Hierarchical Task Mapping of Cell-based AMR Cosmology Simulations

Jingjin Wu*, Zhiling Lan*, Xuanxing Xiong§, Nickolay Y. Gnedin†‡, and Andrey V. Kravtsov‡
*Dept of Computer Science, Illinois Institute of Technology, Chicago, IL
§Dept of Electrical & Computer Engineering, Illinois Institute of Technology, Chicago, IL
{jwu45,xxiong3}@hawk.iit.edu, lan@iit.edu
†Theoretical Astrophysics Group, Fermi National Accelerator Laboratory, Batavia, IL
gnedin@fnal.gov
‡Dept of Astronomy & Astrophysics, The University of Chicago, Chicago, IL
andrey@oddjob.uchicago.edu

***Abstract*—Cosmology simulations are highly communication-intensive, thus it is critical to exploit topology-aware task mapping techniques for performance optimization. To exploit the architectural properties of multiprocessor clusters (the performance gap between inter-node and intra-node communication as well as the gap between inter-socket and intra-socket communication), we design and develop a hierarchical task mapping scheme for cell-based AMR (Adaptive Mesh Refinement) cosmology simulations, in particular, the ART application. Our scheme consists of two parts: (1) an inter-node mapping to map application processes onto nodes with the objective of minimizing network traffic among nodes and (2) an intra-node mapping within each node to minimize the maximum size of messages transmitted between CPU sockets. Experiments on production supercomputers with 3D torus and fat-tree topologies show that our scheme can significantly reduce application communication cost by up to 50%. More importantly, our scheme is generic and can be extended to many other applications.**

I. Introduction

Simulations are critical to making quantum leaps in cosmological research as they provide insight for the evolution of the universe, e.g., the formation of stars and galaxies. There are mainly two categories of cosmology simulation tools: those that only simulate the dark matter (often referred to as "N-body"), and those that model gas dynamics (often called "hydro"). Since cosmologically relevant scales are mainly dependent on the dark matter, a hydro simulation tool always includes an N-body component for modeling the dark matter. Typically, hydro simulations are much more compute-intensive than purely N-body simulations. Modern hydro simulations become even more computationally demanding in terms of both runtime and memory as more and more physical processes are included, e.g., gas cooling, star formation and feedback, radiative transfer, and so on. Adaptive mesh refinement (AMR) [1] has been widely applied to model the dynamics of cosmic baryons (gas and stars) for cosmology simulation, since it can follow the fragmentation of gas down to virtually unlimited small scales. Several parallel codes based on AMR have been developed for efficient large-scale cosmology simulations, including Enzo [2] and the adaptive refinement tree (ART) code [3].

In practice, production cosmology simulations often take a large amount of runtime, e.g., several days, weeks or even months, depending on the problem size and the amount of computing resources used. Even with well-implemented cosmology simulation codes, it is still challenging to achieve scalable performance on high performance computing (HPC) platforms due to the communication time between processes. From the HPC system perspective, the interconnection networks are always sparse, e.g., fat-tree, 3D mesh or 3D torus. As the system scales up, the diameter of the interconnection network (i.e., the maximum distance between two nodes) increases, and the bisection bandwidth (i.e., the minimum total bandwidth of links connecting one half of the HPC system and the other) often decreases. Consider that parallel cosmology simulations usually have sparse communication pattern, it is critical to map processes onto nodes properly, so that the traffic in the network will be localized, leading to better communication performance.

Finding an appropriate mapping of parallel application processes onto nodes is called *topology-aware task mapping*. It considers application-specific communication pattern and the underlying network topology, and typically aims to reduce the amount of traffic in the network. This problem has been much studied for regular communication pattern and regular network topologies [4]–[6]. In practice, the communication pattern may be irregular, e.g., the ART code (detailed in Section II). On the other hand, the user application is often assigned non-continuous nodes, which form an irregular topology graph from the user perspective. To the best of our knowledge, there is little work on mapping real-world application processes with irregular communication patterns. Moreover, most works only consider the mapping of processes onto nodes, while the mapping of processes within a node is not investigated. On modern HPC platforms, each node typically contains multiple sockets, with each socket holding a multicore processor. It is observed that intra-socket communication is usually faster than inter-socket communication. Such performance gap should also be exploited in task mapping for performance optimization.

In this paper, we consider cell-based AMR cosmology simulations, in particular, the ART code, and develop mapping

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00

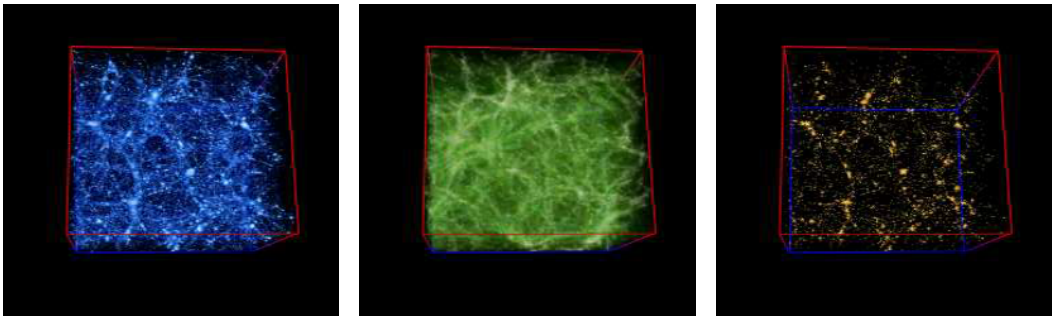

Fig. 1. An example of large-scale ART simulation. Three panels show the dark matter (left), cosmic gas (middle) and stars (right), respectively.

algorithms to map application processes onto multiprocessor clusters, where each node contains several multicore CPUs. In order to exploit the architectural properties of multicore clusters (the performance gap between inter-node and intra-node communication, as well as the gap between inter-socket and intra-socket communication), we propose to perform hierarchical task mapping. First, the mapping of processes onto nodes (i.e., inter-node mapping) is obtained by using the recursive bipartitioning technique to minimize the amount of traffic in the network. Second, for each node, the mapping of processes onto multicore CPUs (i.e., intra-node mapping) is derived by minimizing the maximum size of messages transmitted between CPU sockets. This hierarchical approach has a wide applicability for cell-based AMR cosmology simulations, and the general methodology of performing hierarchical mapping can be extended to many parallel applications. Experiments are performed on NICS Kraken (a cluster with 3D torus network, each node containing two six-core sockets), and TACC Ranger (a cluster with fat-tree network, each node containing four quad-core sockets). Results show that the proposed hierarchical mapping algorithm effectively optimizes both inter-node mapping and intra-node mapping, reducing communication time by up to 50%.

