Cactus Framework: Scaling and Lessons Learnt

*Gabrielle Allen, **Erik Schnetter, Jian Tao***
Center for Computation & Technology
Departments of Computer Science & Physics
Louisiana State University

Also:

Christian Ott (Caltech/LSU),
Ian Hinder (PSU), Yosef Zlochower (RIT)

XiRel
Alpaca

Cactus Code

www.CactusCode.org

- Freely available, modular, portable environment for collaboratively developing parallel, high-performance multi-dimensional simulations (Component-based)
- Applications:
  - Numerical Relativity (Black holes, GRMHD)
  - Petroleum Engineering (Reservoir simulations, EnKF)
  - Coastal Modeling (Shallow water, Boussinesq)
  - CFD, Quantum Gravity, …
- Finite difference, AMR, FE/FV, multipatch
- Over 10 years of development, funded by NSF, DOE, DOD, NASA

Cactus Structure

Application Environment

Domain Specific Toolkits, Applications

Cactus Framework
(APIs, Tools)

New Paradigm

Petascale Hardware, Accelerators

Tools:

Viz
Debug
Profilers
Data

Cactus Flesh

- Written in ANSI C
- Independent of all thorns
- Contains flexible build system, parameter parsing, rule based scheduler, …
- After initialization acts as utility/service library which thorns call for information or to request some action (e.g. parameter steering)
- Contains abstracted APIs for:
  - *Parallel operations, IO and checkpointing, reduction operations, interpolation operations, timers. (APIs designed for science needs)*
- Functionality provided by (swappable) thorns

Cactus Thorns (Components)

- Written in C, C++, Fortran 77, Fortran 90, (Java, Perl, Python)
- Separate swappable libraries encapsulating some functionality (“implementations”)
- Each thorn contains configuration files which specify interface with Flesh and other thorns
- Configuration files into a set of routines which provide thorn information
  - Scheduling, variables, functions, parameters, configuration
- Configuration files have well defined language, use as basis for framework interoperability

Relativistic Astrophysics in 2010

- Frontier Astrophysics Problems
  - Full 3D GR simulations of binary systems for dozens or orbits and merger to final black hole
    - All combinations of black holes, neutron stars, and exotic objects like boson stars, quark stars, strange stars
  - Full 3D GR simulations of core collapse, supernova explosions, accretion onto NS, BH
  - Gamma-ray bursts
- All likely to be observed by LIGO in the timeframe of this facility.

Computational Needs for GRB

- Resolve from 10,000km down to 100m on a domain of 1,000,000km-cubed for 100 secs of physical time
- Assume 16,000 flops per gridpoint
- 512 grid functions
- Computationally:
  - High order (>4th) adaptive finite difference schemes
  - 16 levels of refinement
  - Several weeks with1 PFLOP/s sustained performance
  - (at least 4 PFLOP/s peak, > 100K procs)
  - 100 TB memory (size of checkpoint file needed)
  - PBytes storage for full analysis of output

Cactus Parallelism

- Scheduler calls routines and provides n-D block of data (typical set up for FD codes)
- Also information about size, boundaries, etc.
- Fortran memory layout used (appears to C as 1D array)
- Driver thorns are responsible for memory management and communication.
  - Abstracted from science modules
- Supported parallel operations
  - Ghostzone synchronization, generalized reduction, generalized interpolation.

PUGH UniGrid Driver

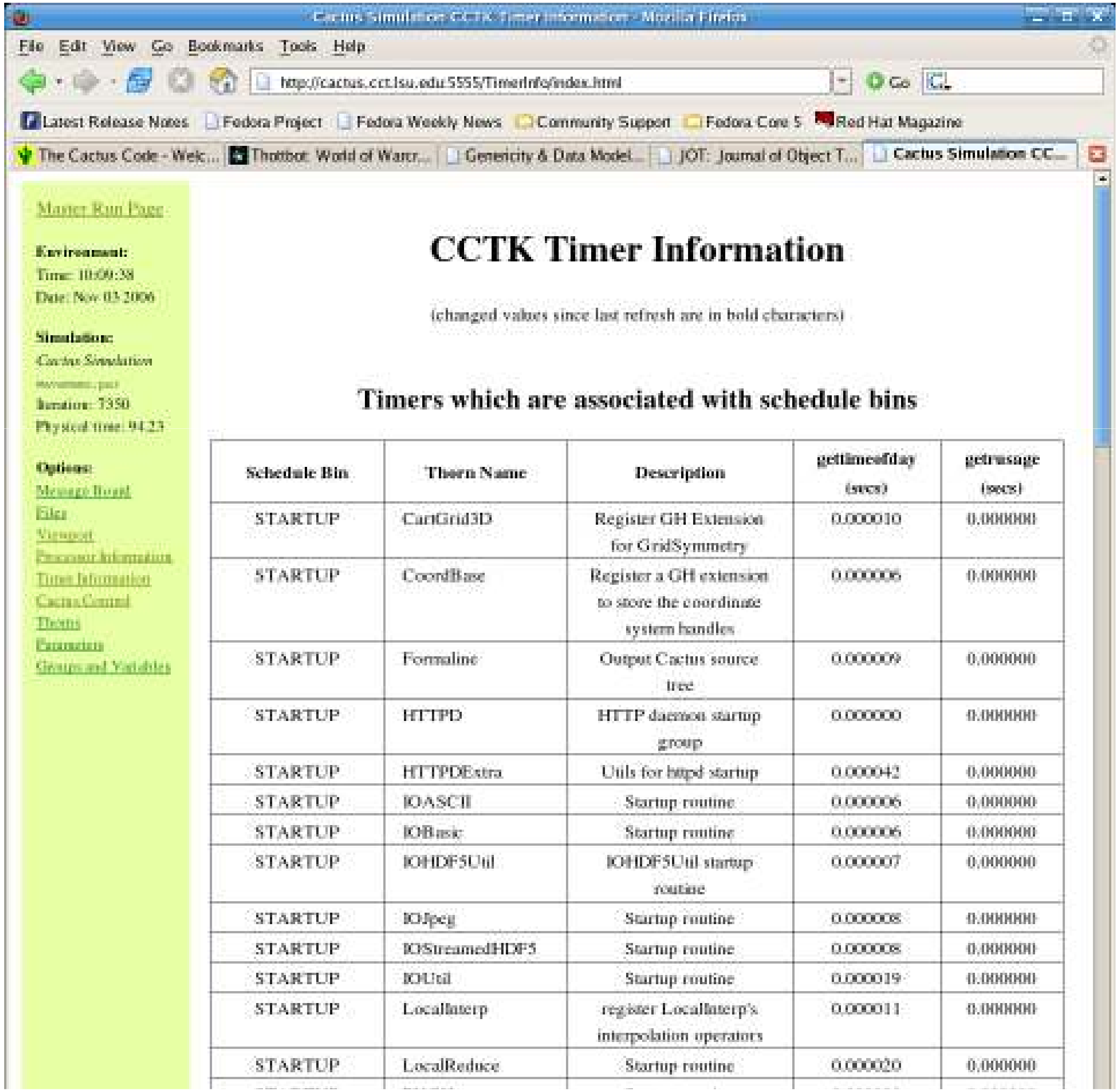
- Standard driver for science runs till a few years ago
- MPI domain decomposition
- Flexible methods for load balancing, processor topology
- Well optimized, scales very well for numerical relativity kernels (e.g. to 130K processors on BG/P)

