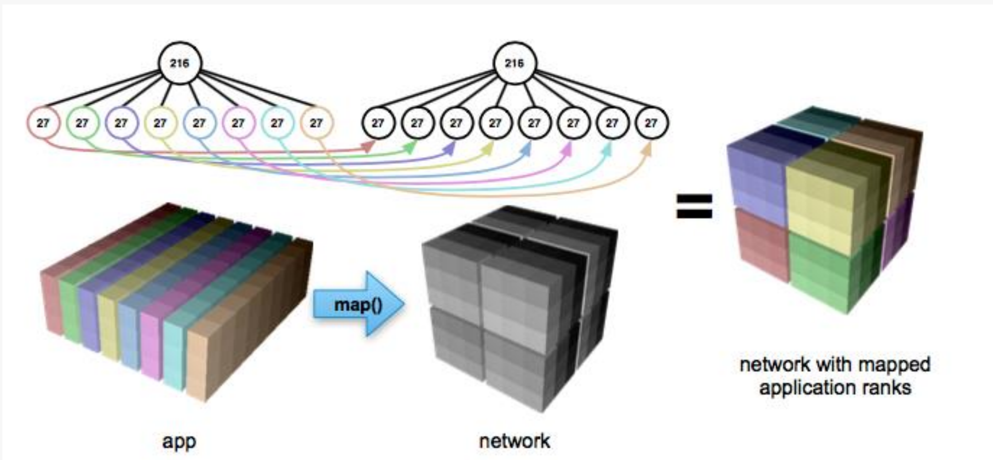
Mapping Applications on Irregular Allocations

Seonmyeong Bak, Nikhil Jain(Mentor), Laxmikant V. Kale(Advisor)

Abstract

- Mapping applications on clusters becomes more difficult as the number of nodes become larger
- Supercomputers assigns allocations with irregular shapes to users to maximize the utilization of resources
- Much more difficult to map applications on these irregular allocations
- We extended Rubik, a python based framework to map applications on irregular allocations with a few lines of python codes
  - Rubik was originally designed for regular allocations, so we added features to handle allocations with irregular structure and unavailable nodes and two mapping algorithms such as row-ordering and recursive splitting
- We evaluate our work with two widely used HPC applications on Blue Waters: MILC and Qbox
  - We reduce execution time by 32.5% in MILC and by 36.3% in Qbox, and communication time by 60% in MILC and 56% in Qbox

Rubik, a python framework for structured communication

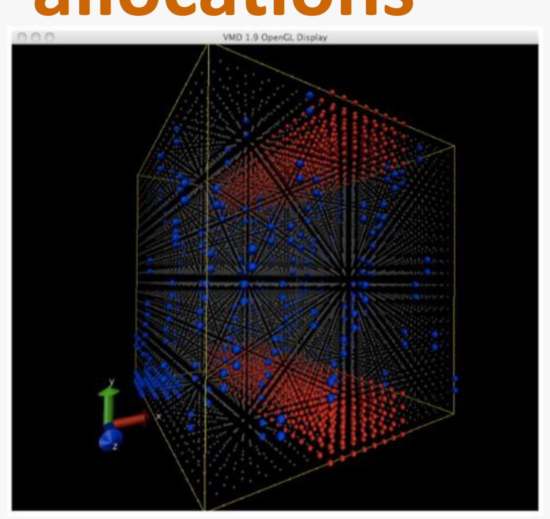



```
# Create app partition tree of 64-task planes
    app = box([8,8])
    app.tile([8,1])
# Create network partition tree of 64-processor cubes
    network = box([4, 4, 4])
    network.tile([2,2,2])
    network.map(app) \# Map Task planes into cubes
```