The rest of this paper is organized as follows. Section II introduces the ART code and illustrates its communication pattern. Section III presents our proposed algorithms for hierarchical task mapping. After experimental results are shown in Section IV, we present concluding remarks in Section V.

II. The ART Code

The ART code is an advanced "hydro+N-body" simulation tool for cosmological research. It simulates the evolution of the universe, or more specifically, the formation of stars and galaxies. It employs a combination of multi-level particle-mesh and shock-capturing Eulerian methods for simulating the evolution of dark matter and cosmic gas, respectively. High dynamic range is achieved by applying adaptive mesh refinement to both gas dynamics and gravity calculations. The ART code is distinguished from the rest of cosmological simulation tools in the large number of physical processes it includes, which enable comprehensive simulation of cosmological phenomena and provide deep insight for cosmologists. Fig. 1 shows the visualization of a large-scale ART simulation, including dark matter, cosmic gas and stars.

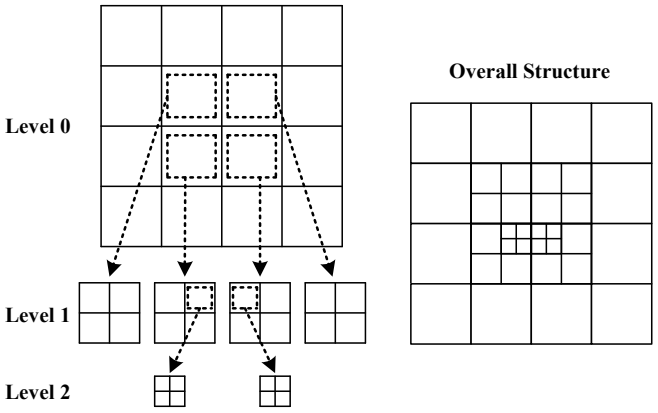

Fig. 2. A 2D cell-based adaptive mesh refinement example: a quad tree with a refinement factor of 2.

As cosmology simulations usually consume a large amount of computing resources in terms of both runtime and memory, they are typically carried out on massively parallel high performance computing (HPC) platforms, e.g., HPC clusters. Production ART simulations typically involve physics computations for thousands of time steps, and can take several weeks or even months using hundreds of processing cores on production HPC platforms. The communication takes significant amount of runtime. Different from existing studies that use dynamic task scheduling and sophisticated load balancing for performance optimization [7], [8], we exploit topology-aware techniques to reduce the communication time. In this section, we present an overview of the ART code to highlight its communication pattern. Additional details about the ART code can be found in [3], [9]–[11].

A. Cell-based AMR

The cell-based AMR algorithm is adopted in the ART code for efficient large-scale cosmology simulation. It achieves high spatial resolution in localized regions by performing refinement and de-refinement based on individual cell. Initially, a uniform mesh is employed for the entire computational domain. During computation, if a cell requires higher spatial resolution, then it is refined into smaller cells at the finer level. If some smaller cells still require higher resolution, then they are further refined into even smaller cells. Each refinement operation decreases the cell size in each dimension by a factor of r, which is called *refinement factor*. If in some region the level of refinement for a cell is larger than required, then

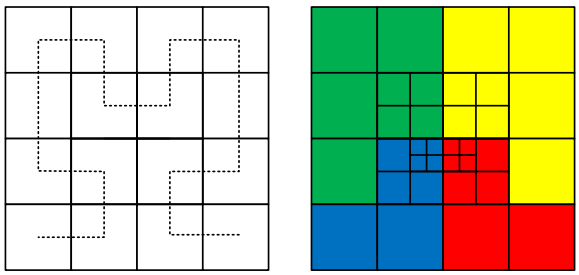

Fig. 3. A space-filling curve traversing a 2D mesh (left), and a parallel partition of the 2D adaptive mesh into four parts (right). The cells with the same color are assigned to the same process.

the high resolution cells are de-refined, i.e., they are removed and replaced with the corresponding coarser cell. The overall computational mesh is refined and de-refined dynamically to fit computation, resulting in a dynamic hierarchy of cells. Fig. 2 shows a cell-based AMR example on the 2D mesh, including the hierarchy of cells and the overall mesh structure. For simplicity, there are only 3 levels of cells from level 0 to 2. The dotted cells are refined into smaller cells at the higher level as higher spatial resolution is required. The cell-based AMR algorithm enables cosmologists to perform large-scale cosmology simulations which are completely intractable on a uniform mesh.

The ART code considers a cubic computational domain, which represents the universe. At the beginning, the cubic domain is divided into many uniform cubic cells, which are called *root cells*. Throughout the ART simulation, cells are dynamically refined and de-refined with a refinement factor of 2, i.e., each refinement operation evenly divides a cubic cell into 8 cells, namely an "oct". All the cells are organized in oct-trees [12], [13], so that the adaptive mesh can be built and modified in parallel efficiently.

B. Domain Decomposition

In order to take advantage of the cell-based AMR and exploit the spatial locality, the ART code adopts a domain decomposition scheme [14] based on Hilbert's space-filling curve (SFC) [15]. The SFC is identified by a traversal of all the root cells according to their spatial coordinates. A parallel partition of root cells is obtained by dividing the curve into N_p (number of processors) equal parts, where each part is weighted by the total workload of the corresponding computational domain. It is to be noted that each root cell keeps all its child cells at finer refinement levels as a single composite unit, thus being the basic object for domain decomposition. Fig. 3 presents a space-filling curve on a 2D mesh and a parallel partition of cells into four parts. Because of the spatial locality preserved by the curve, each part is a continuous domain consisting of nearby cells, and this property also holds for the 3D case of parallel partition of the cubic universe in ART. This SFC-based domain decomposition scheme enables efficient partitioning of the adaptive mesh by transforming a multidimensional problem (e.g., 2D or 3D) into a unidimensional one, and it has been widely employed in parallel AMR implementations [3], [16]–[18].