Carpet AMR Driver

- Mesh refinement library for Cactus, written in C++ (Erik Schnetter)
- Implements (minimal) Berger Oliger algorithm, constant refinement ratio, vertex centered refinement
- MPI to decompose grids across processors, handle communications
- Now experimenting with
  - OpenMP/MPI hybrid models
  - Caching/Tiling optimizations

Profiling

- Cactus has its own timing interface (thorns, timebins, communication, user defined, …)
- Use PAPI and Tau through the Cactus timing interface
- Runtime application level profiling, debugging, correctness through Alpaca Project (Schnetter)

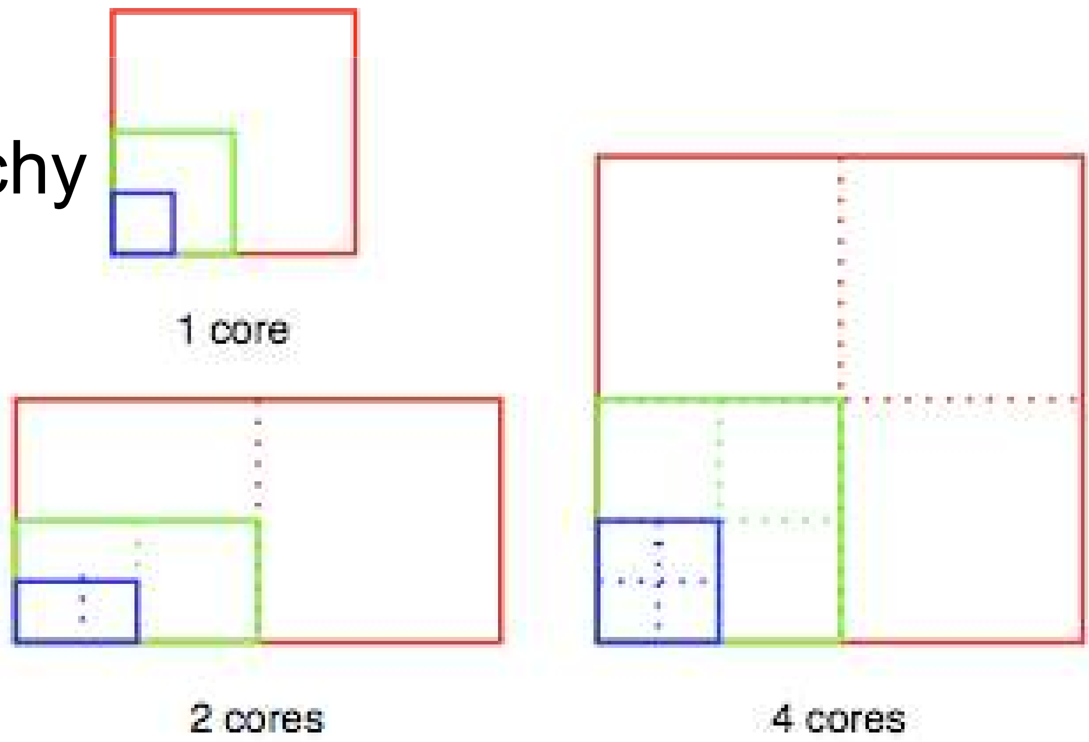


Benchmarking Strategy (XiRel)

