Preliminary results on multiple hybrid nodes of Knights Corner and Sandy Bridge processors

Tan Nguyen and Scott Baden
Dept. of Computer Science & Engineering
University of California, San Diego

Presenter: Tan Nguyen

Introduction

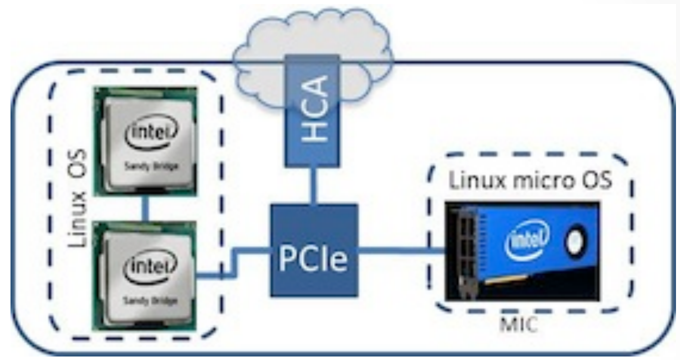
- Systems consisting of hybrid nodes of CPUs and accelerators
  - **Communication overheads** are growing due to node acceleration and the decrease in memory per core
  - **Load balancing** problem arises when heterogeneous resources run at different speeds
- MPI applications have to be optimized/rewritten to
  - Hide communication overheads
  - Handle load balancing
- We present our solutions to overcoming these challenges on Stampede, a system of Sandy Bridge-Knights Corner nodes
  - Semi automatically optimize synchronous MPI code previously written for homogeneous platforms
  - **Bamboo**, **a directive-based compiler,** translates MPI code into a task graph representation that runs under a **dataflow-like execution model**

Overview

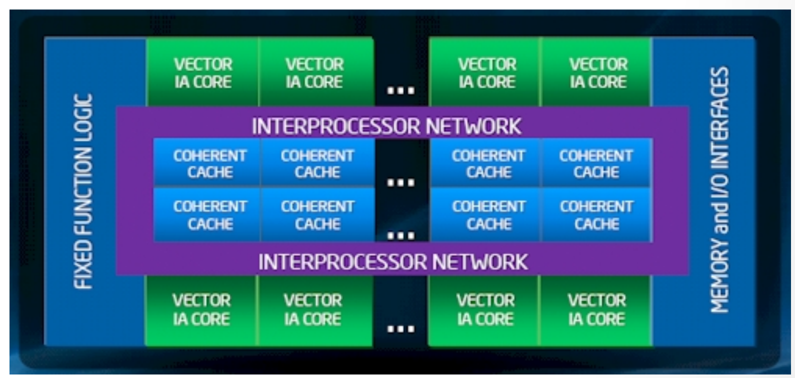
- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

Experimental testbed

- Stampede cluster at TACC
  - 2 Pflops Sandy Bridge + 7 Pflops KnC
  - 6400+ nodes
- A hybrid node configuration
  - 2 Sandy Bridge host and 1 KnC device
  - PCIe between host and device
- Knights Corner
  - Many Integrated Core (MIC)
  - 61 in-order processor cores
  - 512-bit SIMD ALU per core
- Sandy Bridge
  - 8 out-of-order processor cores