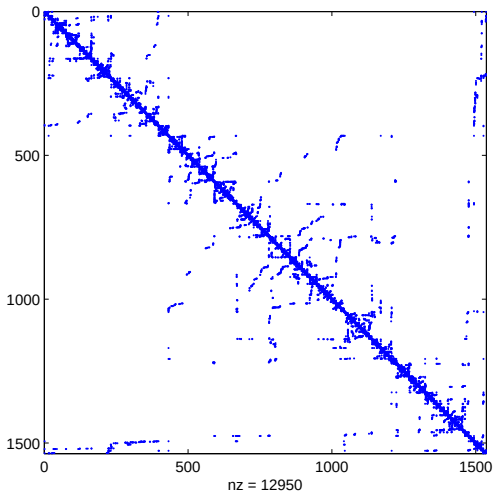

Fig. 4. The communication pattern of ART during a large-scale cosmology simulation with 1536 processes.

Since the computational mesh and cosmological objects evolve dynamically during a cosmology simulation, the workload distribution between processes changes. In order to ensure load balance, ART regularly examines the workload distribution during simulation, and performs domain decomposition to re-balance workload when necessary.

C. Communication Pattern

With the SFC-based domain decomposition, each process of ART mainly performs computation for its local computational domain, and communicates with other processes to get the boundary information, which are the data associated with the external boundary cells of each process. It is worth noting that each process only keeps the data associated with its local computational domain, enabling a fully parallel solution for both computation and memory. Updating the boundary information is the dominating communication routine of ART. It is implemented by using MPI_Irecv() and MPI_Isend() followed by MPI_Waitall(). As the physics computation of each process depends on updated boundary data, such communication cannot be overlapped with computation to reduce runtime.

Generally, each process only communicates with relatively small number of processes whose computational domain is nearby, and the amount of communication between two processes is mainly dependent on the number of boundary cells between their computational domains. Fig. 4 shows the communication pattern of ART for a production simulation with 1536 processes. Each blue dot at (i, j) represents the communication between process i and j, and "nz" denotes the total number of blue dots, i.e., the total number of communicating process pairs. Clearly, each process only communicates with a few other processes, and most communication is between neighboring processes since most blue dots are along the diagonal. The communication pattern may vary for different mesh structure and different number of processes,

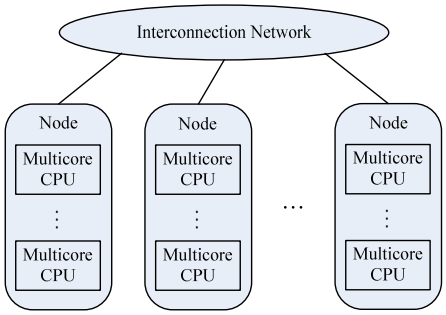

Fig. 5. Interconnected multiprocessor clusters with multicore CPUs on each node.

yet the sparse and diagonal dominant property always holds due to the spatial locality provided by the SFC-based domain decomposition.

For such sparse and diagonal dominant communication pattern, the system default mapping, which assigns continuous processes to multiprocessor nodes, would provide good performance as most communication is within nodes. However, the default mapping can at least be improved by using efficient task mapping techniques to reduce the amount of traffic and contention in the network. As the 3D adaptive mesh for modeling the universe in the ART code is highly irregular, conventional geometric mapping algorithms [6] are not applicable for mapping ART processes. In this paper, we investigate graph methods for task mapping of ART. The task mapping procedure is designed to integrate with domain decomposition, and it generates a mapping of ART processes (i.e., decomposed subdomains) to compute nodes and cores.

III. Hierarchical Task Mapping

A. Problem Statement

We consider the problem of mapping parallel ART processes onto interconnected multiprocessor clusters as illustrated in Fig. 5. Each node has several multicore CPUs. The communication pattern of ART is represented as a *task graph* $G_t(V_t, E_t)$, which can be extracted from the decomposed subdomains. Each vertex in V_t denotes a process, and each edge $(u, v) \in E_t$ represents the communication between process u and v. A weight $c(u, v)$ is introduced for each edge to denote the amount of communication in bytes between respective processes. The topology of a multicore cluster is characterized by a *topology graph* $G_p(V_p, E_p)$, where each vertex in V_p represents a node, and each edge in E_p denotes the link between respective nodes. We also introduce edge weight for the topology graph to represent the distance in hops between nodes, so that both direct and indirect links can be modeled properly.

The mapping of tasks onto nodes is specified by a function $\phi : V_t \rightarrow V_p$. As each node can accommodate multiple processes, without loss of generality, we assume that $|V_t|$ is a multiple of $|V_p|$, so that each node is assigned $\frac{|V_t|}{|V_p|}$ processes. There are at least four metrics to evaluate the quality of a mapping, including the widely-used *hop-bytes* and *dilation*, and recently proposed *maximum interconnective message size* [19] and *the worst-case congestion* [20]. Hop-bytes represents the total amount of inter-node communication. It is computed as the total amount of inter-process communication weighted by the distance in hops between corresponding mapped nodes, i.e.,

$$\textit{hop-bytes}(\phi) \triangleq \sum_{\forall (u,v) \in E_t} c(u,v) d\big(\phi(u), \phi(v)\big),$$

where $d(\phi(u), \phi(v))$ is the hop distance, usually measured by the length of the shortest path between node $\phi(u)$ and $\phi(v)$. The dilation of an edge $(u, v) \in E_t$ is the length of the shortest path connecting $\phi(u)$ and $\phi(v)$ in G_p. The dilation of a mapping is the maximum dilation among all edges in G_p, and it measures the most stretched edge. The maximum interconnective message size is the maximum size of messages transmitted between nodes. The congestion of a link in the network is measured by the amount of traffic on that link divided by the link capacity, and the worst-case congestion over all links denotes the worst-case contention in the network. In general, finding the optimal mapping ϕ that minimizes one or more of the aforementioned metrics is an NP-hard or NP-complete problem. Many heuristic algorithms have been proposed to find a fairly good mapping in reasonable amount of runtime, including greedy approaches [4], [20], geometric methods [6], recursive bipartitioning [20]–[22], graph similarity [20], and so on.

To the best of our knowledge, most studies focus on mapping tasks onto nodes (i.e., a single-level mapping), while the mapping of tasks within nodes is undefined. A proper inter-node mapping typically reduces the communication between nodes by grouping most heavily communicating processes within nodes, leading to large amount of intra-node communication. If inter-node mapping has been optimized properly, then it becomes critical to optimize the intra-node mapping by exploiting the topology within a node. In this paper, we optimize both inter-node mapping and intra-node mapping in a hierarchical manner.

B. Proposed Approach

Typically, the inter-node communication is more expensive than intra-node communication. Likewise, within a node, the inter-socket communication is often more expensive than intra-socket communication [1]. To fully exploit these communication characteristics on multicore clusters, we propose to perform task mapping hierarchically in two phases:

- First, perform inter-node mapping;
- Second, perform intra-node mapping within each multiprocessor node.