- Define good benchmarks:
  - **Weak scaling (kernel, increasing scale of sims)**
  - Strong scaling (physics, reducing runtime)
  - I/O (checkpointing/restart of weak/strong benchmarks)
  - Benchmarks for both vacuum and matter spacetimes.
- For weak scaling we have two cases
  - Unigrid (no AMR, similar to old Cactus PUGH)
  - AMR (9 levels of refinement)
- Study scaling and performance for four different general relativity codes which all use Cactus/Carpet
  - Understand differences between codes (#grid variables, boundary treatment)

Benchmarking AMR

- AMR adapts resolution to areas needing resolving
- Hard to define a typical regridding pattern (in a short benchmark)
- Weak scaling uses constant grid hierarchy (no regridding)
- Strong scaling will use regridding



Weak Scaling BH Benchmarks

- Cartesian Minkowski spacetime as the initial data
- 4th order accurate finite differences
- 4th order accurate Runge-Kutta time integrator
- 3 timelevels for evolved grid functions
- 3 ghostzones for interprocess synchronization
- Reflection symmetries
- 5th order accurate spatial and 2nd order accurate temporal interpolation at mesh refinement boundaries
- 5th order Kreiss-Oliger dissipation terms added to RHS
- Dirichlet boundary condition
- No I/O (Cactus/Carpet timer/memory stats at end
- Grid sizes such that a benchmark run requires approximately 650 MByte per core, allowing it to run efficiently on systems with 1 GByte per core,
- Iterations chosen for 10 minute runs on current hardware.

Weak Scaling BH Benchmark

	Cores	Grid Shape	Points(K)	Iters	Memory /GF/Core	Data Exchanged
Unigrid	1	65× 65× 65	275	24	2.73 MByte	2.42 MByte
	2	130× 65× 65	549			
	4	130×130× 65	1,099			
	8	130×130×130	2,197			
	16	260×130×130	4,394			
	...	...	...			
AMR	1	25× 25× 25	141	128	2.05 MByte	3.22 MByte
	2	50× 25× 25	281			
	4	50× 50× 25	563			
	8	50× 50× 50	1,125			
	16	100× 50× 50	2,250			
	...	...	...			

168K lines of code
+ 50K comments

291 or 219 GFs

25 or 24 GFs synced

Strategy

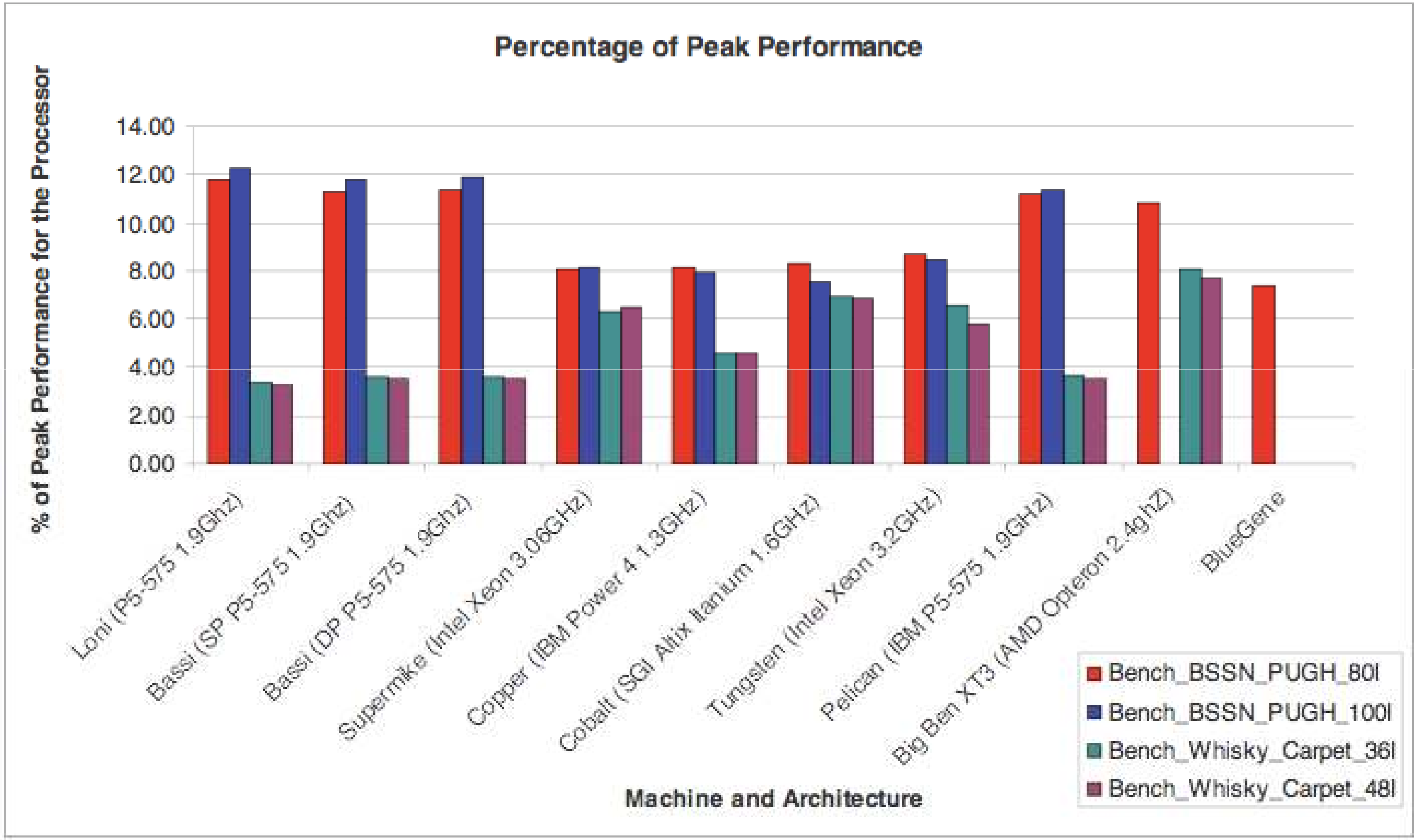
- Single core performance
  - Strategies for better cache use
  - Understanding performance data
- Node scaling
  - Memory bandwidth limitations
  - OpenMP/MPI
  - Accelerators
- MPI scaling
  - Load balancing

