Scalable Communication-aware Task Mapping Algorithms for Interconnected Multicore Systems

I-Hsin Chung
IBM T.J. Watson Research Center
Yorktown Heights, NY USA
ihchung@us.ibm.com

Che-Rung Lee Jiazheng Zhou Yeh-Ching Chung
National Tsing-Hua University
Hsin-Chu, Taiwan
{cherung,jzzhou,ychung}@cs.nthu.edu.tw

***Abstract*—Communication-aware task mapping algorithms, which map parallel tasks onto processing nodes according to the communication patterns of applications, are essential to reduce the communication time in modern high performance computing. In this paper, we design algorithms specifically for interconnected multicore systems, whose architectural property, namely small number of cores per node, large number of nodes, and large performance gap between the communication within a multicore and among multicores, had brought new challenges and opportunities to the mapping problem. Let k be the number of cores per multicore and n be the number of tasks. We consider the practical case that $k \ll n$ for $k = 2, 4$, and 6. The designed algorithms are optimal for the mapping measurement, called Maximum Interconnective Message Size (MIMS), and of time complexity merely $O(m \log m)$ for m communication pairs. Thus, they are highly scalable for large applications. We had experimented the algorithms on the IBM Blue Gene/P system for two synthetic benchmarks and two applications. The results show good communication performance improvement.**

I. Introduction

Scientific computing had become the third pillar in science in addition to theory and experimentation. To fulfill the strong demands for larger and more complex simulations, massively parallel systems that possess hundred thousands processing nodes were built and becoming the major trend of computational facilities. However, the rapid increase of processing nodes and the exponentially enlarged speed gap between communication and computation have made communication the performance bottleneck. As the result, reducing the communication overhead becomes the new paradigm of performance enhancement.

Communication-aware task mapping, which maps parallel tasks onto processing nodes according to the communication patterns of applications, is one of the important method to achieve this goal. Many of the mapping algorithm are designed for specific host machine topologies, such as hypercube, mesh network, or switch based network [10], but only few of them is designed for interconnected multicore systems, whose architectural properties had brought new challenges and opportunities to the communication-aware mapping problem. First, the number of processing nodes is massive; while the number of cores per processing node is small. Second, the communication cost among processing nodes is much larger than that within a multicore.

In this paper, we investigate the mapping algorithms for this system. We call the tasks placed in a multicore processor a *pack*, and the process that partitions tasks into packs *packing*. Assume k is the number of cores per multicore and n is the total number of tasks. For such kind of systems, k is usually very small, like two or four, but n can be extremely large. We define a new metric to evaluate the mapping quality of packing, called Maximum Interconnective Message Size (MIMS), which is the maximum total message size communicated between two tasks in different multicore processors. The purpose of this metric is try to utilize the low communication overhead of tasks with multicore as much as possible. We propose a framework of algorithms to find the optimal mappings minimizing the MIMS, and specify how the framework is implemented for $k = 2, 4$ and 6. We verified this idea on the IBM Blue Gene/P system using two synthetic benchmarks and two applications.

978-0-7695-4538-7/11 $26.00 © 2011 IEEE
DOI 10.1109/HPCC.2011.108

The rest of the paper is organized as follows: Section II reviews related work of mappings. Section III introduces the packing algorithms, as well as the analysis of their correctness and time complexity. Section IV presents the experiment and the evaluation results. Section V discusses and concludes the technical details and the limitations of the proposed method.

II. Related Work

The problem of mapping application tasks onto a targeted hardware physical topology has been considered as a graph embedding problem [4]. In general, an embedding of a guest graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$ is a one-to-one mapping ϕ from V_G to V_H. Graph embedding has been studied and applied to optimize VLSI circuits [8], [9]. The graph embedding for VLSI circuits tries to minimize the longest path where the mapping in HPC tries to reduce the point-to-point communication time.

The quality of the embedding can be measured by various cost functions. First, the *dilation* of an edge $(u, v) \in E_G$ is the shortest path in H that connects $\phi(u)$ and $\phi(v)$. The dilation of an embedding is the maximum dilation over all edges in E_G. Second, the *expansion* of an embedding is $|V_H|/|V_G|$. In other words, the dilation of an embedding measures the worst stretched edge and the expansion measures the relative size of the guest graph. Another measurement of the mapping is the hop-byte [12]. Let $w(u, v)$ be the size of messages transferred on an edge $(u, v) \in E_G$, and $d(\phi(u), \phi(v))$ be the distance, usually measured by the number of hops, of the shortest path in H that connects $\phi(u)$ and $\phi(v)$. The hop-byte of an embedding is computed as

$$\sum_{(u,v)\in E_G} w(u, v)d(\phi(u), \phi(v)). \tag{1}$$

In [11], authors take all possible routes between $\phi(u)$ and $\phi(v)$ into account and utilizes the idea of equivalent resistance to compute the equivalent distance for $d(\phi(u), \phi(v))$.