In the first phase, the mapping of tasks onto nodes can be derived by using conventional task mapping techniques. In the second phase, novel technique is required to map tasks onto multicore CPUs within each node.

[1]This is our observation from the experiments on several production multiprocessor clusters. The performance of intra-node communication is determined by specific MPI implementation.

Algorithm 1 Recursive Mapping

Input: task graph $G_t(V_t, E_t)$, topology graph $G_p(V_p, E_p)$.
Output: mapping $\phi : V_t \rightarrow V_p$.

```
1  recursive_mapping(V_t, V_p)
2  {
3    if (|V_p| = 1) {
4      φ(u) = V_p, ∀ process u ∈ V_t;
5      return;
6    }
7    (V_t0, V_t1) ← bipartition(G_t(V_t, E_t));
8    (V_p0, V_p1) ← bipartition(G_p(V_p, E_p));
9    Calculate C̄_0, C̄_1, D̄_0, D̄_1;
10   if (C̄_0 D̄_0 + C̄_1 D̄_1 ≤ C̄_0 D̄_1 + C̄_1 D̄_0) {
11     recursive_mapping(V_t0, V_p0);
12     recursive_mapping(V_t1, V_p1);
13   }
14   else {
15     recursive_mapping(V_t0, V_p1);
16     recursive_mapping(V_t1, V_p0);
17   }
18 }
```

Fig. 6. The recursive bipartitioning algorithm for inter-node mapping.

1) Inter-Node Mapping: We employ the recursive bipartitioning heuristic [20]–[22] for mapping ART processes onto multiprocessor nodes, aiming at minimizing hop-bytes. This approach solves the task mapping problem in a divide-and-conquer manner. It performs recursive bipartitioning for both task graph and topology graph, and maps subsets of processes to subsets of nodes until a final mapping is obtained. Many topologies and communication patterns can be handled by recursive bipartitioning, and it is proved to be a successful task mapping technique in the software package SCOTCH [23].

Fig. 6 presents our recursive bipartitioning algorithm for inter-node mapping. Recall that both task graph and topology graph are weighted, and their edge weights represent the amount of communication in bytes between processes and the distance in hops between nodes, respectively. In order to reduce hop-bytes, it is critical to map heavily communicating processes onto the same multiprocessor node, or at least nearby nodes. To achieve this goal, the task graph is partitioned with the minimum edge-cut, while the topology graph is split with the maximum edge-cut. In each step of the algorithm, The resulting subsets of processes (V_{t0}, V_{t1}) and subsets of nodes (V_{p0}, V_{p1}) can be mapped in two ways: the direct mapping $V_{t0} \rightarrow V_{p0}, V_{t1} \rightarrow V_{p1}$; and the exchanged mapping $V_{t0} \rightarrow V_{p1}, V_{t1} \rightarrow V_{p0}$. To choose a proper mapping, we heuristically estimate the cost of these two mappings. Let C_u be the amount of communication in bytes associated with process u, D_i be the aggregate distance between node i and other nodes. Let $\overline{C}_k$ and $\overline{D}_k$ be the average C_u and D_i for the subsets of processes V_{tk} and the subsets of nodes V_{pk} $(k = 0, 1)$, respectively.

$$
\begin{aligned}
C_u &= \sum_{\forall (u,v) \in E_t} c(u,v), \\
D_i &= \sum_{\forall \text{ node } j \neq i} d(i,j), \\
\overline{C}_k &= \frac{1}{|V_{tk}|} \sum_{\forall u \in V_{tk}} C_u, \\
\overline{D}_k &= \frac{1}{|V_{pk}|} \sum_{\forall i \in V_{pk}} D_i.
\end{aligned}
$$

The cost of the direct mapping is estimated by $(\overline{C}_0\overline{D}_0 + \overline{C}_1\overline{D}_1)$, while that of the exchanged mapping is $(\overline{C}_0\overline{D}_1 + \overline{C}_1\overline{D}_0)$. This estimated cost can be considered as a prediction of hop-bytes. The mapping with smaller estimated cost is selected in order to minimize hop-bytes.

Theorem 1: The time complexity of recursive bipartitioning is $O\big((|E_t| + |E_p|) \log |V_p|\big)$, and the time complexity for cost estimation is $O\big((|V_t| + |V_p|) \log |V_p|\big)$ based on precomputed C_u and D_i.

Proof: The multilevel k-way partitioning scheme [24] computes a bipartition of a graph $G(V, E)$ in $O(|E|)$ time. The depth of recursive bipartitioning is $\lceil \log_2 |V_p| \rceil$, and the size of the graph is decreased by half in each step. Hence, the total runtime for partitioning the task graph $G_t(V_t, E_t)$ is $\sum_{k=0}^{\lceil \log_2 |V_p| \rceil - 1} 2^k O(|E_t|)/2^k = O(|E_t| \log |V_p|)$. Similarly, the topology graph $G_p(V_p, E_p)$ is recursively bipartitioned in $O(|E_p| \log |V_p|)$ time. Thus the overall runtime of recursive bipartitioning is $O\big((|E_t| + |E_p|) \log |V_p|\big)$. For cost estimation, both C_u and D_i can be precomputed and then they are used to calculate $\overline{C}_k$ and $\overline{D}_k$ in different recursive mapping calls. The C_u of all processes can be computed in $O(|E_t|)$ time. The distance between a pair of nodes can be obtained by using platform-dependent techniques, e.g., the pairwise node distance in a 3D mesh or 3D torus network can be computed from the coordinates. The D_i of all nodes can be calculated in $O(|V_p|^2)$ time using pairwise node distances. In the recursive mapping procedure, the runtime for computing $\overline{C}_k$ and $\overline{D}_k$ is $\sum_{k=0}^{\lceil \log_2 |V_p| \rceil - 1} 2^k O(|V_t| + |V_p|))/2^k = O\big((|V_t| + |V_p|) \log |V_p|\big)$. ∎

In order to reduce the runtime of recursive mapping, the original task graph $G_t(V_t, E_t)$ is partitioned into $|V_p|$ equal parts by minimizing inter-part communication, where each part has $\frac{|V_t|}{|V_p|}$ processes. Then an induced task graph $\widehat{G_t}(\widehat{V_t}, \widehat{E_t})$ which represents the communication between groups of processes is used for efficient recursive mapping.

