Mapping Applications on Irregular Allocations

Seonmyeong Bak*, Nikhil Jain† (Mentor), Laxmikant V. Kale* (Advisor)
*Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
†Lawrence Livermore National Laboratory, Livermore, CA 94551, USA
*{sbak5, kale}@illinois.edu, †nikhil@llnl.gov

I. Introduction

Topology-aware mapping of applications on clusters and supercomputers becomes more difficult as the number of nodes in the systems increase and interconnection networks become more complex. To facilitate this process, Rubik [1] was proposed to generate mappings that lead to good performance on allocations that are symmetric, compact, and convex, e.g. allocations on most Blue Gene/Q systems. However, many supercomputers provide irregular, asymmetric, and disjoint allocations to users for better utilization of resources. It is significantly more difficult to map applications on these allocations because of the lack of structure in the allocations and unavailability of certain nodes which are reserved for other work, e.g. IO servers. In this poster, we extend Rubik to provide mapping support for these complex cases by using different heuristics to project virtual machine topologies onto real machine topologies. We evaluate our work using two widely used HPC applications, namely MILC and Qbox, on Blue Waters, a Cray XE supercomputer. We show that, for these communication intensive applications, MPI time is reduced by 60% in MILC and 56% in Qbox when running on 16,384 ranks using the proposed extension to Rubik.

II. Related Work and Motivation

Rubik [1] is a python based framework for mapping applications with structured communication patterns onto regular allocation grids. Figure 1 shows the basic idea of Rubik.

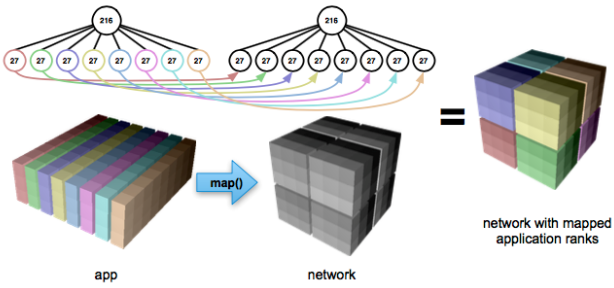

Fig. 1: Rubik: high level overview [1].

In a Rubik script, application and network grids are described as cuboids. The map() function in the script maps each MPI rank in the application grid to the network grid. Using the transformations provided by Rubik, users can group together different sets of points in the two grids, and map these sets to one another based on their choice. With Rubik, users, who have a good understanding of an application's algorithms and communication patterns, can minimize the cost of communication by writing a simple python script. Following is a Rubik script to partition and map MPI ranks in a 8×8 application grid onto a $4 \times 4 \times 4$ 3D torus allocation.

```
# Create app partition tree of 64-task planes
    app = box([8,8])
    app.tile([8,1])
# Create network partition tree of 64-processor cubes
    network = box([4, 4, 4])
    network.tile([2,2,2])
    network.map(app) \# Map Task planes into cubes
```

As we can see, Rubik makes partitioning and mapping of MPI ranks easy by using a few lines of code in a python script. Rubik supports many kinds of partition methods such as tile, divide, mod, and cut (described in [1]). It also provides permuting operations like tilt, zigzag and zorder.

Rubik's limitation on irregular allocations: Currently, Rubik only supports partitioning and permuting of MPI ranks on regular allocations. It neither handles the case in which the allocated nodes may not form full cuboids, nor does it support exclusion of certain unavailable nodes such as the service nodes or IO nodes. This limits usage of Rubik to Blue Gene systems only.

III. Extension and Mapping Schemes

In this work, we extend Rubik to support allocations that are either irregular in shape and/or consist of unavailable nodes. To do so, we introduce the concept of virtual network grid. Instead of mapping the application grid to the real network grid, users can now define a virtual network grid. Application grid is mapped to this virtual grid, which in turn is mapped to the real allocation grid using heuristics defined inside Rubik. Currently, two such heuristics are supported.

Row ordering maps the MPI ranks in the virtual grid to nodes in the real allocation using a dimension ordered traversal ordered by bandwidth of links in each dimension.

Recursive Coordinates Splitting groups neighboring nodes in an allocation into subcuboids. This technique splits the given grids into smaller subcuboids and distributes holes in an allocation into those small cuboids to prevent the cost of the holes from being concentrated in certain regions of the allocation. Figure 2 shows how the recursive splitting works and the advantages of recursive coordinates splitting over row-ordering in asymmetrical allocations having holes inside. As shown in this figure, applications with near-neighbor communication pattern such as stencil benefit from recursive splitting because it groups neighboring MPI ranks into groups

so neighboring MPI ranks can be placed in tiles even on the asymmetrical allocation having service nodes. For example, Rank 6 communicates with Rank 2, 5, 7 and 8. The number of total hops between Rank 6 and others is reduced by half when it is placed with recursive splitting compared to row-ordering.

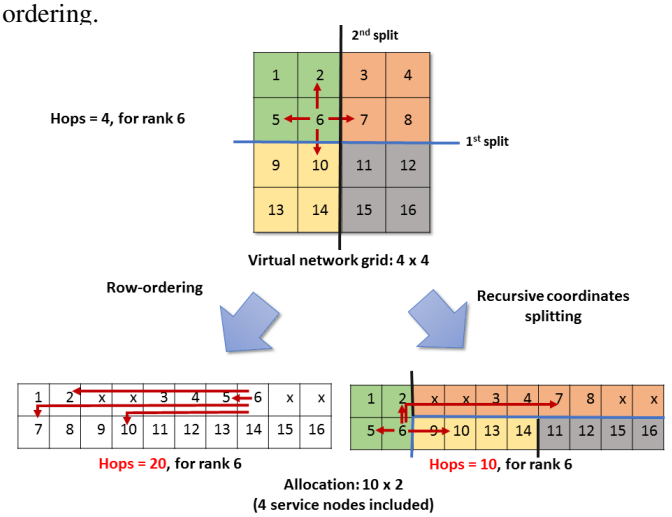

Fig. 2: The advantage of Recursive Coordinates Splitting on an asymmetrical allocation having service nodes.