Algorithms designed to optimize different metrics have dissimilar assumptions. For dilation and expansion, the connectivity information of tasks, meaning the pairs of communicated tasks, and the topology of the host machine need to be known. One of the well studied method is the utilization of space filling curves, which improve proximity by mapping the geometric wireframe connected tasks onto the host machine. The paper [3] extends the concept of space filling curves to space filling surfaces. It describes three different classes of space filling surfaces and calculate the distance between facets. Another type of algorithm develops topology mapping libraries [12], [6], whose mapping techniques are based on topology-aware heuristics.

For the hop-byte metric, various techniques were proposed. The first kind of algorithms are search-based optimization methods, such as greedy method, local search, genetic algorithms, or simulated annealing [5]. The second kind of algorithms uses graph-partitioning, which clusters communication intensive tasks together and minimizes the communication cost among partitions. Various methods of graph-partition were proposed, such as recursive mincut, normalized cut[7], etc.

Mapping algorithm designed for scalability purpose is also considered in [7], the HMA algorithm. The basic algorithm of the HMA is the optimization method, and it uses two techniques to improve the scalability and convergence: the task partition and initial mapping. The task partition clusters tasks into equal size groups, called supernodes. By doing so, one can apply the algorithm recursively, which can significantly reduce the time complexity. The initial mapping uses topology-aware techniques to provide better initial guesses to accelerate the convergence and the quality of the optimization method.

III. The packing algorithms

The goal of the packing algorithm is to partition tasks into packs, each of which is a group of k tasks for a k-cores multicore processor. Here we assume that the number of tasks n is a multiple of k, and $k \geq 2$. The criterion of packing is formally defined as follows. Let C be a matrix whose element $C(i, j)$ represents the total message size transmitted between task i and task j, measured for the entire execution. To simplify the discussion, we assume C is symmetric.

Here we give definitions to the problem:

Definition 1: A pack is a set of tasks. A k-pack is a pack of cardinality exact k.

Definition 2: Let V be a set of tasks, and S be a set of packs. A packing is a function $f : V \to S$ that maps each task in V to a pack.

Definition 3: A packing is called *k-feasible*, or simply called feasible, if all packs are k-packs.

In another word, a feasible packing partitions n tasks into (n/k) k-packs.

In this paper, we propose a new metric to measure the quality of a packing, called Maximum Interconnective Message Size (MIMS), which is defined below

$$\text{MIMS}(f) = \max_{\substack{f(v_i) \neq f(v_j) \\ \forall v_i, v_j \in V}} C(i,j). \tag{2}$$

By that, the object of the packing can be defined as

$$\min_{f:V \to S_k} \max_{\substack{f(v_i) \neq f(v_j) \\ \forall v_i, v_j \in V}} C(i,j). \tag{3}$$

Simply speaking, we want the pair of tasks that have the large total communication message size are assigned to the same multicore processor.

Definition 4: A packing f is optimal if it is feasible and MIMS(f) is minimum.

In this section, we will introduce algorithms for $k = 2, 4$ and 6.

A. The packing algorithm for $k = 2$

In this section, we present the framework first. The algorithm is greedy, which means it packs tasks with larger communication cost first. To do that, we need to sort nonzero $C(i,j)$ for all communicating pair v_i and v_j. Then, we try to pack v_i and v_j together from the largest $C(i,j)$. This packing may fail because v_i or v_j may be already packed with other tasks. Therefore, we need to do the check every time. If the packing does make any confliction, then we continue with the next $C(i,j)$. The algorithm is sketched as follows. It can be shown that algorithm 1 will find an optimal packing in terms of MIMS.

For $k = 2$, the feasibility test is just to test if all packs are of size 2, as stated in Algorithm 2.

The time complexity of Algorithm 1 with Algorithm 2 is $O(m \log m)$ for m nonzero $C(i,j)$ because the sorting in Step 1. For $k = 4$ and $k = 6$, the only difference is the how to perform the feasibility test.

