Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Topology Aware Task-To-Processor Assignment

Reha Oğuz Selvitopi[a], Ata Türk[a], Altay Güvenir[a], Cevdet Aykanat[a]

[a]*Bilkent University, Computer Engineering Department, 06800 Ankara, Turkey*

Abstract

Topology aware mapping has started to attain interest again by the development of supercomputers whose topologies consist of thousands of processors with large diameters. In such parallel architectures, it is possible to obtain performance improvements for the executed parallel programs via careful mapping of tasks to processors by considering properties of the underlying topology and the communication pattern of the mapped program. One of the most widely used metric for capturing a parallel program's communication overhead is the hop-bytes metric which takes the processor topology into account which is in contrast to the assumptions made by the wormhole routing. In this work, we propose a KL-based iterative improvement heuristic for mapping tasks of a given program to the processors of the parallel architecture where the objective is the reduction of the communication volume that is modeled with the hop-bytes metric. We assume that the communication pattern of the program is known beforehand and the processor topology information is available. The algorithm basically tries to improve a given initial mapping with a number of successive task swaps defined within a given processor neighborhood. We test our algorithm for different number of tasks and processors and demonstrate its results by comparing it to random mapping, which is widely used in recent supercomputers.

1. Introduction

With the advent of the supercomputers built with thousands of cores in the last decade, the topology aware mapping has began to attain a certain research interest from parallel, scientific, and high performance computing communities. In such architectures, where many users submit parallel applications that heavily rely on scalability, the performance implications are of primary concern. Some of these parallel applications follow a predefined communication pattern. For such an application, it is possible obtain performance improvements by taking advantage of the communication pattern of the application and mapping tasks of that application to processors in such a way that the interaction between tasks is reduced. The mapping schemes must surely take the underlying topology of the parallel system into account in the mapping process so that the interacting tasks are assigned to processors that are close to each other. Hence, careful mappings that use the processor topology and the interaction information between tasks can benefit the performance of the target parallel applications greatly.

The topology aware mapping has become important again for two basic reasons:

- The wormhole routing assumes that the message latencies are independent of the distance between processors in the absence of blocking where no contention occurs in the network. This assumption is no more valid for the recent supercomputers where the diameter of the supercomputer can be quite high. Especially for small-to-medium sized messages, message latencies cannot be considered independent of the number of links (hops) between processors.
- More importantly, in the presence of blocking which is the realistic setting in most of the used interconnection networks for recent supercomputers, the situation becomes more complex. The bandwidth is now shared by multiple processors and

generally, it is not possible to utilize the available bandwidth to its full extent. Since networks resources are shared, contention occurs, the bandwidth available per link decreases, and consequently, message latencies increase.

Hence, it can be said that in the *absence* of contention, message latencies are not independent of the number of hops between processors. In the *presence* of contention, the importance of hop count between interacting tasks becomes more important, especially for large messages [1].

The current literature on topology aware mapping is generally centered on specific type of applications that have certain properties in their communication patterns and they are often designed for specific type of parallel architectures. The very early approaches for the mapping problem are not directly applicable to today's supercomputers since they focus on different type of parallel architectures. The recent approaches, on the contrary, although designed for the recent parallel architectures, lack breadth and are far from elegancy.

Our contribution in this work is to propose and implement an algorithm for mapping a given set of tasks to a given set of processors so that the communication volume is reduced. Our algorithm is based on a well-known iterative improvement heuristic, KL [2], that tries to improve the mapping by task swaps. We define proper neighborhood definitions that suit well for the addressed parallel architectures, mainly meshes and torus. Starting from a random task to processor assignment, the algorithm improves the quality of the initial mapping by a number of successive task swaps. We test our algorithm on realistic task interaction and processor organization graphs and demonstrate results for different number of tasks and processors.

