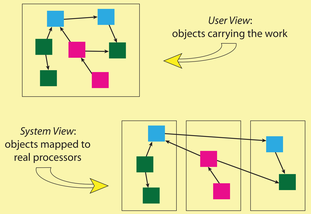
N-chilada: Cosmological Simulations on Supercomputers

Filippo Gioachin, Sayantan Chakravorty, Celso Mendes, Laxmikant Kale, Thomas Quinn

PARALLEL PROGRAMMING LAB

CHARM++ RUNTIME SYSTEM



Objects in the system are called *chares* and can be gathered in collections:
arrays: each chare has a unique index inside the collection, the number of elements in the collection is user defined
groups: there is one and only one chare from this collection in every processor

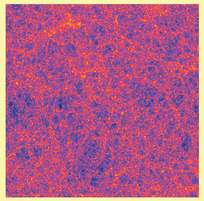
Benefits

Software engineering
- Number of virtual processors can be independently controlled
- Separate VPs for different modules

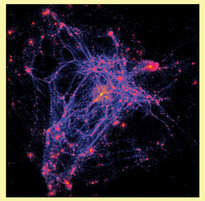
Message driven execution
- Computation performed upon receipt of a message
- Adaptive overlap of communication
- Predictability:
  - Automatic out-of-core execution
- Asynchronous reductions

Dynamic mapping
- Heterogeneous clusters
  - Vacate, adjust to speed, share
- Automatic checkpointing/restarting
- Automatic dynamic load balancing
- Communication optimization
- Change set of processors used

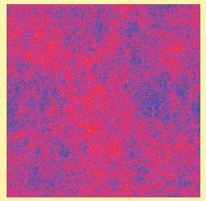
DATASETS



Simulation with 16 million particles of a volume of the Universe 90 Mpc on a side. The initial conditions have been provided by Heitmann et al (2005) in order to test the robustness of cosmological simulations. Statistics of the final state as produced by ChaNGa compare well with the results of Heitman et al using other simulation codes.



Snapshot at z=.3 of a multi-resolution simulation of a dwarf galaxy forming in a 28.5 Mpc volume of the Universe with 30% dark matter and 70% dark energy. Two resolutions are used, with 5 million and 50 million particles. The *mass* distribution is uniform, but the *particle* distribution is very centrally concentrated and therefore highly clustered.



Simulation with 700 million particles of a volume of the Universe 70 Mpc on a side. This simulation is used to follow the star formation history of the Universe.

RESULTS

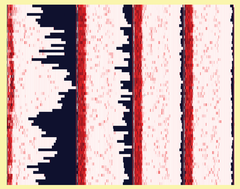


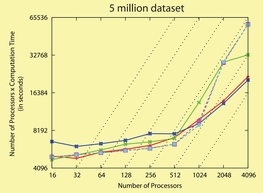
Image from Projections performance analysis tool. The horizontal axis represents time, while the vertical axis represents processors. Colors represent processor utilization, from low in blue to high in white. Four iterations of a simulation with 64 processors on the small dataset. After each iteration the Greedy load balancer redistributes the work, and after two iterations it reaches excellent load balancing.

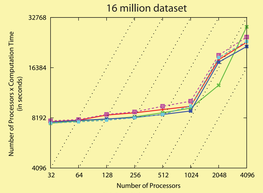
The table shows the percentage of time spent in the various parts of the simulation for the 16 million dataset. When increasing the number of processors, the overhead (marked in purple) due to the parallelization increases, but is still extremely small even for a few thousand processors. Most of the time, as expected, is spent in the force computation.
The values in the table include the idle time occurred during the respective phases. In order to understand the scalability in terms of parallel efficiency, the four graphs below plot the force computation for the different datasets as the number of processors is varied. They also show the behaviour of the application with the two major decompositions (ORB is not considered here due to its bad properties), and with two of the load balancers part of Charm++. The execution times are from the BlueGene/L system.
The smaller datasets reach saturation and stop scaling at a few thousand processors, but for dataset more relevant to modern science, this saturation point is pushed to tens of thousands of processors.

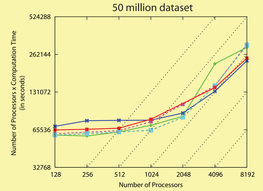
	Number of Processors			
	256	512	1024	2048
Domain decomposition	0.53	0.74	1.45	3.37
Load balancing	0.20	0.42	1.25	4.79
Tree building	0.49	0.75	1.16	2.54
Force computation	98.21	97.56	95.73	89.04
Time integration	0.57	0.54	0.41	0.26
Total	100.00	100.00	100.00	100.00

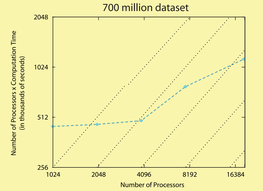
Graphs legend
- OCT decomposition, Orb load balancer
- OCT decomposition, Greedy load balancer
- OCT decomposition, no load balancer
- SFC decomposition, Orb load balancer
- SFC decomposition, no load balancer

The graphs plot execution time multiplied by number of processors against number of processors used in the computation. This implies that horizontal lines show perfect scalability, while diagonal lines show no scalability.