B. The packing algorithm for $k = 4$

A correct feasibility check for $k > 2$ can be reduced to the bin-packing problem.

Definition 5: Given m items with sizes $s_1, \ldots, s_m \in \{1 \ldots k\}$, pack them into the fewest number of bins possible, where each bin is of size k.

Algorithm 1 Framework of packing algorithms

```
Input:  $V$, $k$, and $C$
Output: a packing function $f$.
1) Sort $C(i,j)$ in the descending order.
2) While there are unpacked tasks and nonzero $C(i,j)$
    a) Select the largest $C(i,j)$ and pack $v_i$ and $v_j$
    b) Run the feasibility test.
    c) If the test fails
        i) Unpack $v_i$ and $v_j$.
    d) Set $C(i,j) = 0$
3) Pack tasks that are unpacked with the best effort.
```

Algorithm 2 The feasibility test for $k = 2$

```
Input:  A set of packs.
Output: A feasible packing or not.
1) Find the sizes of all packs.
2) If any pack has size larger than 2, return false.
3) Return true.
```

The argument is: if the answer of the bin-packing for packs of different sizes is less or equal to n/k, then the feasibility test should return true. The algorithm that verifies the feasibility based on bin-packing is described in Algorithm 3. We use the term *histogram* to characterize a set of packs, $(n_1, n_2, \ldots, n_k)$, in which n_i denotes the number of size i packs.

Algorithm 3 Feasibility check for Algorithm 1

```
Input:  A set of packs.
Output: A feasible packing or not.
1) If there is a pack of size larger than $k$, return false.
2) Given the Histogram $(n_1, n_2, \ldots, n_k)$ of the current packs, run the bin-packing algorithm.
3) If the number of bins is less than or equal to $n/k$, return true.
4) Otherwise, return false.
```

The following algorithm finds the optimal bin packing in $O(m)$ time for m existing packs.

The time complexity of Algorithm 4 is constant, which makes the time complexity of the entire packing

Algorithm 4 Bin-packing for $k = 4$

```
Input:  Histogram $(n_1, n_2, n_3, n_4)$
Output: The minimum number of bins, of
        capacity 4, for packs of size 2, 3, and 4.
  1) Return $n_4 + n_3 + \lceil n_2/2 \rceil$.
```

Table I
POSSIBLE PACKINGS FOR $k = 6$

class A	$\{6\}, \{5, 1\}^*, \{4, 2\}, \{3, 3\}$
class B	$\{4, 1, 1\}^*, \{3, 2, 1\}$
class C	$\{3, 1, 1\}^*, \{2, 2, 2\}$
class D	$\{2, 2, 1, 1\}$
class E	$\{2, 1, 1, 1, 1\}^*$
class F	$\{1, 1, 1, 1, 1, 1\}^*$

algorithm $O(m \log m)$.

C. The packing algorithm for $k = 6$

The packing algorithm for $k = 6$ is under the same framework described in Algorithm 1 and Algorithm 3, except how to solve the restricted bin-packing problem for $k = 6$. Here we only consider the possible packings to fill a 6-bin, because if there is a bin not full, the packing won't be optimal.

Table I lists all 11 possible full packings for $k = 6$, prioritized into six classes. The classification of packings will be used in the bin-packing algorithm, which tries the possible packings from class A, then class B, and so on. For example, it will try first to find the packings of size 4 and size 2. If there is not enough size 2 packs to match size 4s, it will try to search one size 4 and two size 1s. However, for the possible packings with asterisk marks, if the largest sized pack exists but the packing is not full, it early returns n to indicate infeasibility. The entire process is sketched in Algorithm 5.

The time complexity of the packing algorithm for $k = 6$ is also $O(m \log m)$ because the feasibility test can be done in the constant time.

Algorithm 5 Bin-packing for $k = 6$

```
Input:  Histogram $(n_1, n_2, n_3, n_4, n_5, n_6)$
Output: The minimum number of bins, of capacity 6
  1) For class $i$ = A to F
     a) Find possible packings in class $i$.
     b) Update $(n_1, n_2, n_3, n_4, n_5, n_6)$ by subtracting packed bins.
     c) If there is any negative number in the histogram, return $n$.
  2) Return $n/6$.
```