2. Problem Definition

For the applications that are designed to execute on parallel systems, the interaction between tasks can be static and such interactions can easily be modeled by a graph. Vertices of this graph correspond to tasks and the edges correspond to interactions between tasks. If there is apriori information about the computational complexities of the tasks, weights can be assigned to the vertices to denote the computational loads of associated tasks. There are two basic schemes for modeling static interaction between tasks: Task Precedence Graph (TPG) and Task Interaction Graph (TIG). In TIG model, tasks can be executed independent of each other and there are no dependencies among tasks. For this reason, the interactions between tasks i and j are captured by the *undirected* edge e_{ij}. A TIG is denoted by $G_T = (V, E)$ and has $|V| = N$ vertices. Vertices of G_T represent the tasks of the modeled parallel program and the weightw_i denotes the computational cost associated with task$i$. The edge e_{ij} denotes the interaction between tasks i and j and its cost corresponds to the amount of communication between these tasks.

The parallel architecture or the processor topology can also be modeled by a graph. The Processor Organization Graph (POG) is a graphical representation of the interconnection topology where the nodes represent the processors and the edges represent the communication links between them. Communication between non-adjacent pairs of processors can be associated with relative unit communication costs. That is generally happened to be the shortest path between any two processors (or it can be configured using the routing scheme the software/hardware uses). Therefore, we can obtain a complete undirected graph, called the Processor Communication Graph (PCG), whose nodes represent the processors and edge weights represent the unit communication costs between pairs of processors. A PCG is denoted by $G_P = (P, D)$ is a complete graph with $|P| = K$ nodes. Vertices of G_P denote the processors of the parallel system. There is an edge between every pair of vertices and the cost of the edge d_{pq} denotes the unit communication cost between processors p and q.

Thus, the mapping problem is to map a given TIG G_T to a given PCG G_P so that the inter-processor communication is minimized while computational load is maintained. Formally, we are to find a mapping function $f_M: V \rightarrow P$ that maps tasks to processors so that the inter-processor communication

$$\sum_{e_{ij} \in \varepsilon, f_M(i) \neq f_M(j)} e_{ij} \times d_{f_M(i) f_M(j)}$$

is minimized while the computational load

$$\sum_{v_i \in V, f_M(i) = p} w_i \text{, for } 1 \leq p \leq K$$

of each processor is maintained. Here, $f_M(i)$ denotes the processor that the task i is mapped to. The inter-processor communication between any two tasks i and j depends on the communication volume (generally denoted by number of words and modeled by the edge e_{ij}) occurring between these two tasks and the number of hops between the processors which these

tasks are assigned to (denoted by $f_M(i)$ and $f_M(j)$, and the number of hops being $d_{f_M(i)f_M(j)}$). The computational load of a task roughly corresponds to the amount of computation performed by that task throughout the execution and it is denotes by w_i.

3. KL-based Iterative Improvement Heuristic for Topology Aware Mapping

The KL algorithm is originally proposed for partitioning graphs. It basically performs a number of vertex pair swaps between two partitions, which consist of a set of vertices. At each iteration, the swap operation that has the largest gain is selected, performed and locked for the rest of the pass to prevent these vertices to be selected for other swap operations throughout the same pass. The gain of a swap operation is basically defined as the improvement in the objective function if the swap of vertices involved in this swap operation were actually performed. After swapping a pair of vertices, the gains of other unlocked swap operations might need to be updated. Generally, a pass continues till there is no improvement in the objective for a predetermined number of swap operations or there remains no more swap operations to be performed. Multiple passes are generally performed and at the end of each pass, a rollback operation is performed to the point where the best cost is obtained.

Our proposed algorithm is based on this basic KL heuristic. It assumes that the number of tasks is equal to the number of processors. The basic properties of our heuristic are as follows:

- *Swap Operation:* Swap operation is defined on a pair of tasks. Assume that tasks i and j are assigned to processors p and q, respectively. Then, the swap operation on these tasks swaps these tasks by assigning task i to processor q from processor p and assigning task j to processor p from processor q. We limit our search space by selecting the pair of vertices to be swapped in a clever manner, which is described next.