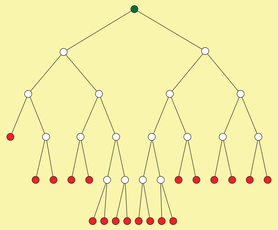


CHANGA: CHARM N-BODY GRAVITY

in serial → and in parallel

Motivation

Current cosmological parallel simulators have difficulties scaling beyond a few hundred processors. In particular, the main limitation is the increased time spent in decomposing the datasets among the processors, and maintaining the load balancing.
Charm++ can handle application decompositions having uneven granularity with its automatic load balancing framework. This will help in both reducing the decomposition time and enhancing the overall load balance.
By using an asynchronous message based system, objects can proceed in the computation independently, without spending time to explicitly wait for data.



Computing gravitational forces

Newtonian gravity is a long-range force: each particle is influenced by all others in the system. $O(n^2)$ complexity.

Barnes-Hut approximation:

The force contribution due to a group of particles distant from the point in which the force is applied can be approximated by the particles center of mass and multipole expansion. Such particles are said to satisfy the Barnes-Hut opening criteria.

Reduction in algorithmical complexity from $O(n^2)$ to $O(n\log n)$

Supported physics

Newtonian gravitational force. Implemented with Barnes-Hut and quadrupole moment expansion (first order accuracy).
Adaptive multiple timestepping. Particles that have high accelerations require more precision and therefore they are advanced in time more often with smaller steps.
Periodic boundary conditions. Periodic Universe is supported by means of Ewald summation.

Building the tree

Tree built on top of the simulation's particles.

- Root node corresponds to the entire simulation volume
- Internal nodes correspond to a subregion of the simulation volume
- Buckets correspond to internal nodes containing a few particles

Top-down construction (step 1):
Starting from the root, internal nodes are recursively created by dividing the space in halves. When a node contains less than '*b*' particles (default 12), it is transformed into a bucket and the recursion terminates

Bottom-up construction (step 2):
The center of mass and multipole expansion are computed for each node; the region of space associated to each node is also tightened to the particles contained by the node. These properties are directly computed from the particles for the buckets, and from the children nodes for internal nodes.

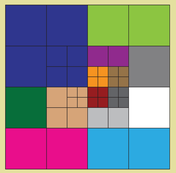
Walking the tree

For each particle, to compute the force it is subject to:
1) Start at the root node of the tree;
2) Visit the nodes of the tree recursively until nodes representing groups of particles that satisfy the Barnes-Hut opening criteria are found;
3) Compute the interaction between the particle in subject and all the nodes where the recursive visit has stopped.

Notice, in some parts of the tree, where particles are close, direct computation will be necessary.

Decomposing the data into *TreePieces*

The entire set of particles of the simulation is divided into the desired number of pieces and distributed among the computation objects. These objects are called *TreePieces* and are implemented as a Charm++ array collection.
ChaNGa has three methods to partition the simulation's particles set. These are described below, with a simplified example of such decomposition in 2D. The big boxes represent the entire simulation's space, while each small box represents a region of space containing an equal number of particles. Smaller boxes represent higher density regions. Colors represent the regions of space associated with different TreePiece.



OCT

The space is recursively divided in halves, each of equal volume size. Each TreePiece is assigned a cubic (or rectangular) portion of space.

🙂 Can split dense regions only after higher cuts are made. Not a problem with enough chares.
🙂 Potentially some chares will be underloaded. The Charm++ Load Balancer will take care of them.
🙂 Spatial locality preserved.

SFC

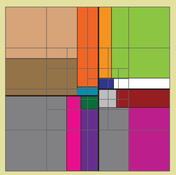
After a total ordering of the particles is imposed with a Morton Space-Filling-Curve, each chare is assigned a segment of the curve. Each segment contains an equal number of particles.

🙂 All chares will contain the same amount of particles.
☹ Many chares will have non-contiguous regions of space, therefore spacial locality is not preserved.

ORB

The ORB (Orthogonal Recursive Bisection) decomposition is performed by recursively dividing the space in halves, each containing an equal amount of particles.

🙂 Expensive due to the high dependence of subdivisions from the global distribution.
🙂 Creates elongated shapes. As a consequence, the amount of work required for the force computation increases significantly.
🙂 It preserves spacial locality.