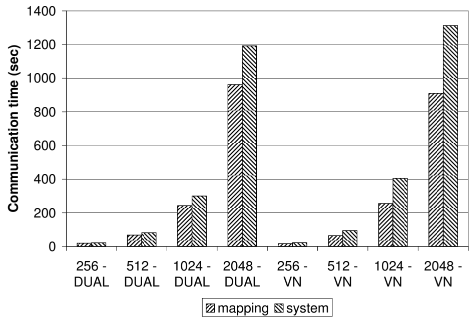

Figure 1. Power Law benchmark communication performance

IV. EXPERIMENTS AND RESULTS

In this section, we use the IBM Blue Gene/P (BG/P) as the platform for experiments. The compute node of the BG/P is a quad-core PowerPC 450 SMP processor running at 850MHz. There are three job execution modes on the BG/P: SMP, Dual and Virtual Node (VN) where 1,2,4 process(es) is(are) assigned to each compute node. In the experiments, we compare the communication performance between our proposed mapping algorithm and numeral order mapping with two synthetic benchmarks and two applications.

A. Synthetic benchmarks

1) Power Law: The Power Law benchmark simulates the communication network that is based on the scale-free network where the degree distribution follows a power law. When there are n nodes in the communication network, we set the i^{th} node's degree to be $n \times (i + 1)^{-0.6}$. The scale-free networks are commonly observed empirically including the World Wide Web, biological networks, traffic networks, and some social networks.

Figure 1 shows the results. In the figure, the "system" label represents the best of "XYZT" and "TXYZ"

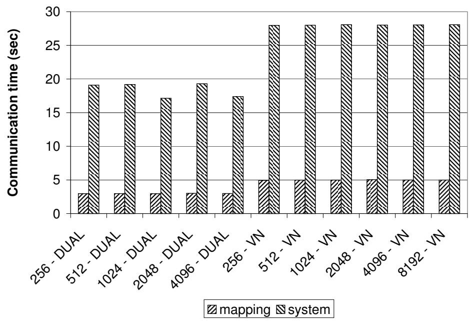

Figure 2. Mesh benchmark communication performance

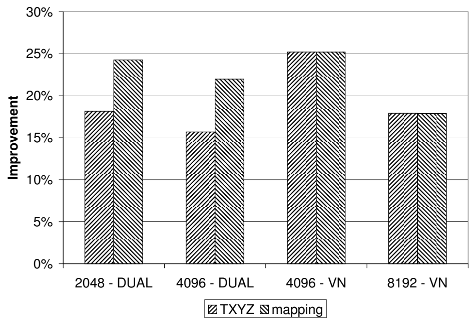

Figure 3. SWEEP3D communication performance

system mappings [1] and the “mapping” label represents the mapping file from our proposed algorithm. The x-axis label means number of MPI ranks and the execution mode. Shorter bars in the figure mean less communication time. The impact of mapping is more obvious as the scale goes up. For the DUAL mode, the proposed mapping improves the communication time about 20% and for the VN mode, the improvement is about 30%.

2) Mesh: The Mesh benchmark is to simulate the finite element method for multilayer structure, in which the mesh in the same layer is more dense than between layers. If we arrange elements in the same layer to a row, and stack rows of different layers, we will have the communication pattern like a two dimensional mesh, but the connectivity of the row is stronger than that within a column. In the benchmark, each pair (i, j) exchange messages with other nodes in the same row $(i, :)$ and the direct neighbors in the adjacent columns $(i-1, j), (i+1, j)$.

Similar to the Power Law benchmark results, the results from the Mesh benchmark show significant improvement as in Figure 2. Since each rank communicate with a group of ranks frequently, it is helpful if these ranks can be mapped onto the cores on the same compute node. The difference is more obvious in the VN mode (quad core) than it is in the DUAL mode (dual core).

[1]On BG/P,XYZ represents the coordinate of a compute node/processor and T is the core ID within the processor. The user may specify either the numeral order using the permutation of XYZT or a mapping file as a mpirun argument to map the tasks on to the cores. The leftmost character in the permutation string is the fastest changing dimension.

B. SWEEP3D

SWEEP3D [2] is a simplified benchmark program that solves a neutron transport problem using a pipelined wave-front method on a two-dimensional process mesh. Input parameters determine problem sizes and blocking factors, allowing for a wide range of message sizes and parallel efficiencies. The communication pattern is the message exchange in the neighborhood. The sweep of the wave-front starts in the upper-left quadrant towards the bottom-right quadrant. The communication in the same wave can be executed simultaneously.