- *Neighborhood Definition:* We limit the number of swap operations ($(N^2 - N)/2$ at most) by defining a selection metric based on the distances between processors. We call this parameter *threshold t*. Assume that tasks i and j are assigned to processors p and q, respectively. Then the swap of these tasks is considered if distance between processors p and q are smaller than or equal to t, i.e., $d_{pq} \leq t$. This limits the search space greatly, which makes the expensive KL-based heuristic cheaper without giving much off the quality of the obtained mappings.

- *Gain Definition:* As mentioned, our objective is the communication volume obtained via hop-bytes metric. The gain of a swap operation is defined as the reduction in the total communication volume if the tasks involved in this operation were to be swapped. The gain of a swap operation can be negative, which we allow to happen.

- *Gain Update:* The complexity of a KL-based algorithm is mainly determined by its gain update complexity. After swapping a pair of tasks i and j, the gain values of the swap operations, which involve the unlocked neighbors of tasks i and j need to updated. A single such gain update can be performed in constant time. Thus after swapping a pair of tasks, the total gain update can take $O(N)$ at worst-case where the swapped tasks are connected to all other vertices in the task interaction graph.

The proposed algorithm starts from a random initial mapping and improves it by a number of successive task swaps whose basics are outlined above. We perform all swap operations in a single pass till there remain no more swaps and rollback to the point where the best cost is seen. Traditionally, in graph partitioning, a small number KL passes are enough. However, that is not the case for our algorithm, which is basically because we need to take the processor distances into account while swapping tasks. Thus, we let the algorithm to perform as many passes as it needs till improvement drops below a certain threshold.

4. Results Obtained

We tested the proposed KL-based heuristic for different number of tasks and number of processors. The modeled parallel application is the sparse matrix vector multiplication. This application is modeled with hypergraph as mentioned in [3]. There are two basic models in this representation, row-net (RN) and column-net (CN). The tested number of tasks/processors varies from 64 to 2048 for the values with power of two. Table 1 gives information about the maximum task degrees for the tested matrices for both row-net and column-net models.

	Matrices													
	2cubes_sphere		**fullb**		**G3_circuit**		**Language**		**stokes128**		**ted_A**		**tmt_sym**	
ntasks nprocs	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**
64	12.34	12.281	7.34	7.22	5.06	5.25	63.00	62.78	4.97	5.09	4.63	4.38	5.31	5.16
128	13.45	13.28	7.92	7.66	5.25	5.19	123.42	122.19	5.55	5.45	6.20	5.95	5.48	5.41
256	13.84	13.84	8.30	8.14	5.66	5.62	201.68	197.70	5.67	5.71	8.59	9.72	5.65	5.67
512	14.17	13.98	9.25	9.29	5.94	5.95	224.73	221.64	6.02	6.03	12.08	13.45	5.77	5.72
1024	14.42	14.63	9.70	9.60	6.62	6.60	176.57	184.23	6.24	6.29	15.94	18.93	5.84	5.88
2048	14.53	14.59	9.65	9.64	6.62	7.66	115.32	124.29	6.61	6.62	20.99	28.62	5.91	5.88

Table 1: Average task degrees for the task interaction graphs (RN = row-net, CN = column-net)

Table 2 represents the diameter values of the tested topologies.

Number of processors	Maximum hops
64	9
128	13
256	17
512	12
1024	16
2048	24

Table 2: Maximum hops for each topology

As noted, the threshold value (t) can be set to different values.Table 3,4 and 5 demonstrate the percentage improvement values of the KL-based heuristic with respect to initial random partitioning for $t = 1,2$ and 4, respectively.

