Fault tolerant issues in large scale applications

Romain Teyssier

University of Zurich

George Lake, Ben Moore, Joachim Stadel
and the other members of the project
«Cosmology at the petascale»

HP2C
High Performance and
High Productivity Computing

SPEEDUP 2010
Computational cosmology: an overview

Our 2 massively parallel applications
  - RAMSES
  - PKDGRAV

Fault-tolerant issues in large scale simulations

Some solutions at the application level:
  - efficient checkpoint restart
  - «divide and conquer»
Cosmological simulations

From Gaussian random fields to galaxies: non-linear dynamics of gravitational instability with N-body and hydrodynamics codes.
Cosmological simulations

Springel et al., Nature, 2006
Adaptive Mesh Refinement with Octree
RAMSES: a parallel AMR code

• Graded octree structure: the cartesian mesh is refined on a cell by cell basis
• Full connectivity: each oct have direct access to neighboring parent cells and to children octs (memory overhead 2 integers per cell).
• Optimize the mesh adaptivity to complex geometry but CPU overhead can be as large as 50%.

**N body module:** Particle-Mesh method on AMR grid (similar to the ART code). Poisson equation solved using a multigrid solver.

**Hydro module:** unsplit second order Godunov method with various Riemann solvers and slope limiters.

**Time integration:** single time step or fine levels sub-cycling.

**Other:** Radiative cooling and heating, star formation and feedback.

MPI-based parallel computing using time-dependant domain decomposition based on Peano-Hilbert cell ordering.

Download at [http://irfu.cea.fr/Projets/Site_ramses](http://irfu.cea.fr/Projets/Site_ramses)
PKDGRAV2: JS and Doug Potter

3. Memory usage reduced by about 70% to 200 bytes/particle.
4. Use of SSE2/3 and Altivec assembly code for interactions.
5. Over 20 times faster for large simulations than PKDGRAV.
6. New I/O system: HDF5 file support, concept of I/O CPUs (RAM Disk).
7. For Solar System work: Very Active Particles, TreeHermite and TreeSymba! R. Morishima
8. Python interface to many high level functions - Analysis!
9. Built in parallel GRAFIC1 and GRAFIC2 initial conditions generation.
10. *No Hydrodynamics, yet…*
Domain decomposition for parallel computing

Parallel computing using the MPI library with a domain decomposition based on the *Peano-Hilbert curve* for adaptive tree-based data structure.

Peano-Hilbert binary hash key is used for domain decomposition (MPI).
Hilbert ordering for optimal data placement in local memory (OpenMP).
Data compression based on 6D Hilbert indexing.

**Implemented in our 2 codes:**
- **PKDGRAV** (TREE + SPH) by J. Stadel
- **RAMSES** (PIC + AMR) by R. Teyssier

Weak-scaling up to 20,000 core.

Dynamical load balancing
Load-balancing issue

Scaling depends on problem size and complexity. Large dynamic range in density implies large dynamic range in time steps. Main source of load unbalance: multiple time steps and multiple species (stars, gas, DM).

Problem: load balancing is performed globally. Intermediate time steps particles are idle.

Solution: multiple tree individually load balanced

Strong-scaling example.
The Horizon simulation

Billion dollar experiments need support from HPC

http://www.projet-horizon.fr

70 billion dark matter particles
140 billion AMR cells

6144 core
2.4 GB per core
Wall-clock time < 2 months

performed in 2007 on the CCRT BULL Novascale 3045

Teyssier et al. 2009
Sousbie et al. 2009
Pichon et al. 2010
Cosmological N body simulations

Moore’s law

Courtesy of V. Springel
The MareNostrum simulation

1 billion dark matter particles
3 billion AMR cells
100 million «star» particles
100 000 galaxies

2048 core
2 GB per core
Wall-clock time 4 weeks

RAMSES code

Performed in 2006/2007 on the IBM PowerPC 970 at BSC

Dekel et al. 2009
Ocvirk et al. 2009
Devriendt et al. 2010
Zoom-in strategy: focus computational resources on a particular Region-Of-Interest and degrade the rest of the box. Much more demanding than full periodic box simulations.