Theorem 2: With the induced task graph $\widehat{G_t}(\widehat{V_t}, \widehat{E_t})$, the time complexity of recursive bipartitioning is $O\big((|\widehat{E_t}| + |E_p|) \log |V_p|\big)$, and the time complexity for cost estimation is $O\big(|V_p| \log |V_p|\big)$.

We use the graph partitioning tool hMETIS [25] for partitioning the original task graph, and the recursive bipartitioning of both the induced task graph and the topology graph. The resulting subsets of processes and subsets of nodes may be unbalanced in some rare cases. A greedy approach is applied to

Algorithm 2 Intra-node Mapping

Input: intra-node task graph $\widetilde{G_t}(\widetilde{V_t}, \widetilde{E_t})$.
Output: mapping of processes onto multicore CPUs.

```
1  Partition intra-node task graph into ncpus equal parts P_i (0 ≤ i < ncpus);
2  Map processes in P_i onto CPU i;
3  Loop
4    Identify the edge (u, v) ∈ E_t leading to MIMS, i.e., u ∈ P_i, v ∈ P_j, i ≠ j, MIMS = c(u, v);
5    If the minimum IMS of exchanging a pair of processes to group both process u and v onto either CPU i or j is smaller than MIMS;
6      Exchange the pair of processes with the minimum IMS;
7    Else
8      Break;
9    End If
10 End Loop
```

Fig. 7. The algorithm for intra-node mapping by minimizing the maximum inter-socket message size (MIMS).

achieve balanced bipartition by moving the vertex which leads to the optimal edge-cut. In practice, if $|V_p| \neq 2^k$, the number of nodes in some step of the recursive mapping will be odd, then the cost estimation is not required, and a direct mapping which maps process groups to equal number of nodes is adopted. The mapping produced by the recursive bipartitioning algorithm may be further improved to reduce hop-bytes. The local search algorithm in [26], and the heuristic in [20] with the threshold accepting technique can be employed to further optimize the mapping, but such mapping optimization has not been exploited in this work.

2) Intra-Node Mapping: The motivation of intra-node mapping is to exploit the performance gap between intra-socket communication and inter-socket communication for performance optimization. For each multiprocessor node, we choose to minimize *the maximum inter-socket message size (MIMS)*, which is computed as

$$\text{MIMS} \triangleq \max_{(u,v)\in E_t} c(u,v),$$

subject to the condition that process u and v are on different CPU sockets of the same node.

MIMS is the maximum size of process-pairwise bidirectional messages transmitted between CPU sockets. The resulting mapping places heavily communicating processes on the same multicore CPU.

Fig. 7 shows the sketch of our algorithm for intra-node mapping. It utilizes the intra-node task graph $\widetilde{G_t}(\widetilde{V_t}, \widetilde{E_t})$ which represents the communication between processes within a multiprocessor node, and maps processes onto multicore CPUs. We assume that the number of processes $|\widetilde{V_t}|$ is a multiple of $ncpus$ (number of CPUs), so that each CPU is assigned $\frac{|\widetilde{V_t}|}{ncpus}$ processes. The intra-node mapping is performed in two steps:

- First, initial mapping by graph partitioning;
- Second, fine tuning with a greedy heuristic.

The processes are partitioned into $ncpus$ (number of CPUs) equal parts $P_i (0 \leq i < ncpus)$ by minimizing inter-part communication, where each part is mapped onto the corresponding CPU. In each iteration of fine tuning, the edge $(u, v) \in E_t$ resulting in MIMS is identified, i.e., $u \in P_i, v \in P_j, i \neq j, \text{MIMS} = c(u, v)$. Then we evaluate whether we can group both processes u and v onto CPU i or j by exchanging a pair of processes to reduce MIMS. The resulting inter-socket message size (IMS) of exchanging process $u \in P_i$ and a process $w \in P_j \setminus \{v\}$ can be computed as

$$\text{IMS}(u \leftrightarrow w) = \max\Big(\max_{x\in P_i, x\neq u} c(u,x), \max_{x\in P_j, x\neq w} c(w,x)\Big),$$

Likewise, the resulting IMS of exchanging process $v \in P_j$ and a process $w \in P_i \setminus \{u\}$ can also be computed. The minimum IMS of exchanging a pair of processes can be derived by evaluating all possible process pairs:

$$(u \leftrightarrow w), u \in P_i, w \in P_j \setminus \{v\},$$
$$(w \leftrightarrow v), w \in P_i \setminus \{u\}, v \in P_j.$$

If it is less than MIMS, then we exchange the pair of processes with the minimum IMS. Otherwise, the algorithm terminates.

The initial mapping can be obtained in $O(|\widetilde{E_t}|)$ time by using the multilevel k-way partitioning algorithm [24]. In each iteration of fine tuning, the edge (u, v) leading to MIMS can be identified in $O(|\widetilde{E_t}|)$ time. In order to evaluate the resulting IMS of exchanging process pairs, for each process in P_i and P_j, the maximum amount of communication with another process in the same part need to be computed. This procedure takes $O(|\widetilde{E_{ti}}| + |\widetilde{E_{tj}}|)$ time, where $\widetilde{E_{ti}}$ and $\widetilde{E_{tj}}$ be the set of edges representing the communication between processes within P_i and P_j, respectively. Then it requires $O(\frac{|\widetilde{V_t}|}{ncpus})$ comparisons to find the minimum IMS of exchanging two processes.

Theorem 3: The overall time complexity of each iteration of fine tuning is $O(|\widetilde{E_t}| + |\widetilde{E_{ti}}| + |\widetilde{E_{tj}}| + \frac{|\widetilde{V_t}|}{ncpus})$.

hMETIS [25] is employed to partition the intra-node task graph into balanced parts for initial mapping, which is further improved through fine tuning. It is worth noting that the initial mapping has the minimum total amount of inter-socket communication, and the fine tuning procedure usually terminates in a few iterations.

IV. Performance Evaluation

A. Experimental Setup

Experiments are carried out on two production multiprocessor clusters, NICS Kraken [27] and TACC Ranger [28], which have different network topologies. Kraken is a Cray XT5 system with a 3D torus interconnect topology. It is comprised of $9,408$ compute nodes and each node contains two 2.6 GHz six-core AMD Opteron processors. Ranger is a Sun Constellation Linux cluster with a full-CLOS fat-tree topology. It has a total of $3,936$ compute nodes and each node has four 2.3 GHz quad-core AMD Opteron processors.