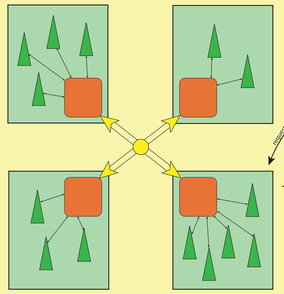


Building the tree in parallel

In the entire system there is one single global tree which is built across all chares, and it is distributed among all of them.
A Treepiece has information regarding the nodes of the tree that are ancestors of any particle belonging to that Treepiece. As a consequence, the nodes closer to the root will be shared and duplicated among multiple Treepieces.

During the top-down construction, each Treepiece builds the part of the tree related to the particles that it contains. During the bottom-up construction of the tree properties, the information related to nodes that are non-local is requested to one of the Treepieces co-sharing the missing node.

Overview of the entire system



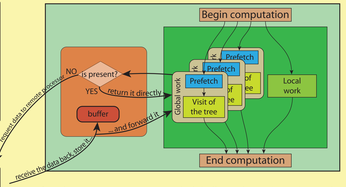
Treepiece: contains a portion of the particles of the simulation and performs the computation and update of those particles.

CacheManager: performs optimization at the processor level allowing a transparent reuse of imported data both within a single TreePiece, and across all TreePieces in the same processor.

Intraprocessor communication: Communication between TreePieces and the CacheManager. This is mostly performed through function calls.

Interprocessor communication: The CacheManagers communicate with each other to fetch the remote information needed for the computation. This is performed through Charm++ messages.

Control flow inside a single processor



After the computation is started globally, each TreePiece (green box) receives messages to perform its part. The TreePiece splits the total work into local work, which includes interaction with its own portion of the tree, and global work, which includes interaction with the rest of the system. The global work is composed of two stages: prefetching of the needed data, and visit of the tree. To pipeline the prefetching with the visit of the tree, the global work is splitted into chunks.
During the global work, the TreePiece requests the CacheManager (orange box) for data not locally present. If the data is already buffered, it is immediately returned, otherwise it is requested to another processor. When the data comes back, it is buffered and forwarded to the requesting TreePiece.

Gravity computation in parallel

Each TreePiece is responsible for the particles it contains. It computes all the forces applied to such particles. The information needed for such computation that is not available locally is fetched from the TreePiece which has it.

CacheManager:
Since the same information will be needed multiple times both by a single TreePiece, and by different TreePieces, it is important for performance to fetch it once and use it as much as needed.
The CacheManager is a software cache placed on the data fetch path. Its purpose is to buffer the fetched information for later reuse. It is implemented as a Charm++ group collection.

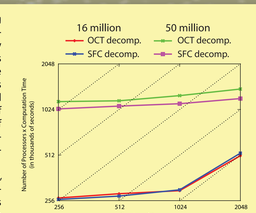
CacheManager importance

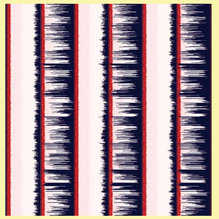
The CacheManager is essential for the efficiency since it dramatically reduces the number of messages exchanged. The table shows this reduction, as well as its impact on the execution time.

Number of Processors		4	8	16	32	64
Number messages (in thousands)	No cache	48,723	59,115	59,116	68,937	78,086
	With cache	72	115	169	265	397
Time (in seconds)	No cache	730.7	453.9	289.1	67.4	42.1
	With cache	39.0	20.4	11.3	6.0	3.3
Speedup		18.74	22.25	25.58	11.23	12.76

	Number of Processors			
	256	512	1024	2048
Domain decomposition	1.54	2.43	6.20	12.49
Tree building	0.45	0.80	1.89	3.08
Force computation	96.78	95.69	91.03	83.53
Time integration	1.22	1.08	0.89	0.90
Total	100.00	100.00	100.00	100.00

During multistepping, many iterations compute the forces for a small subset of all the particles in the system. This leads to different distribution of work in each iteration, and therefore more complexity for load balancing. Currently we do not use the Charm++ load balancer for multistepping simulations.
The Projections figure below shows that during some of the phases, where only few thousand particles are being updated, the load is unbalanced. Nevertheless, the overall performance of the application is remarkable as can be seen by both the time spent in the different phases in the table above, and in the scalability graph on the right.





FUTURE WORK

On the astrophysics side, we are planning to add more relevant physics to the application, in particular SPH (Smooth Particles Hydrodynamics) to allow gas simulations.
On the computer science side, we are planning to study in detail the limitations that we are facing with thousands of processors to allow better scaling on large configurations. In particular, we plan to enhance the Charm++ load balancing framework both to provide a more stable behaviour with large configuration, and to enable the load balancing of multistepping simulations.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

For more information, please visit us at

http://charm.cs.uiuc.edu/
http://hpcc.astro.washington.edu/