With recursive coordinates splitting, how an allocation is split is important. More specifically, the benefit of this scheme can change depending on the order of directions of each splitting and the shape of the logical torus. We placed MPI ranks to maximize bisection bandwidth in each splitting and used simple heuristic to give users approximately similar shape of logical torus to the real allocation. The heuristic factorizes the number of PEs and starts from 1 x 1 x 1 for a logical torus. Repeatedly it multiplies each dimension with factors by the factorization. It enables more similar shape of virtual grids than using a simple fixed shape of virtual grids

IV. Experimental Results

# of tasks	Geometry	Internal fragmentation	Service Nodes
4096	9 x 2 x 8	24/280 (8.57 %)	8 (2.78%)
16384	11 x 2 x 24	0/1024 (0.00%)	32 (3.03%)

TABLE I: Details of the allocations used in this experiment.

To evaluate the benefit of this work, we have run MILC and Qbox on NCSA Blue Waters with the parameters and allocations shown in Table I. Internal fragmentation is the number of gemini routers (nodes) in each allocation which are not available for the experiment. Service nodes is the number of service nodes in each allocation; percentage values shown are the ratio of service nodes to the number of nodes in each allocation excluding the internal fragmentation. We ran both the applications with 4K and 16K cores and used CrayPat to get timing results for user functions and MPI routines.

Each stacked bar graph in Figures 3 & 4 shows time spent in MPI communication routines. In each graph, 'default' means the result with the default mapping, 'rcb' means the result with recursive coordinates splitting, 'row' means the result with row-ordering, and 'With opt' are the results with heuristics for better shape of virtual grid in a Rubik script. All the data points are normalized by the result of run with the default mapping.

In both applications, we achieved significant reduction in MPI communication time. For MILC, we achieved reduction in MPI communication routines by 27% on 4K ranks and 59% on 16K ranks. For Qbox, the benefit of this work was not significant in experiments on 4K ranks but we achieved significant reduction in timing for SCALAPACK functions on 16K ranks. Note that the functions of the SCALAPACK used by Qbox uses MPI routines such as MPI_Send/Recv, MPI_Bcast and MPI_Alltoall.

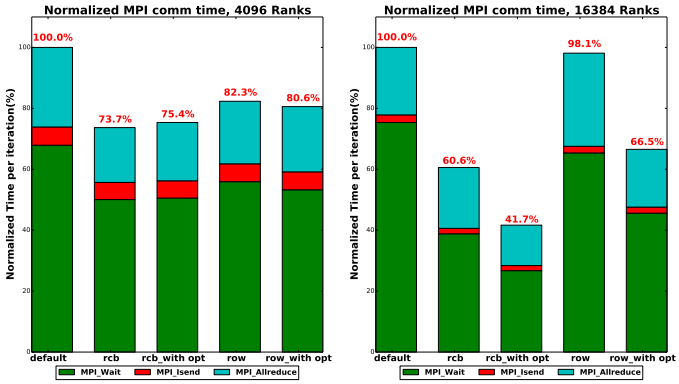

Fig. 3: MILC with 4096 and 16384 ranks on 9x2x8 and 11x2x8 allocations

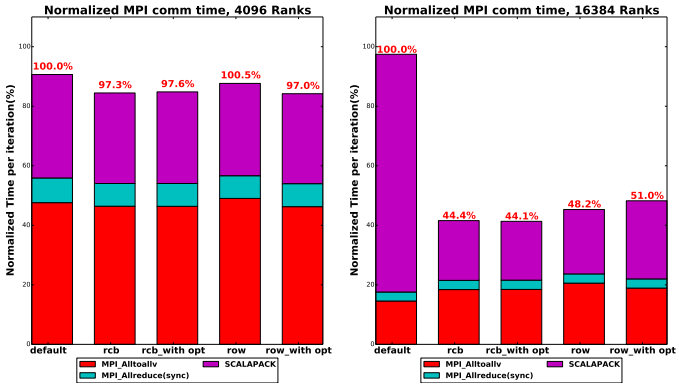

Fig. 4: Qbox with 4096 and 16384 ranks on 9x2x8 and 11x2x8 allocations

V. Conclusion

We extended Rubik to help users map their applications on irregular allocations with minimal changes to their existing scripts. We implemented and evaluated two mapping schemes, namely the row-order and recursive splitting, and applied a few heuristics to maximize the bisection bandwidth and minimize hops. Our experiments on NCSA Blue Waters show significant improvement in MPI communication time for two widely used HPC applications, MILC and Qbox.

References

[1] A. Bhatele, T. Gamblin, S. H. Langer, P.-T. Bremer, E. W. Draeger, B. Hamann, K. E. Isaacs, A. G. Landge, J. A. Levine, V. Pascucci, M. Schulz, and C. H. Still, "Mapping applications with collectives over sub-communicators on torus networks," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. IEEE Computer Society, Nov. 2012 (to appear), lLNL-CONF-556491.