Single Core

	Unigrid CCATIE	PSU	RIT	*AMR* CCATIE
L2 cache bandwidth (MByte/s)	308.637	160.382	277.261	272.158
L3 cache bandwidth (MByte/s)	254.191	207.130	224.477	264.886
Flop/cycle	0.133	0.157	0.180	0.160
Flop/grid point update	5,708	5,352	6,481	8,399
L2 cache miss ratio	0.022	0.012	0.023	0.021
MIop/s (wall clock)	1242.138	1479.648	1376.781	1452.769
MFlop/s (wall clock)	416.236	490.004	566.753	490.924
Percentage of cycles stalled	5.3%	6.8%	4.1%	6.8%
Percentage of peak	6.5%	7.7%	8.9%	7.7%
Processor utilization	98.11%	98.02%	98.90%	96.36%

Measured with Cactus timers (PAPI) and Perfsuite

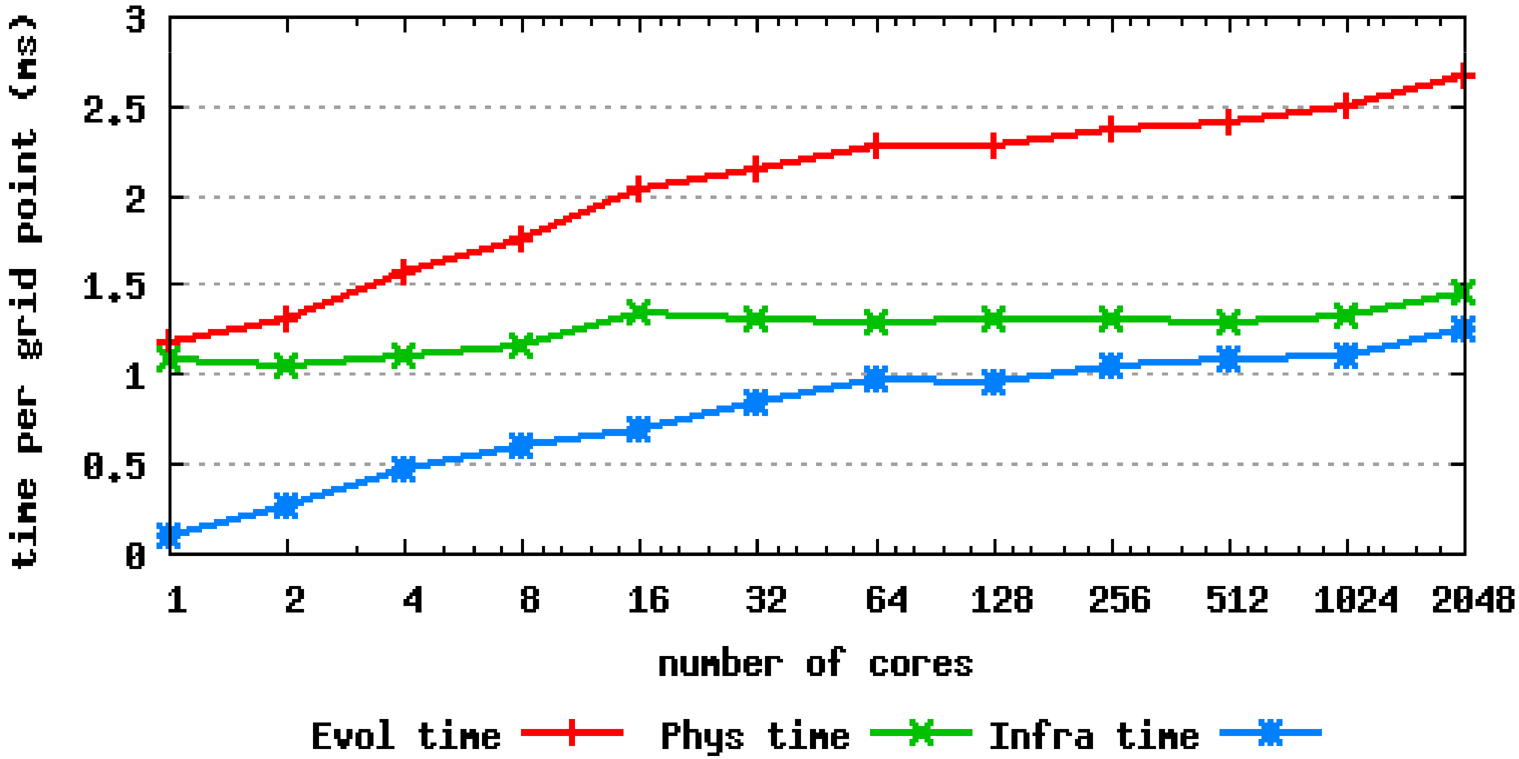
Single Core Study (2006)