From the “overmerging” problem to the “missing satellites” problem…

Moore et al. 1999
The GHALO project

1 billion particles in the Virial radius
1024 core
2 GB per core
on MareNostrum

PKDGRAV code

Stadel et al. 2009
Zemp et al. 2009
Kuhlen et al. 2010
At the heart of the halo: galaxy formation
Fault-tolerant issues for our large scale simulations

Barcelona simulations (MareNostrum and GHALO):
- time to failure = 1 day
- checkpoint restart file = 100 to 200 GB
- Strong GPFS file-system contention
- time to write = 3 hours

Horizon simulation
- Pre-commissioning mode
- time to failure = 1 hour
- checkpoint restart file = 3 TB
- Very efficient LUSTRE file system
- time to write = 5 minutes

Requirement for a fault-tolerant application:
- time to writes < 10\% time to failure

- time to write depends on the number of I/O nodes
What is a failure in our applications?

From very basic to very subtle failures:
- file system (hardware or software failure) during write
- hardware node failure
- MPI communication failure
- MPI communication completed but corrupted

How to detect a failure?
- log file size not increasing from more than X minutes
- size of check-point restart file not correct
- NaN
- energy conservation errors and small time step size

When do we need to detect a failure?
- should be more frequent than checkpoints
- run daemon with alarm
- stop and restart
Large scale application and fault tolerance

Time to failure scales up with system and problem size. Massively parallel applications use flat model of message exchange. Failure in one node results in the failure of the entire computation. Because gravity is a long-range force, present-day simulations need to access the whole computational volume (fully-coupled mode).

2 solutions:
- efficient checkpoint restart write of global data (dense parallel I/O system)
- divide the volume in independent compute elements
Output files several times per run (one every 2 hours).

Need to minimize computational downtime due to parallel file system contention (competitive disc access).

Compute group: 2048 compute cores:
MPI_COMM_WORLD_COMP

I/O group: 1 I/O core per 32 compute cores:
MPI_COMM_WORLD_IOGRP (64 dedicated I/O cores)

Compute core: -Write replaced by MPI_Send
I/O core:
-Wait for compute core to send data
-Dump data to local scratch disc
-Resume computation when all data received
-Transfer data from local scratch to GPFS
Data compression of simulation snapshots. Each particle belong to a different time step group.

Store less frequently slowly varying particles.
Store more frequently a small subset of the data.

When check-point restarting the application, need to reconstruct the missing data (error control)
Brute force checkpoint restart induces overhead

Cost of Reliability (at 10min/ckp)

Overhead

MTBF (in days)

- Hourly
- Every 2 hours
- Every 6 hours
- Daily

DARPA Exascale Report on Resilience
Divide and conquer: multiple zooms

**Fast multipole method:** using an octree decomposition, distant particles are grouped together on coarser nodes in a pseudo-particle (multipole approximation of the force).

**Zoom-in method:** distant particles are actually replaced by the pseudo-particle. Error is controlled by the «opening angle».

Idea: use the “zoom-in” technique to segment the computational volume into independent zoom simulations. Distant particles are grouped together into massive particles and evolved locally: maximize data locality at the prize of degraded accuracy and overheads.
Divide and conquer: multiple zooms

Increase particle mass gradually to minimize errors.
Divide and conquer: multiple zooms

Initial and final particle distribution for one single halo.
Divide and conquer: multiple zooms

Check a posteriori mass distribution errors (<1%) and contamination (none).
Divide and conquer: multiple zooms

Divide a large scale cosmological application with $10^4+$ core into many (>1000) independent small scale zooms (<32 core).

Each individual zoom use small checkpoint restart files.
In case of failure, resubmit to another node.
Work in «embarrassingly parallel» mode.

Caveats:
- very large spectrum of halo sizes. Larger halos won’t fit.
- need a redundant, highly available storage device
- selection of the halo sample based on user priors: biased approach?

Tests:
- deploy our multiple zoom scripts in a fault simulation framework.
- use available queue system to perform «idle time» computation.
- use grid infrastructures.
Grid computing as a laboratory for fault-tolerant computing.

We used the DIET grid middleware to run a large scale experiment on Grid5000, the French research grid.

We obtained a 80% success rate on 3200 cores deployed over 13 sites. The main cause of failure was file system related (2 sites lost).

Caniou et al., Fourth HPGC, 2007
Conclusion

- Computational cosmology is still limited by computational power.

- Our large scale application are massively parallel and very memory demanding.

- We had some (unpleasant) experience with system, disc and MPI failures.

- Our present-day application are fault-tolerant because of frequent checkpoint restart dumps: requires a very efficient file system and/or an optimal «parallel write» strategy.

- A radical modification of our applications is considered using a multiple-zoom approach.

- Other approaches are also possible and most welcome (RAID computing, system resilience).
The HP2C Cosmology Team

George Lake

Romain Teyssier

Ben Moore

Joachim Stadel

Jonathan Coles

Markus Wetzstein

Doug Potter

Christine Moran