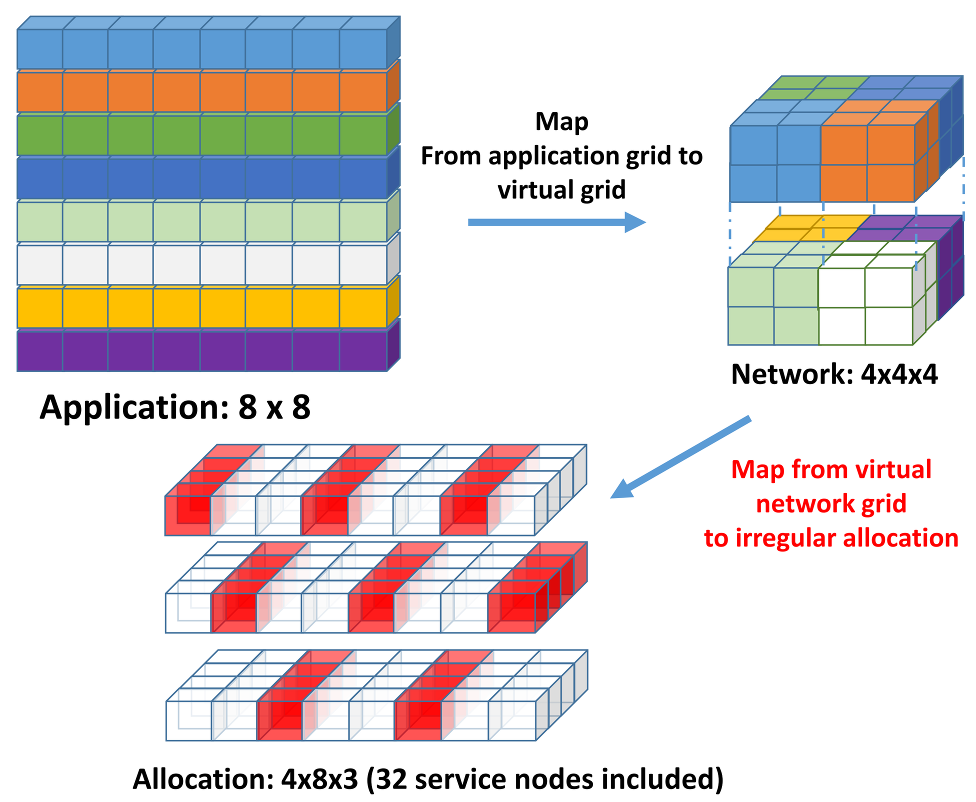
- Rubik is a python based framework developed at LLNL for mapping applications with structured communication onto regular allocation[1]
- Rubik facilitates mapping of user application using a few lines of python code
  - Users can easily general several different mappings for their applications
- Rubik supports many types of operations for better mapping of application grids onto network grids

Limitation of Rubik for irregular allocations

- Rubik is designed for mapping applications onto regular and symmetrical allocations
  - However, in many cases, the shapes of the allocations are irregular as the Blue Waters (Gray -> compute nodes, red -> XK nodes, blue-> service nodes)
  - Motivation of this work: how to enable use of Rubik on irregular allocations for its broader applicability