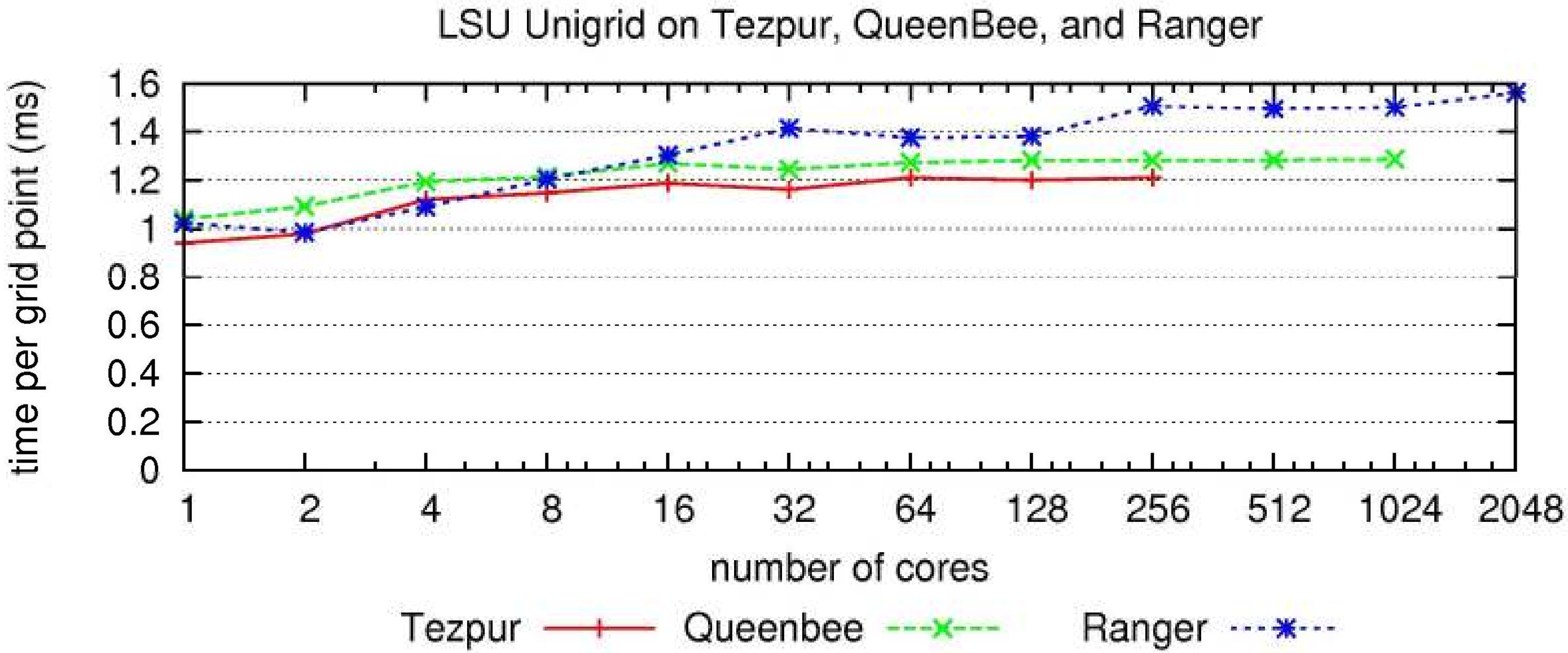
http://www.cactuscode.org/Articles/Cactus_Madiraju06.pdf/

Timing Methodology for Scaling

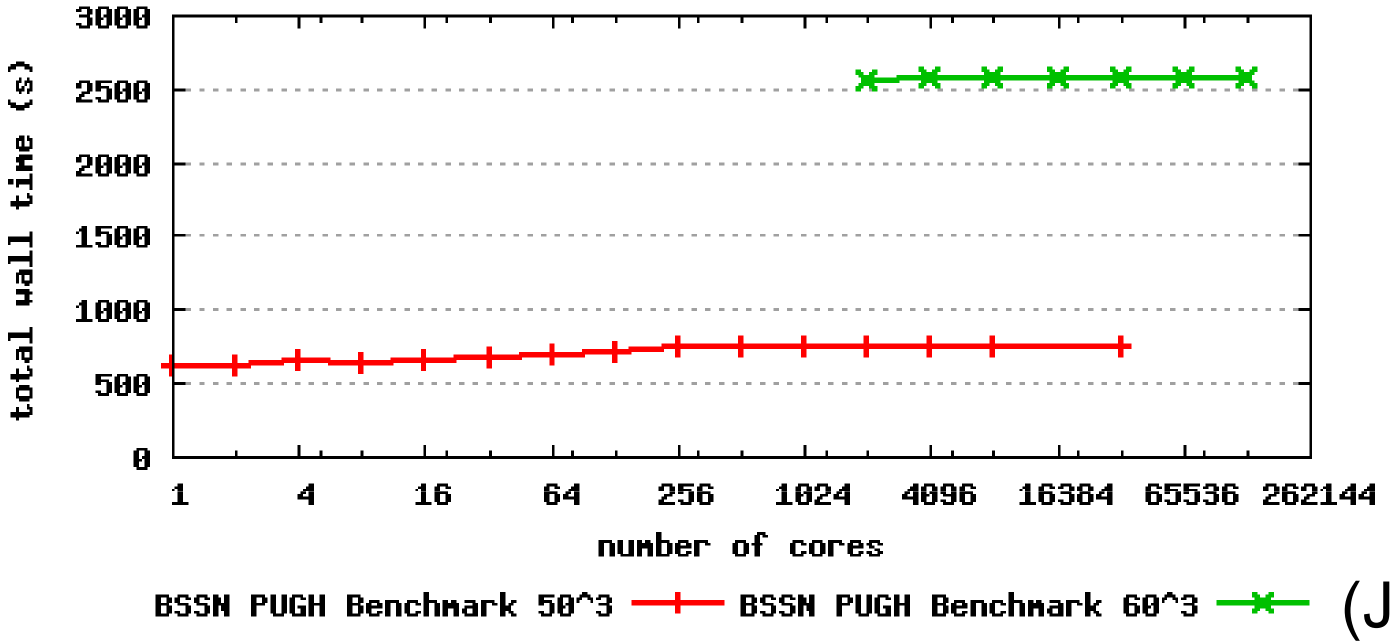
- Cactus timers: getrusage, gettimeofday
- 3 runs, average across procs
- Calculate: Evolution time, Physics time, Infrastructure (comm, regrid, ..) time



