Performance Emulation of the Cell-based AMR Cosmology Simulation Code – ART

Jingjin Wu[1], Roberto E. González[2], Zhiling Lan[1], Nickolay Y. Gnedin[2,3], Andrey V. Kravtsov[2], Douglas H. Rudd[4], Yongen Yu[1]

[1]Illinois Institute of Technology, [2]The University of Chicago, [3]Fermi National Accelerator Laboratory, [4]Yale University

{jwu45,lan,yyu22}@iit.edu, {regonzar,andrey}@oddjob.uchicago.edu, gnedin@fnal.gov, douglas.rudd@yale.edu

Motivations

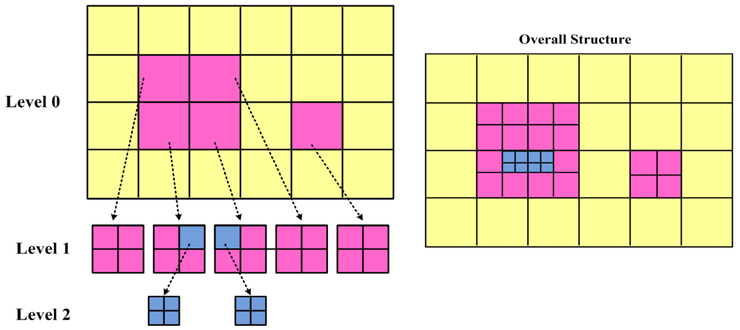
Challenges in designing highly efficient cosmology simulation code:

- As more physics processes are included, cosmology simulations become more realistic and complex than ever before.
- Cosmology simulation codes often employ adaptive mesh refinement (AMR), which use complicated data structures and uneven communication among processes in parallel implementation.
- The physics evolution of cosmic structures is rapid and uneven, thus making dynamic load balancing a challenging task.

In order to explore efficient load balancing schemes, we study the Adaptive Refinement Tree (ART) code, and design a performance emulator for quickly evaluating the performance of various load balancing schemes.

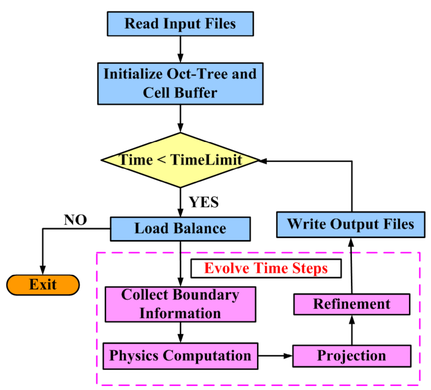
The Adaptive Refinement Tree (ART) Code

- The ART code is an advanced N-body+hydro simulation tool.
- It is an "MPI+OpenMP" C code, with Fortran functions for compute intensive routines.
- It employs the cell-based adaptive mesh refinement (AMR) algorithm.

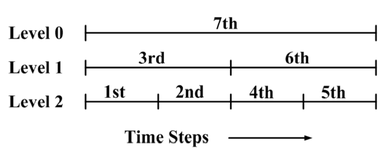


A 2D cell-based AMR example: a quad-tree with a refinement factor of 2. The ART code adopts a 3D computational domain consisting of cubic cells, which form an oct-tree.

The ART code simulates the evolution of the universe. In each iteration, it evolves a time advance *dt*. This is achieved by recursively evolving time steps for all the levels. The figure shows the flow control of the ART code. The four steps in the dotted region are the major steps for evolving a time step at each level.



The figure presents the recursive execution order for evolving time steps at three levels (level 0 to level 2).



Details about cell-based AMR and the ART code are available in [1] and [2], respectively.

Performance Models

Physics Computation Time:

$$T_P = w_1 \times N_{nlc} + w_2 \times N_{lc} + w_3 \times N_{particle}$$

T_P is the physics computation time of a time step for the level of interest. w_i (i=1,2,3) are constant coefficients for the level. N_{nlc}, N_{lc} and $N_{particle}$ are the number of non-leaf local cells, leaf local cells and particles, respectively. To extract these constant coefficients for each level, we can simulate a few iterations, formulate the above equation for each process, and then solve a linear system to obtain the best fit solution of coefficients. These coefficients can be viewed as the expected runtime or each cell or particle.

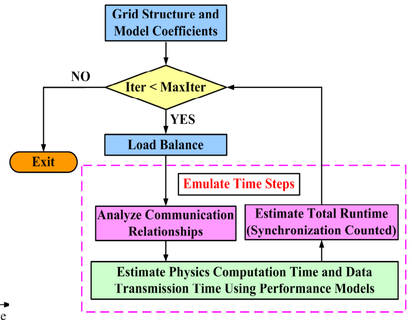
Data Transmission Time:

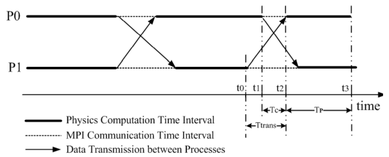
$$T_{trans} = t_s + n \times t_c$$

T_{trans} is the data transmission time, t_s is the latency for message passing, t_c is the inverse of the bandwidth, and n is the number of bytes for one time data transmission. The latency and bandwidth can be obtained by using Intel MPI Benchmarks (IMB).