	Matrices													
	2cubes_sphere		**fullb**		**G3_circuit**		**language**		**stokes128**		**ted_A**		**tmt_sym**	
ntasks nprocs	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**	**RN**	**CN**
64	37.5	38.7	48.4	48.7	45.4	41.4	14.0	16.2	55.9	46.3	51.3	53.2	47.8	40.2
128	48.1	43.2	52.4	50.0	48.3	51.7	17.6	18.9	57.2	53.8	55.9	58.6	51.8	57.4
256	51.6	52.9	51.3	54.2	53.2	56.9	19.3	23.0	57.2	55.1	57.4	55.8	56.0	56.9
512	51.8	51.8	54.0	54.4	59.5	58.2	18.7	20.2	56.0	56.7	55.7	55.8	57.7	56.7
1024	55.3	53.5	59.8	59.5	62.1	62.0	20.2	23.8	61.5	61.6	58.6	58.3	60.2	60.3
2048	59.1	57.2	59.8	58.7	61.5	62.5	21.9	25.6	63.0	63.1	58.1	57.2	62.5	62.9

Table 3: Percentage improvement (%) for tested matrices for $t = 1$

	Matrices													
	2cubes_sphere		fullb		G3_circuit		language		stokes128		ted_A		tmt_sym	
ntasks nprocs	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN
64	48.9	44.8	45.0	46.3	51.7	56.9	17.6	20.5	54.3	50.6	61.3	59.1	56.7	56.7
128	52.0	49.5	57.1	61.8	61.5	66.5	16.5	23.6	61.0	62.3	65.8	65.2	63.6	59.3
256	59.4	58.6	61.0	62.6	67.1	66.7	23.9	26.0	65.3	60.6	66.6	66.2	65.9	63.0
512	54.6	57.1	59.9	61.3	65.2	64.3	20.8	23.3	65.0	64.8	63.2	62.0	64.8	62.8
1024	62.5	60.9	65.4	65.1	67.6	69.2	22.6	25.6	69.0	67.5	65.3	65.1	67.5	67.6
2048	66.2	64.9	68.4	69.2	69.6	71.2	28.1	28.3	70.6	71.9	68.8	67.2	71.3	71.7

Table 4: Percentage improvement (%) for tested matrices for $t = 2$

	Matrices													
	2cubes_sphere		fullb		G3_circuit		language		stokes128		ted_A		tmt_sym	
ntasks nprocs	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN	RN	CN
64	46.4	46.4	54.8	50.4	56.1	55.4	17.2	21.5	55.7	58.0	60.4	61.5	54.3	57.6
128	52.5	55.3	62.3	60.1	65.2	66.7	20.8	24.0	63.2	63.6	65.3	67.7	64.2	64.0
256	58.9	58.2	63.2	62.0	64.9	67.3	25.1	30.0	67.8	67.3	68.5	68.5	65.9	67.0
512	57.5	58.8	62.1	61.9	65.9	66.0	21.0	24.1	66.5	64.4	64.2	64.1	65.7	64.9
1024	64.2	63.4	66.8	66.2	70.9	70.9	25.0	27.4	69.6	70.0	67.2	67.5	69.7	69.7
2048	69.0	67.7	70.4	71.1	73.4	73.8	30.2	31.7	74.2	73.9	72.5	69.2	74.6	75.0

Table 5: Percentage improvement (%) for tested matrices for $t = 4$

It is seen from these tables that as the threshold increases, the obtained improvement also increases. This is because we extend the search space by increasing the threshold. On the other hand, the complexity of the algorithm increases as we increase the threshold since we need to consider more swap operations and in turn more gain updated.

5. Conclusion

We have developed an iterative improvement based algorithm for topology aware mapping. The algorithm proceeds by successive task swaps with the objective of reducing communication volume defined by the hop bytes metric, which has proved its importance in recent supercomputers. The algorithm is tested for different matrices with different number of tasks and processors. We tested the algorithm for different threshold values by comparing it to the random mapping. The results show how important performance gains can be obtained by careful mapping.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-283493 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources.

References

[1] Abhinav Bhatele. *Automating topology aware mapping for supercomputers*. PhD thesis, Champaign, IL, USA, 2010. AAI3425400.

[2] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.*, 49:291–307, 1970.

[3] U. Catalyurek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Syst. 10 (1999) 673–693.