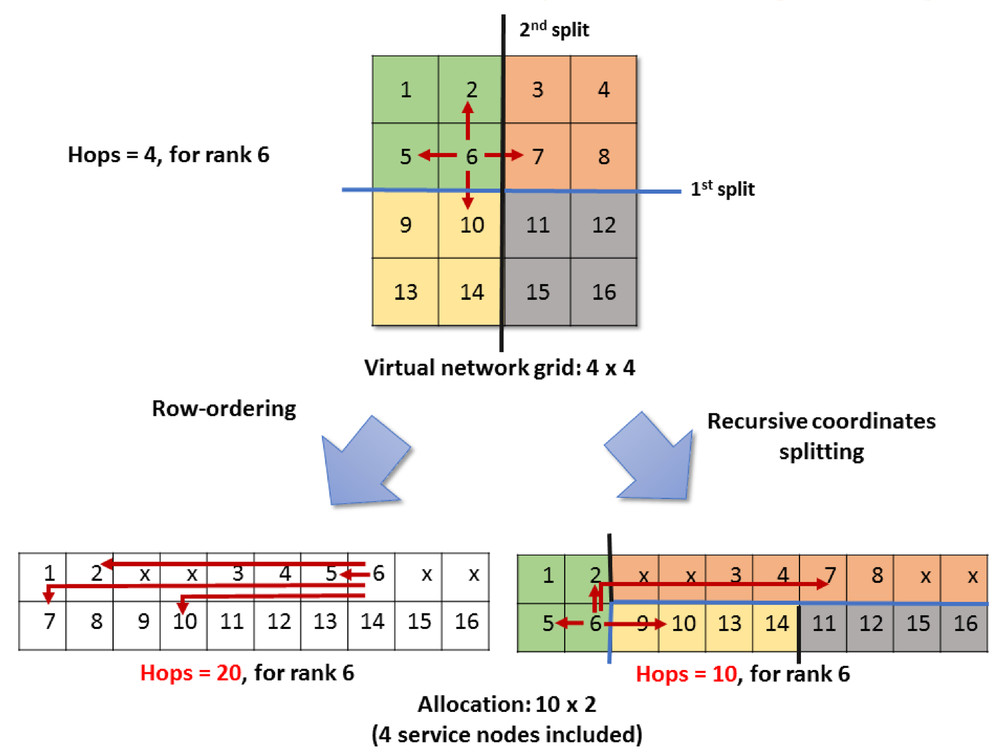


Basic mechanism



- Users write a Rubik script to map their application grids onto virtual network grids
- Our projecting algorithms maps virtual grids onto real irregular allocations with possibly unavailable nodes
- All our changes in Rubik are hidden from users – old rubik scripts will continue to work with very few changes

Projecting Algorithm



- Row ordering
  - Place MPI ranks in the order of each axis in an allocation
  - Suitable for allocations that are mostly regular
- Recursive splitting
  - Split virtual network grid and an allocation into subcuboids
  - Map each subcuboid in the virtual network grid into corresponding subcuboid in the allocation
  - Suitable for irregular allocations and can minimize hops for apps having neighbor communication pattern

Optimizations

- The direction of splitting
  - Calculate bisection bandwidth in each splitting and choose the direction where the bisection BW is lowest
    - To maximize the bisection bandwidth between closest subcuboids

Input: the shape of the allocation, number of tasks
Output: $\mathcal{V}$: Direction vector for the Recursive Splitting

```
partition ← the shape of the allocation
factors ← factorise (the number of tasks)
factorIdx ← 0
B ← calculate bandwidth between subcuboids by factors[factorIdx] in each axis
while product(partition) > 1 do
    currentAxis ← choose axis where the bandwidth in B is lowest
    partition[currentAxis] ← partition[currentAxis] / factors[factorIdx]
    B[currentAxis] ← B[currentAxis] / factors[factorIdx]
    Add currentAxis to 𝒱
    factorIdx ← factorIdx + 1
end while
```

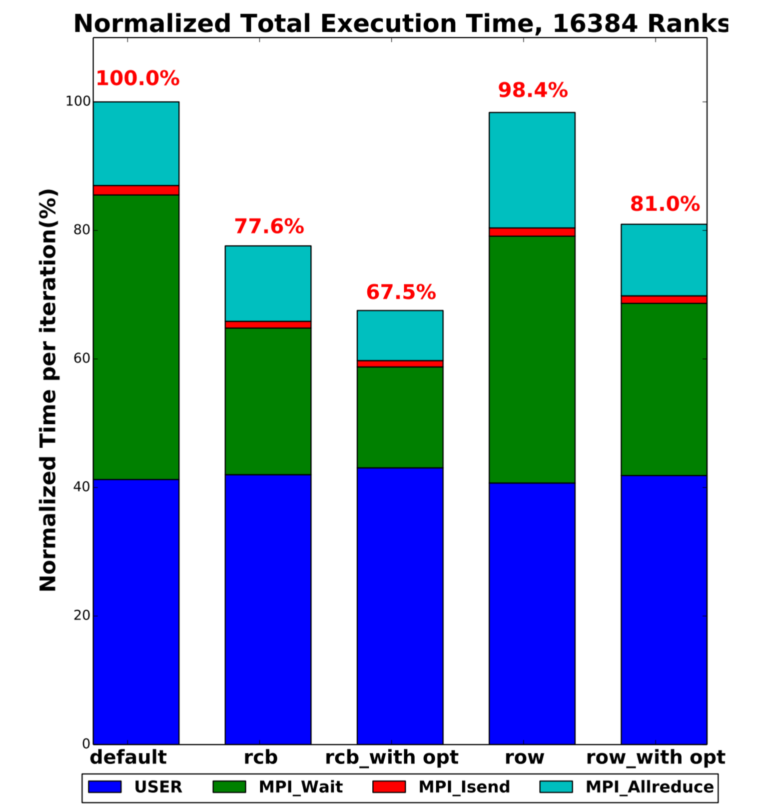
- The shape of the virtual network grid
  - Estimate the shape of the virtual network grid with the shape of the allocation
  - Factorize the number of tasks and use factors by this factorization for the estimation
    - E.g.) 9 x 2 x 8 allocation, 4096 ranks
      - Factorize 4096 -> 2^{12}
      - Start from 1 x 1 x 1 -> multiply each dimension with factors until each dimension becomes equal or larger than the corresponding dimension of the allocation
      - 1 x 1 x 1 -> 2 x 1 x 1 -> 2 x 2 x 2 -> 4 x 2 x 2 -> 4 x 2 x 4(because y dimension is already 2) -> 8 x 2 x 4 -> 8 x 2 x 8 (done)