Emulator Design

The right figure presents the design of our emulator. According to the cell and particle counts and communication relationships, the emulator estimates physics computation time and data transmission time using the performance models. The total runtime is achieved by maintaining a time axis (shown in the left figure) for each process to record computation and communication intervals.





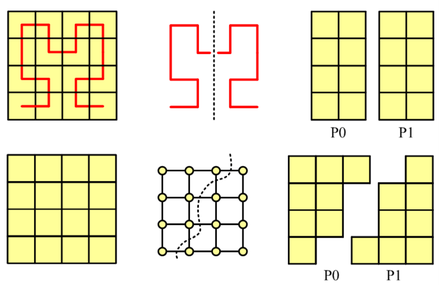
The left figure shows the time axes of two processes P0 & P1. The 2nd MPI communication communication time interval of P0 is computed as $Tc = t2\text{-}t1 = (t0 + Ttrans) - t1$. Besides, $t3 = t2 + Tp$. The MPI communication time include both data transmission time and synchronization time without evaluating synchronization time separately.

Load Balancing Schemes

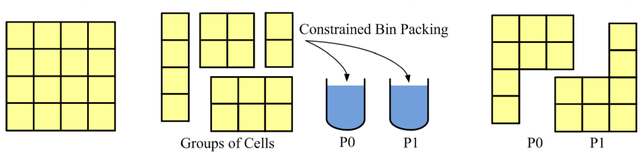
The basic unit for load balancing is root cell at level 0.

SFCLB: It employs a Hilbert space filling curve (SFC) [3] to cover all root cells, and then cuts the curve into some segments with similar amount of workload.

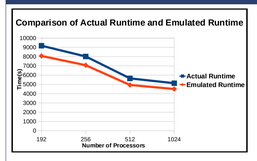
GraphLB: It converts the load balancing problem into a graph partitioning problem by mapping root cells into vertices, and the neighboring relationships into edges. Its objective is to minimize the total edge weight subject to the constraints that the partitions are of equal size. METIS [4] is employed for implementation.



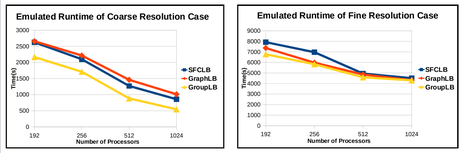
GroupLB: It takes into consideration level-by-level balance and attempts to minimize synchronization cost. It generates many groups of root cells according to their communication relationships, and then assigns these groups to processes by solving a constrained bin packing problem.



Experimental Results



We employ a realistic cosmology data with a computational domain of 256^3 root cells. As the ART code only supports SFCLB, we compare the emulated runtime with the actual runtime using the SFCLB. Their difference is within 12% of the actual runtime. Both curves have the same trend as the number of processors increases.



Use two resolution case for performance evaluation with the emulator:

(1) Coarse resolution case with max refinement level of 6.

(2) Fine resolution case with max refinement level of 9.

$$Load\_Balance\_Ratio \triangleq \frac{\frac{1}{N_p}\sum_{i=0}^{N_p-1} W_i}{\max_{0\le i\le N_p-1} W_i} \times 100\%$$

Wi is workload of process i and Np is the number of processes.

Overall load balance ratio of different load balancing schemes

Number of Processors	Coarse Resolution Case			Fine Resolution Case		
	SFCLB	GraphLB	GroupLB	SFCLB	GraphLB	GroupLB
192	92.83%	64.89%	96.63%	90.21%	85.48%	91.17%
256	91.16%	73.54%	96.43%	71.35%	78.81%	88.51%
512	82.51%	53.76%	95.84%	53.74%	54.02%	51.73%
1024	70.49%	42.74%	92.21%	26.98%	27.01%	26.75%

Generally, GroupLB provides the best performance. It achieves a good load balance quality by balancing both overall and level-by-level workload, and minimizes communication cost by preserving spatial locality. For the fine resolution case, these three load balancing schemes have similar performance when there are more than 512 processors because of granularity. Granularity is imposed by our choice of simulation and input. It is not a fundamental limitation.

References

[1] T. Plewa, T. Linde and V. G. Weirs, *Adaptive Mesh Refinement–Theory and Applications*, Springer, 2005.

[2] A. V. Kravtsov, A. A. Klypin, and A. M. Khokhlov, "Adaptive refinement tree: a new high-resolution N-body code for cosmological simulations," *Astrophysical Journal Supplement*, vol. 111, p. 73, Jul. 1997.

[3] A. R. BUTZ, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Trans. Computers*, pp. 424–426, Apr. 1971.

[4] METIS, http://glaros.dtc.umn.edu/gkhome/views/metis.

Acknowledgments

This work is supported in part by National Science Foundation grants OCI-0904670. Jingjin Wu is in part supported by China Scholarship Council.