The results are shown in Figure 3. The TXYZ is a commonly used mapping strategy when there are more than one process per compute node. Since the execution time depends on the input, we choose to show the improvement percentage. The baseline performance for comparison is the system default mapping using XYZT. Higher bars in the figure represent better performance improvement. The proposed mapping algorithm is competitive with TXYZ mapping when using the VN mode but is better in the DUAL mode.

C. NEK5000

NEK5000 [1] is a spectral element mixed C and Fortran MPI code for the simulation of unsteady incompressible fluid flow heat transfer and MHD in general three-dimensional domains. The code is based on the spectral-element method(SEM), a hybrid spectral and finite-element methods. The grid is normally unstructured. The communication is mainly boundary exchange with a wide variety of message sizes. The locality for communication is not perfect and it may

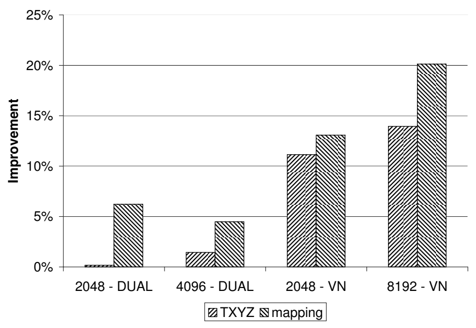

Figure 4. NEK5000 communication performance

take the advantage to use a mapping to get better locality for communication.

Figure 4 shows the results. The proposed mapping strategy further improves the communication performance up to 5% as compared to the TXYZ mapping strategy from the baseline XYZT mapping performance.

V. Discussion and conclusion

As the system is getting larger (e.g., number of processors, cores per processor and threads per core) and the architecture is getting more complicated (e.g., different levels of execution units - processor, core, thread; higher dimensional interconnections), how to utilize the computing resource efficiently becomes an important issue. One of the challenges is to map the tasks onto the target host machine properly to improve the communication performance .

The algorithm framework we proposed in this paper is trying to address this issue. We show the algorithm finds the optimal solution that minimize the maximum size of messages exchanged between the processors with 2-, 4- or 6-core cases. The experiments show good communication performance improvement for the benchmarks and applications.

The algorithm may not be able to find the optimal solution with $O(m \log m)$ time complexity for the cases where core number $k > 6$. However, with increased time complexity, it is possible to modify the Bin-packing function used by Algorithm 3 to generalize it so it can handle cases with $k > 6$. In addition, the parallelization of our framework for even better scalability is also of great interests. We plan to further improve the algorithm towards those directions.

References

[1] http://nek5000.mcs.anl.gov/.

[2] The ASCI sweep3d Benchmark Code.

[3] Masood Ahmed and Shahid Bokhari. Mapping with space filling surfaces. *IEEE Trans. Parallel Distrib. Syst.*, 18:1258–1269, September 2007.

[4] Romas Aleliunas and Arnold L. Rosenberg. On embedding rectangular grids in square grids. *IEEE Transactions on Computers*, 31(9):907–913, September 1982.

[5] G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C . Sexton, and R. Walkup. Optimizing task layout on the blue gene/l supercomputer. *IBM Journal of Research and Development*, 49(2):489–500, March 2005.

[6] Abhinav Bhatelé, Eric Bohm, and Laxmikant V. Kalé. A case study of communication optimizations on 3d mesh interconnects. In *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, pages 1015–1028, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] I-Hsin Chung, Che-Rung Lee, Jiazheng Zhou, and Yeh-Ching Chung. Hierarchical mapping for HPC applications. In *Workshop on Large-Scale Parallel Processing*, 2011.

[8] John A. Ellis. Embedding rectangular grids into square grids. *IEEE Trans. Comput.*, 40(1):46–52, 1991.

[9] Rami G. Melhem and Ghil-Young Hwang. Embedding rectangular grids into square grids with dilation two. *IEEE Trans. Comput.*, 39(12):1446–1455, 1990.

[10] Sangman Moh, Chansu Yu, Dongsoo Han, Hee Yong Youn, and Ben Lee. Mapping strategies for switch-based cluster systems of irregular topology. In *8th IEEE International Conference on Parallel and Distributed Systems*, Kyongju City, Korea, June 2001.

[11] Juan Manuel Orduna, Federico Silla, and Jose Duato. On the development of a communication-aware task mapping technique. *Journal of Systems Architecture*, 50(1):207–220, March 2004.

[12] Hao Yu, I-Hsin Chung, and Jose Moreira. Topology mapping for Blue Gene/L supercomputer. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116, New York, NY, USA, 2006. ACM.