
  - **MPGRMHD_transform_conserv**: Obtain primitive variables.
  - **MPGRMHD_con2prim_magnetos**: Attempt polytropic recovery at failed con2prim.
  - **MPGRMHD_interpolate_boundary**: Perform boundary zone interpolations.
  - **MPGRMHD_enforce_physical**: Set unphysical primitive variables to atmosphere.
  - **MPGRMHD_calc_exact_solution**: Calculate the exact solution.
  - **MPGRMHD_boundary_extrapolate**: Perform extrapolation on outer boundaries.
  - **MPGRMHD_calc_solution_error**: Calculate the solution error.
  - **MPGRMHD_transform_mag_p**: Transform to magnetic conserved variables on this group.
  - **MoL_ResetDeltaTime**: If necessary, change the timestep.
  - **MoL_FinishLoop**: Control the step size.
  - **MoL_RestoreSandR**: Restoring the Save and Restore variables.
  - **GROUP MoL_PseudoEvolution**: Calculate pseudo-evolved quantities.
  - **GROUP ADMBase_SetADMVar**: Set the ADM variables before this group, and evolve finer grids recursively.
Invoking CLI interface

Visualizing the z-component of magnetic field near the boundary

Function MPGRMHD\_boundary\_extrapolation:

before: after:

This function causes the problem. We can still save the simulation:

```python
sim.MultiPatchGRMHD.boundary_extrapolation.value = False
```
Occasional $NaN$s during large-scale simulations

Symptoms:
- happens rarely;
- difficult to reproduce;
- localized to just a few points;
- not critical for the physics.

Possible solution:
- pause simulation if NaNs have been detected;
- manually ”repair” the data;
- continue.
Correcting simulation parameter values

Symptoms:
- errors in simulation parameters;
- example: output frequency;
- resubmitting jobs not desirable due to long queue waiting times.

Solution:
- pause the simulation;
- correct wrong parameters;
- if several jobs have been submitted with wrong parameters, CLI allows to automate the parameter correction procedure.
We have shown the potential advantages for building a system of tools for interactive steering and/or debugging into the infrastructure of the Cactus component-based software framework. The system is ultimately portable in a way that regular debugging solutions are not, and has access to the intrinsic capabilities of the framework.

This work is supported by NSF award 0721915 Alpaca.
Conclusion

How to develop interactive runtime CLI for other applications:

1. Simple UNIX domain sockets on the root nodes for CLI interface.
2. Base64 encoding for control/data transfer through the socket.
3. Implement atomic commands for interacting with the simulation.
4. Use Python classes and properties on the user side to reflect the high-level data structures of the simulation and to use the atomic commands.