TABLE I
AVERAGE INTRA-SOCKET AND INTER-SOCKET COMMUNICATION TIME (PINGPING)

No. of Bytes	No. of Repetitions	Kraken (us)			Ranger (us)		
		Intra-Socket	Inter-Socket	Difference	Intra-Socket	Inter-Socket	Difference
0	1000	0.56	0.81	**43.38%**	1.27	1.76	38.85%
1	1000	0.58	0.74	27.57%	1.33	1.78	**33.85%**
2	1000	0.59	0.74	26.96%	1.32	1.78	**34.78%**
4	1000	0.60	0.75	25.26%	1.32	1.79	**35.45%**
8	1000	0.62	0.74	20.89%	1.35	1.79	**32.68%**
16	1000	0.61	0.75	21.90%	1.39	1.81	**30.73%**
32	1000	0.64	0.83	28.95%	1.35	1.78	**31.15%**
64	1000	0.65	0.84	30.43%	1.35	1.85	**36.92%**
128	1000	1.09	1.31	20.06%	1.43	1.88	**31.63%**
256	1000	1.16	1.39	19.82%	1.55	2.00	**28.95%**
512	1000	1.37	1.64	19.84%	1.92	2.36	**22.76%**
1024	1000	1.75	2.08	19.19%	2.48	3.04	**22.72%**
2048	1000	2.57	3.02	**17.77%**	3.69	4.34	17.63%
4096	1000	4.39	5.17	**17.76%**	6.31	6.99	10.77%
8192	1000	7.30	8.87	**21.45%**	11.59	12.22	5.42%
16384	1000	14.64	16.95	15.76%	24.64	29.12	**18.17%**
32768	1000	27.86	32.63	17.13%	46.25	54.37	**17.54%**
65536	640	53.92	64.14	**18.94%**	89.79	104.73	16.64%
131072	320	36.74	38.11	3.74%	168.94	201.66	**19.37%**
262144	160	71.09	71.76	0.93%	257.23	342.63	**33.20%**

A communication benchmark similar to IMB (Intel MPI Benchmarks) [29] is designed to measure the intra-socket communication time and the inter-socket communication time for performance analysis. Note that IMB cannot be employed for this purpose because we would like to test all possible communicating pairs of processors. We are particularly interested in the performance of PingPing, since the communication between ART processes can be viewed as concurrent PingPing operations between many process pairs.

To evaluate the performance of the proposed hierarchical task mapping algorithm, we extract the communication part of a production ART simulation (with a box of 36 comoving Mpc on a side and the uniform top level mesh of 256^3 root cells) for tests. For comparison, we evaluate five different mapping mechanisms: (1) the system default mapping, which is topology-agnostic; (2) an optimized mapping using MPI topology function MPI_GRAPH_CREATE [30]; (3) the pure inter-node mapping using the algorithm in Section III-B1; (4) the pure intra-node mapping using the algorithm in Section III-B2; (5) the hierarchical mapping integrating both inter-node mapping and intra-node mapping. Hope-bytes, the maximum inter-socket message size (MIMS) and communication time are used as evaluation metrics.

All the experiments were conducted in production mode without dedicated nodes, and there are other users sharing the interconnection network. For each set of experiments with a particular number of processes, we get the topology information between nodes at runtime, generate different mappings by using the proposed algorithms, and run all the tests with different mappings in a single batch script. It is worth noting that different runs often get nodes with different pairwise distances, and the interference of other running applications may also be different. Hence, the results obtained from different runs may not be comparable.

B. Results

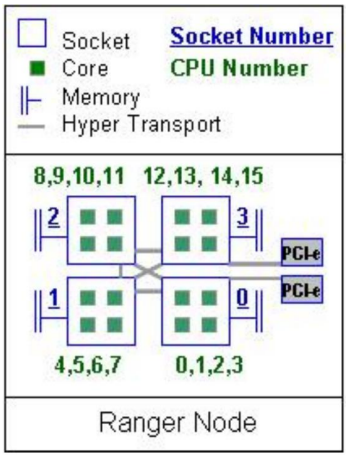

Fig. 8. The intra-node topology of Ranger (from TACC Ranger website [28]).

Table I presents the average intra-socket and inter-socket PingPing communication time over all possible communicating processor pairs. The overhead of inter-socket communication compared to intra-socket communication is reported under the columns "Difference", and the larger difference values between Kraken and Ranger are highlighted in bold. Clearly, Ranger shows a larger performance gap between intra-socket and inter-socket communication for most message

sizes, because it has more complicated intra-node topology. As shown in Fig. 8, there are four interconnected CPU sockets and no direct link exists between socket 0 and 3 on Ranger. Meanwhile, there are only two CPU sockets on each node of Kraken. Such architectural difference results in different communication saving when applying the pure intra-node mapping and the hierarchical mapping (see Fig. 12 and 16).

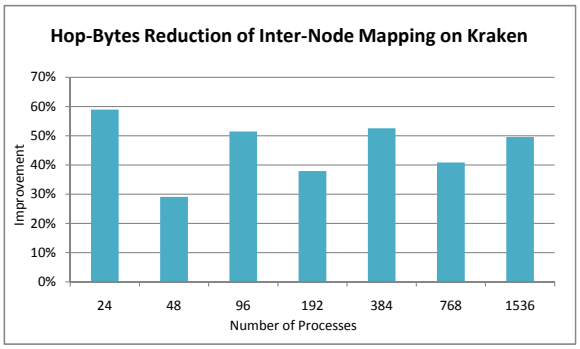

Fig. 9. Comparison of inter-node mapping and default mapping on Kraken in terms of hop-bytes.

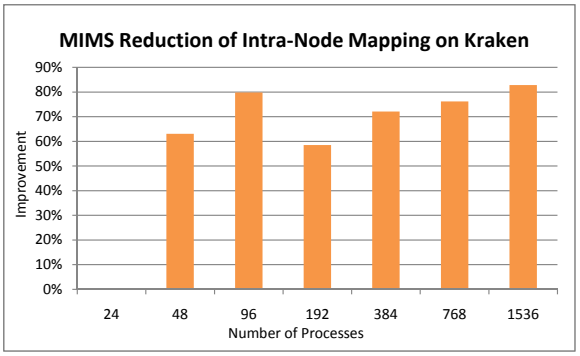

Fig. 10. Comparison of intra-node mapping and default mapping on Kraken in terms of MIMS (the maximum inter-socket message size).