Weak Scaling Unigrid



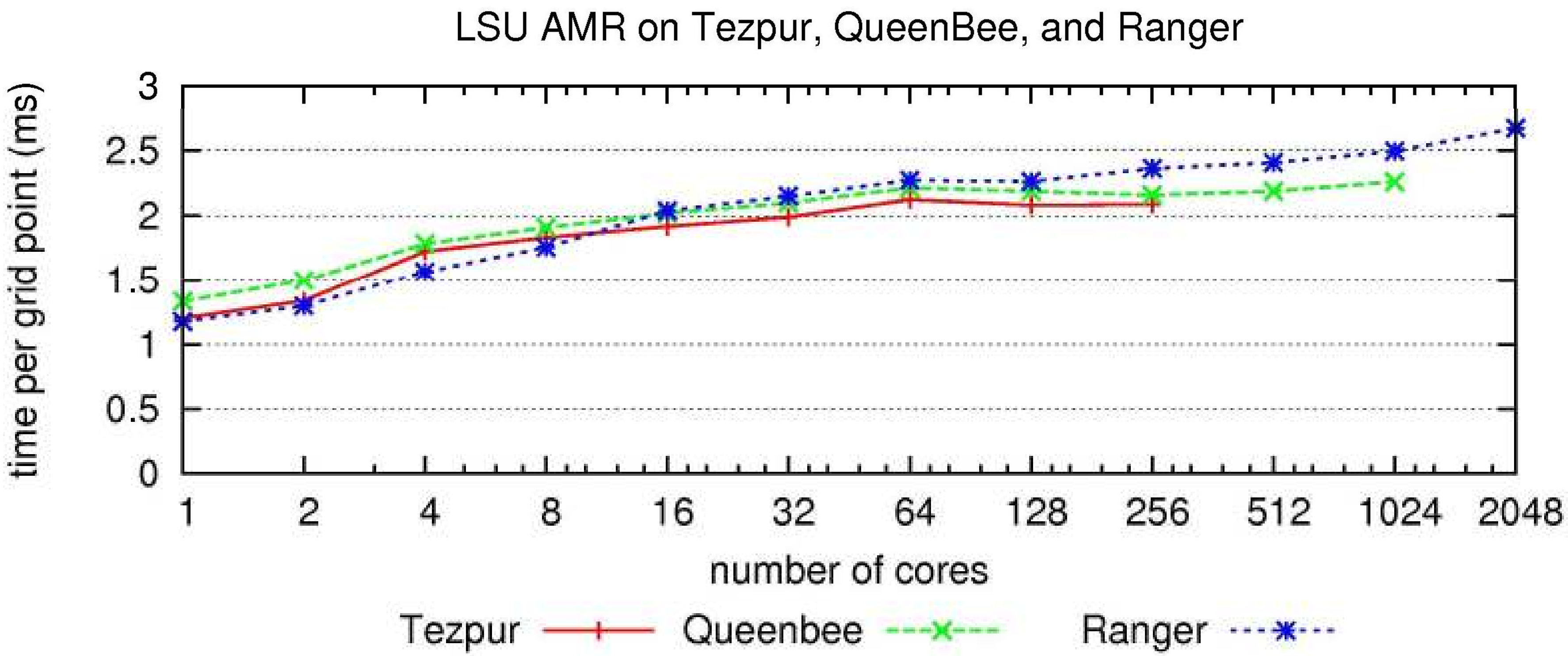
UniGrid: ANL BG/P



(Jian Tao)

- Weak scaling to 131,072 cores (out of 163,840 available) with PUGH
- Amended vacuum weak scaling benchmark (smaller grids)