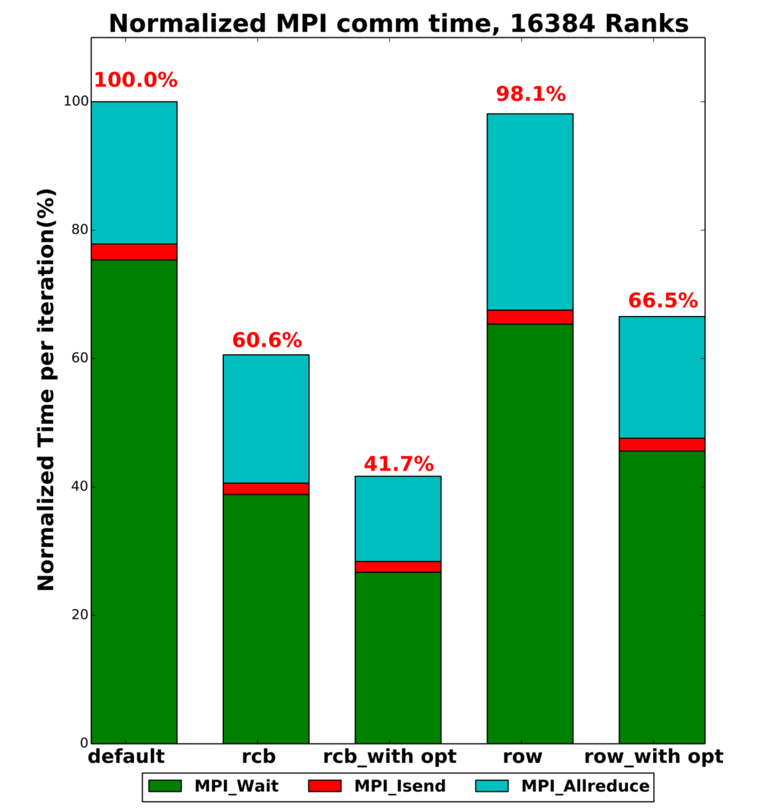
Experimental Results

- Configuration for experiments
  - The details of the allocations

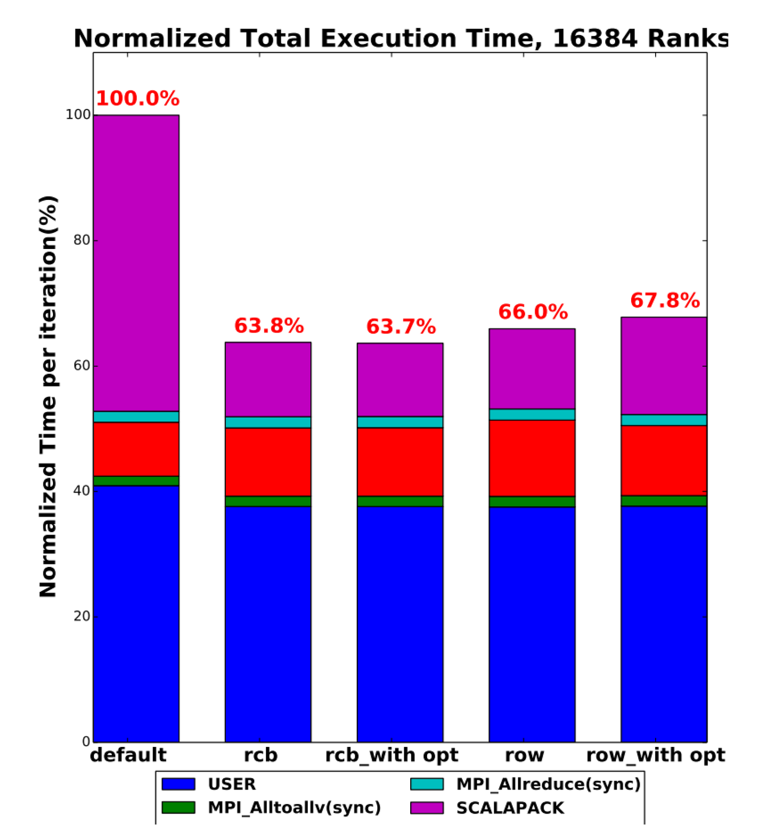
# of tasks	Geometry	Internal fragmentation	Service Nodes
16384	11 x 2 x 24	0/1024(0.00%)	32(3.03%)