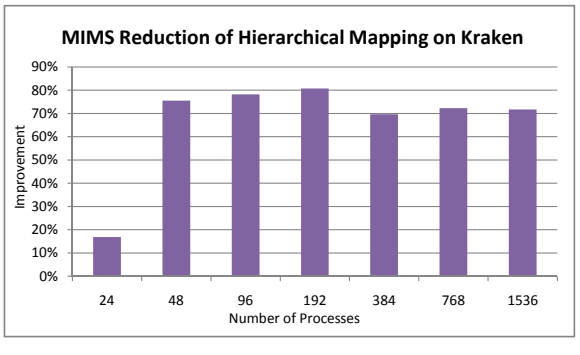

Fig. 11. Comparison of hierarchical mapping and inter-node mapping on Kraken in terms of MIMS (the maximum inter-socket message size).

Fig. 9 compares of the pure inter-node mapping and the system default mapping on Kraken in terms of hop-bytes. The inter-node mapping algorithm (listed in Fig. 6) effectively reduces hop-bytes for all the test cases, and the maximum reduction is up to 59%.

Fig. 10 compares the pure intra-node mapping and the system default mapping on Kraken in terms of MIMS. and Fig. 11 compares the hierarchical mapping and the pure inter-node mapping on Kraken in terms of MIMS. The intra-node

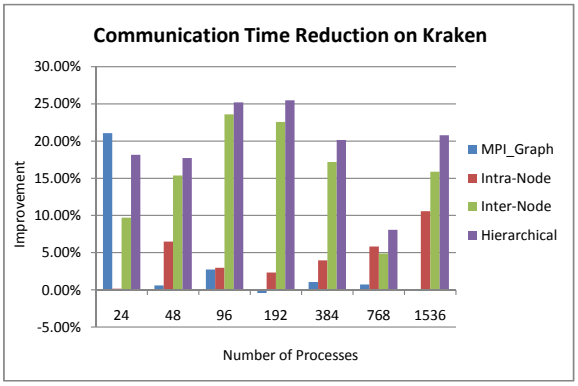

Fig. 12. Communication time reduction of different mapping mechanisms compared to default mapping on Kraken.

mapping algorithm (listed in Fig. 7) largely reduces MIMS by up to 83%. For the first test case with 24 processes, the system default mapping happens to have the minimum MIMS, and the MIMS of the pure inter-node mapping is also close to the minimum, so no MIMS reduction or only limited reduction can be achieved.

The communication time reduction of different mapping mechanisms compared to the system default mapping is illustrated in Fig. 12, where "MPI_Graph", "Intra-Node", "Inter-Node" and "Hierarchical" represent MPI topology mapping, the pure intra-node mapping, the pure inter-node mapping and the hierarchical mapping, respectively. MPI_Graph does not achieve much performance improvement except the first test case with 24 processes, and it fails for the largest test with 1,536 processes. The pure intra-node mapping often provides minor performance improvement, while the pure inter-node mapping has much better performance. In contrast, the hierarchical mapping always outperforms both the pure intra-node mapping and the pure inter-node mapping, achieving communication time reduction by up to 25%.

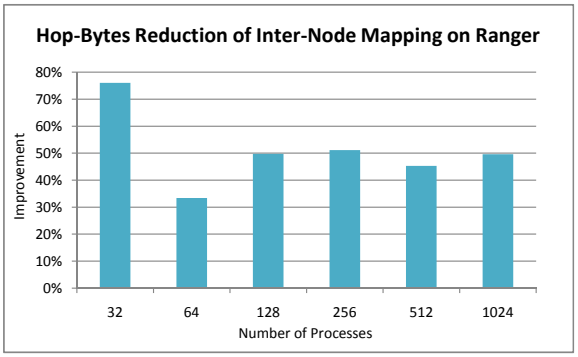

Fig. 13. Comparison of inter-node mapping and default mapping on Ranger in terms of hop-bytes.

The experiments on Ranger have similar performance results as illustrated in Fig. 13 to 16. For all the test cases, the inter-node mapping algorithm (listed in Fig. 6) is able to reduce hop-bytes by up to 76%, and the intra-node mapping algorithm (listed in Fig. 7) reduces MIMS by up to 79%. MPI_Graph provides similar performance as the system default mapping. Both the pure intra-node mapping and the

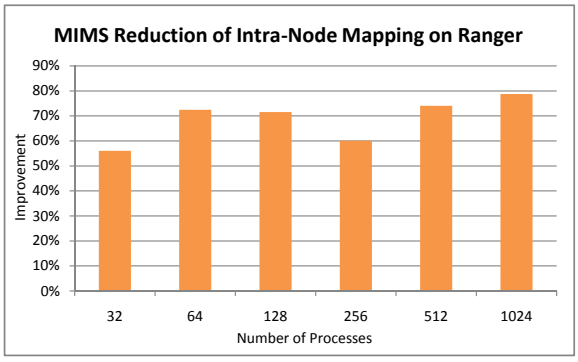

Fig. 14. Comparison of intra-node mapping and default mapping on Ranger in terms of MIMS (the maximum inter-socket message size).

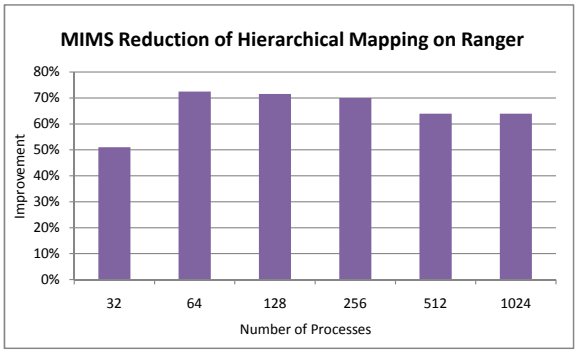

Fig. 15. Comparison of hierarchical mapping and inter-node mapping on Ranger in terms of MIMS (the maximum inter-socket message size).

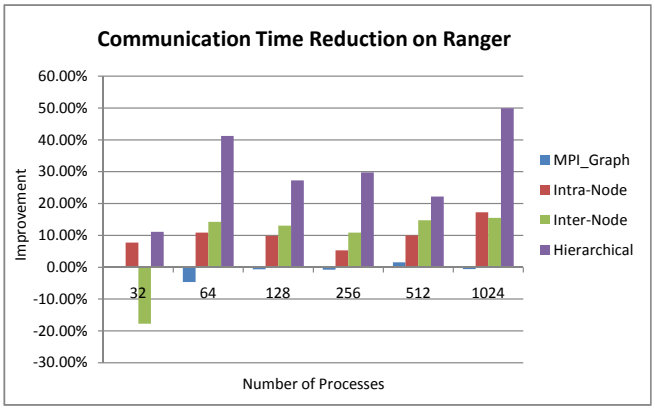

Fig. 16. Communication time reduction of different mapping mechanisms compared to default mapping on Ranger.

pure inter-node mapping can achieve communication time reduction for most test cases. For the test with 32 processes, the pure inter-node mapping results in more communication time despite of reduced hope-bytes. This phenomenon is mainly attributable to the fact that the intra-node PingPing communication can be slower than the nearby inter-node PingPing communication for large message sizes on Ranger. This is due to small memory bandwidth per core on Ranger, and possibly as well as inefficient implementation of intra-node communication algorithms in the mvapich library. The hierarchical mapping often achieves much better performance than both the pure intra-node mapping and the pure inter-node mapping, reducing the communication time by up to 50%.