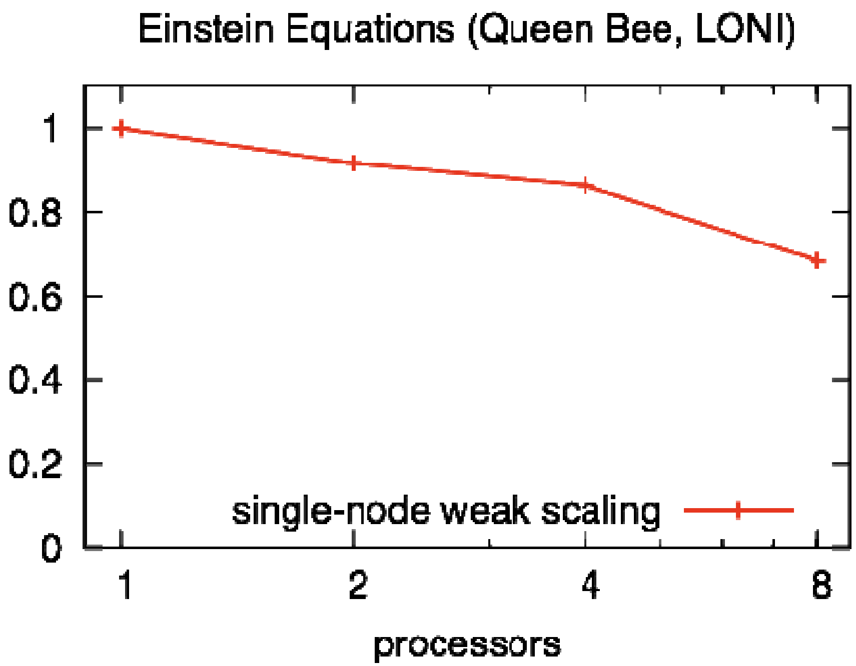
Weak Scaling AMR (9 levels)



OpenMP

- Much easier to program than MPI
- Different processors can access the same memory, only the work is distributed – saves parallelisation ghost zone overhead
- Can add OpenMP directives to serial code piece by piece, starting with expensive routines
- Directives are ignored by default

Single-Node Scaling

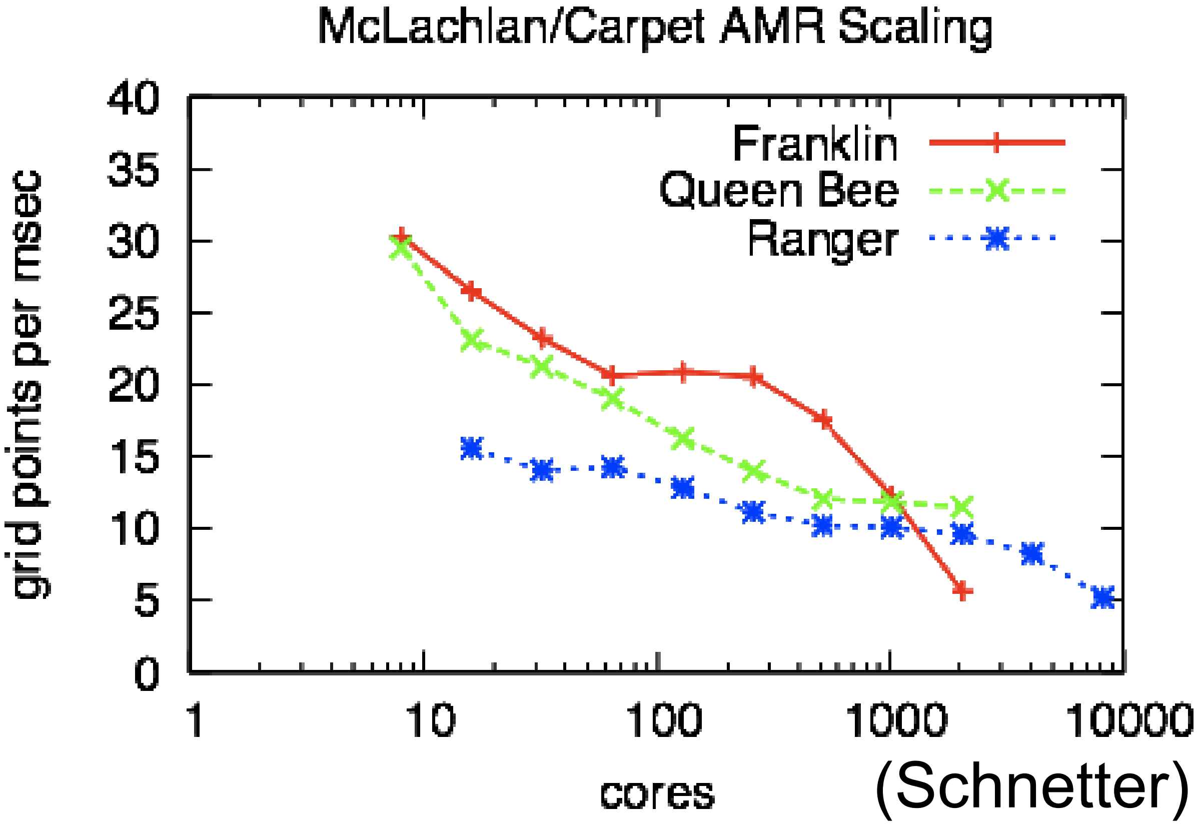


Parallelisation with OpenMP

- Full Einstein equations, 65^3 grid points per processor
- Scaling limited by cache performance
- 8th core still increases performance (but not linearly)
- Need advanced, dynamic cache optimisations

Hybrid Weak Scaling

- Franklin (NERSC): Cray XT4, 2 cores/node [preliminary results; using only 1 thread]
- Queen Bee (LONI): Intel, 8 cores/node [using 8 threads]
- Ranger (TACC) AMD, 16 cores/node, NUMA with 4 banks [using 4 threads]