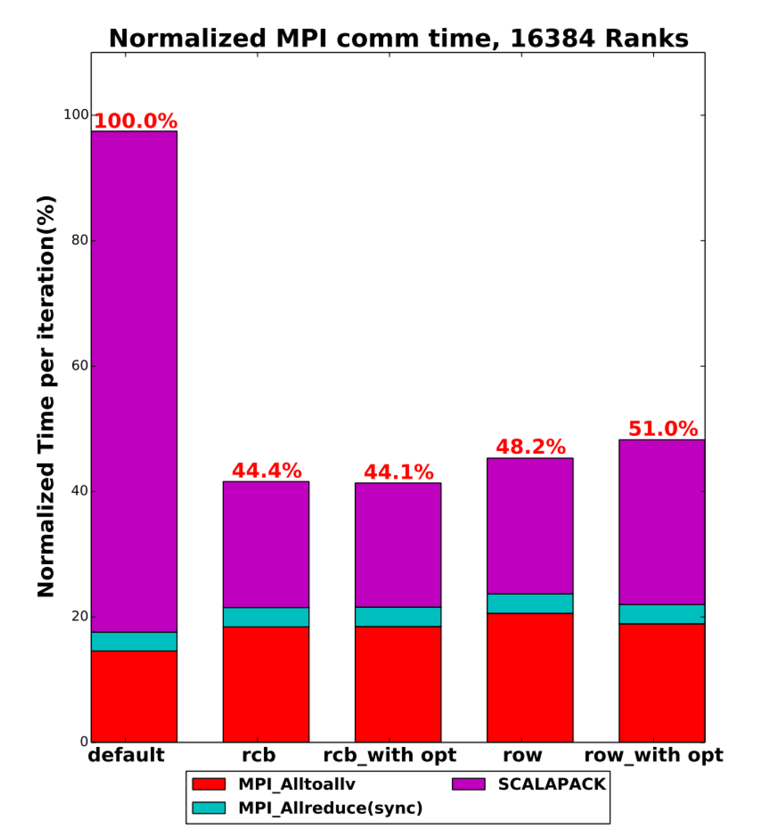
  - Machine: NCSA Blue Waters , hybrid machine of Cray XE/XK nodes
    - Blue Waters has 2 nodes in one gemini and we used 16 cores per node
  - Applications: MILC, Qbox
  - All the results are normalized to the corresponding result with the default mapping of the Blue Waters
  - Rcb with opt, row with opt means the results with optimization for better shape of the virtual network grid
    - RCB and row without optimizations used regular shape of the virtual network grid(e.g. 8 x 4 x 4 for 4096 ranks, 8 x 8 x 8 for 16384 ranks.)
  - On each stacked bar, USER means execution time for functions in each application excluding elapsed time for communication related functions
- MILC
  - 32.5% improved in execution
  - 59.3% improved in communication
  - With many number of cores, MILC spent more time on communication
  - This work minimize hops between ranks so p2p operations are more improved than collective operations Optimization to estimate the shape of the virtual network grid seems effective in more irregular allocations with more number of cores
    - The random shape of the virtual network grid can increase hops between ranks by inefficient placement of ranks





- Qbox
  - 36.3% improved in execution
  - 59.3% improved in communication
  - Qbox doesn't call p2p routines directly. Instead, it uses SCALAPACK for p2p communication between ranks
  - SCALAPACK in each graph include elapsed time for functions in SCLAPACK and most of them use MPI routines significantly
  - In 16384 ranks, most of reduction comes from the reduction in SCALAPACK





[1] A. Bhatele et al., "Mapping applications with collectives over sub-communicators on torus networks," High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for, Salt Lake City, UT, 2012, pp. 1-11. doi: 10.1109/SC.2012.75