By comparing Fig. 12 and 16, we can observe that the hierarchical mapping is much more effective on Ranger (with respect to the performance of the pure inter-node mapping), and the pure intra-node mapping generally achieves more communication time reduction (in percentage) on Ranger, because Ranger has a larger performance gap between intra-socket and inter-socket communication as shown in Table I. Basically, such performance gap indicates how critical it is to perform intra-node task mapping, and we can expect that the intra-node task mapping will become increasingly important as more processors are included in each node.

V. Conclusion

In this study, we have presented hierarchical task mapping of cell-based AMR cosmology simulations onto a cluster composed of multiprocessor nodes. Our mapping algorithm contains two phases: an inter-node mapping (i.e., mapping application processes onto nodes) by minimizing hop-bytes and an intra-node mapping (i.e., mapping application processes onto CPUs within each node) by minimizing the maximum inter-socket message size. Experimental results on production HPC platforms demonstrate that the proposed hierarchical mapping greatly outperforms the system default mapping that is topology-agnostic and the optimized mapping using MPI topology function. Our preliminary results show that for cosmology simulations, the proposed hierarchical mapping can reduce their communication time by up to 50%, as compared to the system default mapping. More importantly, this hierarchical approach is not limited to cell-based AMR cosmology simulations, instead the algorithm itself is quite generic and can be applied to many other applications that show similar communication patterns.

Acknowledgement

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

References

[1] T. Plewa, T. Linde, and V. G. Weirs, *Adaptive Mesh Refinement–Theory and Applications*. Berlin: Springer, 2005.
[2] Enzo. [Online]. Available: http://lca.ucsd.edu/portal/software/enzo
[3] A. V. Kravtsov, A. A. Klypin, and A. M. Khokhlov, "Adaptive refinement tree: A new high-resolution N-body code for cosmological simulations," *The Astrophysical Journal Supplement Series*, vol. 111, pp. 73–94, Jul. 1997.
[4] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kale, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2006.
[5] A. Bhatele and L. V. Kale, "Application-specific topology-aware mapping for three dimensional topologies," in *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008, pp. 1–8.
[6] A. Bhatele, "Automating topology aware mapping for supercomputers," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Urbana, Aug. 2010.
[7] Q. Meng, J. Luitjens, and M. Berzins, "Dynamic task scheduling for the Uintah framework," in *Proc. IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*, 2010, pp. 1–10.

[8] J. Luitjens and M. Berzins, "Improving the performance of Uintah: A large-scale adaptive meshing computational framework," in *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010, pp. 1–10.

[9] A. V. Kravtsov, "High-resolution simulations of structure formation in the universe," Ph.D. dissertation, New Mexico State University, Las Cruces, Dec. 1999.

[10] J. Wu, R. E. Gonzalez, Z. Lan, N. Y. Gnedin, A. V. Kravtsov, D. H. Rudd, and Y. Yu, "Performance emulation of cell-based AMR cosmology simulations," in *Proc. IEEE International Conference on Cluster Computing (CLUSTER)*, 2011, pp. 8–16.

[11] Y. Yu, D. H. Rudd, Z. Lan, N. Y. Gnedin, A. V. Kravtsov, and J. Wu, "Improving parallel IO performance of cell-based AMR cosmology applications," in *Proc. IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2012, pp. 933–944.

[12] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree N-body algorithm," in *Proc. ACM/IEEE conference on Supercomputing*, 1993, pp. 12–21.

[13] A. M. Khokhlov, "Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations," *J. Comput. Phys.*, vol. 143, pp. 519–543, Jul. 1998.

[14] A. Patra and J. T. Oden, "Problem decomposition for adaptive hp finite element methods," *Computing Systems in Engineering*, vol. 6, no. 2, pp. 97–109, 1995.

[15] A. R. Butz, "Alternative algorithm for Hilbert's space-filling curve," *IEEE Transactions on Computers*, vol. C-20, no. 4, pp. 424–426, Apr. 1971.

[16] Flash. [Online]. Available: http://flash.uchicago.edu/site/

[17] J. Steensland, S. Chandra, and M. Parashar, "An application-centric characterization of domain-based SFC partitioners for parallel SAMR," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 12, pp. 1275–1289, Dec. 2002.

[18] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, "Extreme-scale AMR," in *Proc. ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2010, pp. 1–12.

[19] I.-H. Chung, C.-R. Lee, J. Zhou, and Y.-C. Chung, "Scalable communication-aware task mapping algorithms for interconnected multicore systems," in *Proc. IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2011, pp. 759–764.

[20] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proc. the international conference on Supercomputing (ICS)*, 2011, pp. 75–84.

[21] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task allocation onto a hypercube by recursive mincut bipartitioning," in *Proc. the third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues - Volume 1*, ser. C3P, 1988, pp. 210–221.

[22] F. Pellegrini, "Static mapping by dual recursive bipartitioning of process architecture graphs," in *Proc. the Scalable High-Performance Computing Conference*, 1994, pp. 486–493.

[23] F. pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *High-Performance Computing and Networking*, ser. Lecture Notes in Computer Science, vol. 1067, 1996, pp. 493–498.

[24] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 96–129, Jan. 1998.

[25] hMETIS. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview

[26] I.-H. Chung, C.-R. Lee, J. Zhou, and Y.-C. Chung, "Hierarchical mapping for HPC applications," in *Proc. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, 2011, pp. 1815–1823.

[27] The NICS Kraken Website. [Online]. Available: https://www.xsede.org/web/guest/nics-kraken

[28] The TACC Ranger Website. [Online]. Available: http://www.tacc.utexas.edu/user-services/user-guides/ranger-user-guide

[29] Intel MPI Benchmarks. [Online]. Available: http://software.intel.com/en-us/articles/intel-mpi-benchmarks/

[30] MPI: A message-passing interface standard. version 2.2. [Online]. Available: http://www.mpi-forum.org/