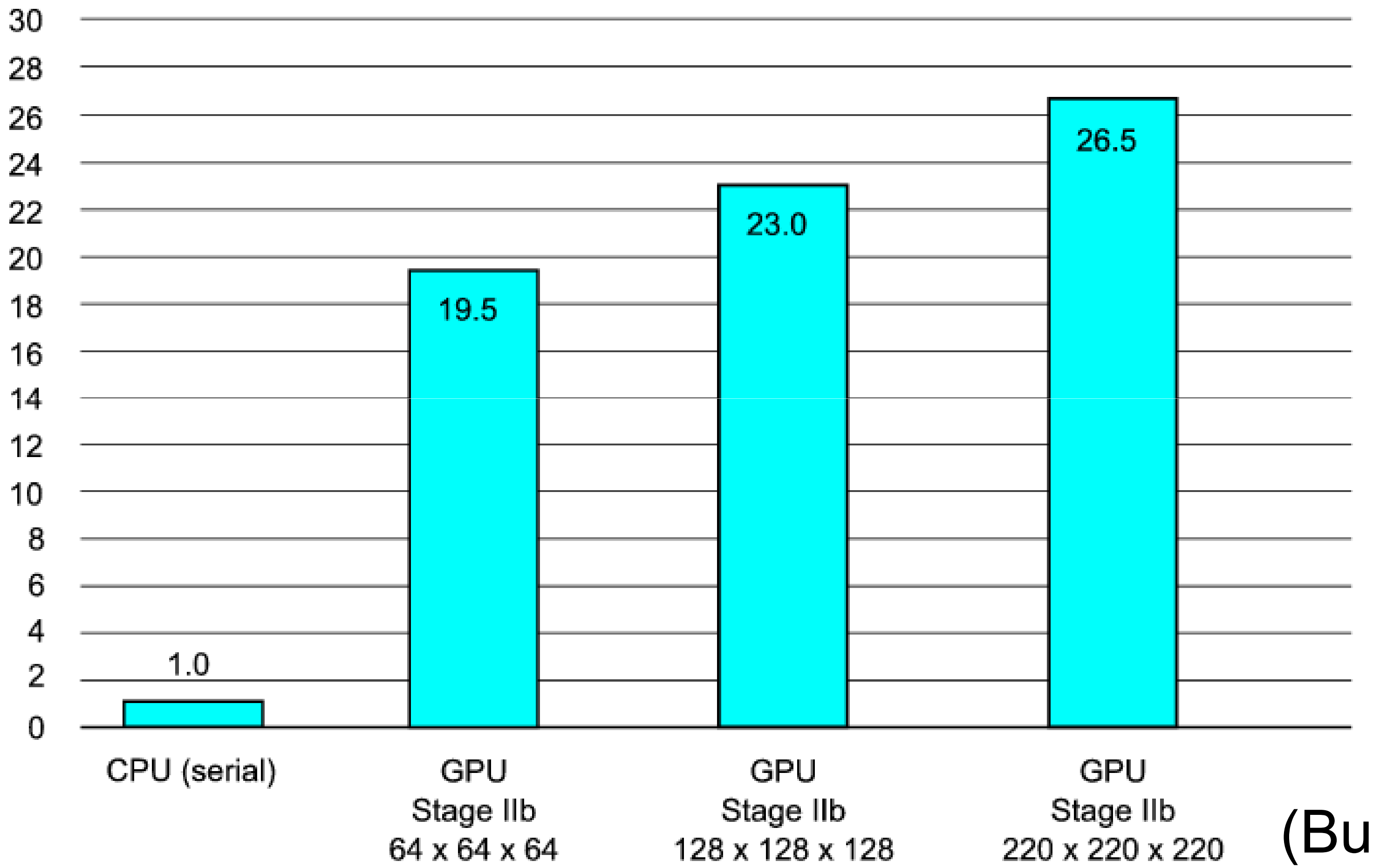
(Schnetter)

Hydrid approach so far ranges from no speed up to 10% speed up (Abe/QB) over pure MPI. Benefits are future optimization possible, less memory used (no ghostzones), more stable for large scale (with developing MPI implementations)

LoopControl

- New thorn (library), providing macros to iterate over 3D arrays, easy to use
- Uses loop tiling to use the cache efficiently
- Uses OpenMP, if enabled
- Uses random-restart hill-climbing algorithm to optimise its parameters automatically at run time
- 10% speed up seen currently, more investigations needed, potential for multiple times speed up if can better use cache.

Accelerators: GPUs



(Burkhard Zink)

Speed up of Black Hole code on
NVIDIA Quadro FX 5600 GPU (CCT-TR-2008-1)

Other Issues

- I/O
  - Checkpoint/restart
  - HDF5
  - Many files, different formats
- Provenance information (Formaline)
  - Automatically collect information on machine config, Cactus source code, profiling information, etc

Final Thoughts

- Cactus/Carpet development challenges
  - Dynamic AMR load balancing
  - I/O (different strategies for diff machines)
  - Regridding still too expensive
  - Performance across **all** thorns
  - Hydrid model/Accelerators
- General challenges
  - Need better access to machines (short queues, interactive, large procs)
  - Main tools rdtsc, printf. gprof too coarse, PAPI, Tau hard to install/configure
  - Data structures becoming more complex
  - Cactus model has many developers, most do not produce scalable code. Need application level tools to guide them (ALPACA project)