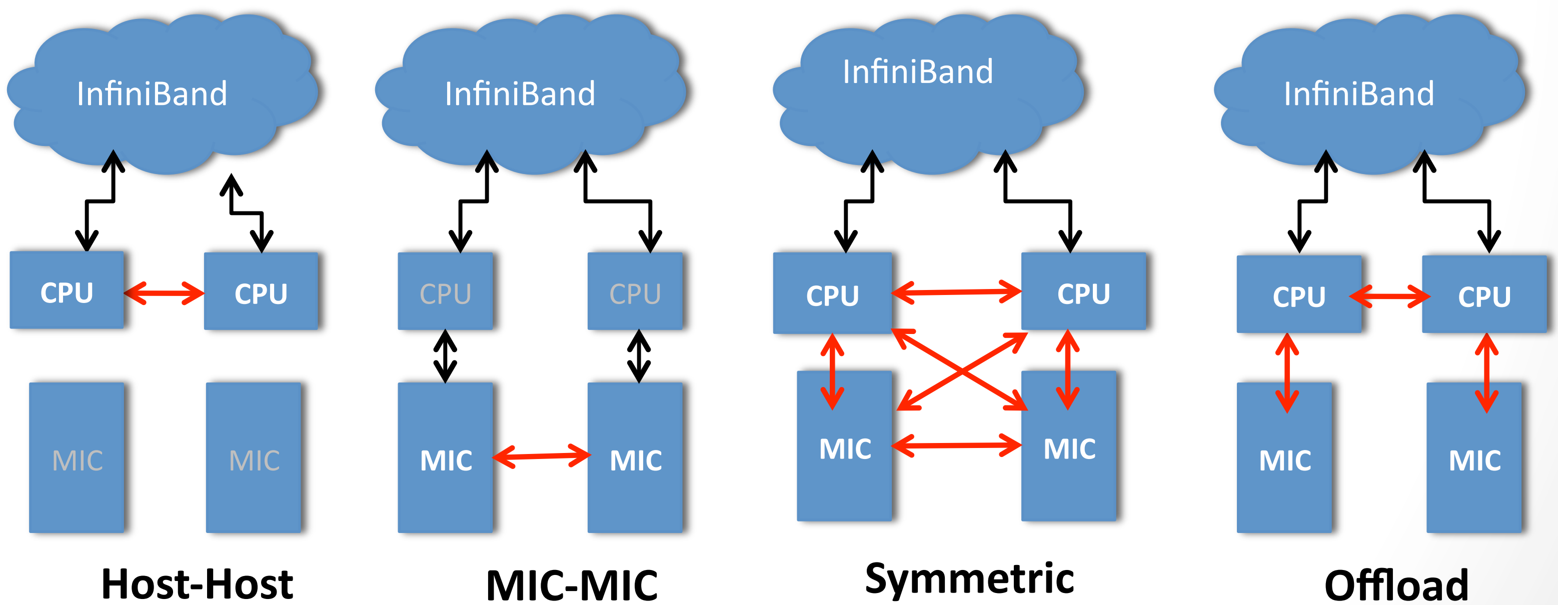
A node configuration with 2 Sandy Bridge and 1 Knights Corner. Image source: TACC



The MIC Architecture
Image source: Intel

Computation modes on Stampede

- Hosts only: Sandy Bridge processors only
- MIC-MIC: Knights Corner processors only
- **Symmetric**: Sandy Bridge and Knights Corner work as SMP nodes
- Offload: Sandy Bridge as host and Knights Corner as device
- Reverse offload: currently not available



Overview

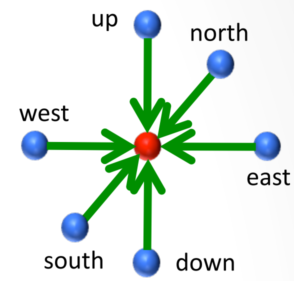
- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

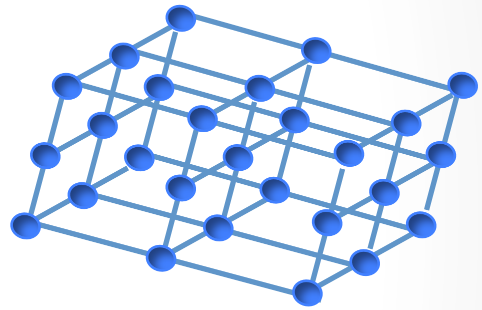
A motivating app in 3D

- Jacobi iterative solver for 3D Poisson’s equation

```
// V, U, and rhs are N x N x N grids
for step = 1 to num_steps{
 for k = 1 to N-2 //Z
  for j = 1 to N-2 //Y
   for i = 1 to N-2 //X: the leading dimension
     V[k,j,i]= alpha *(U[k-1,j,i]+U[k+1,j,i]+U[k,j-1,i]+U[k
          ,j+1,i]+U[k,j,i-1]+U[k,j,i+1])-beta*rhs[k,j,i]
 swap(U,V)
}
```

Un-optimized kernel of Jacobi solver





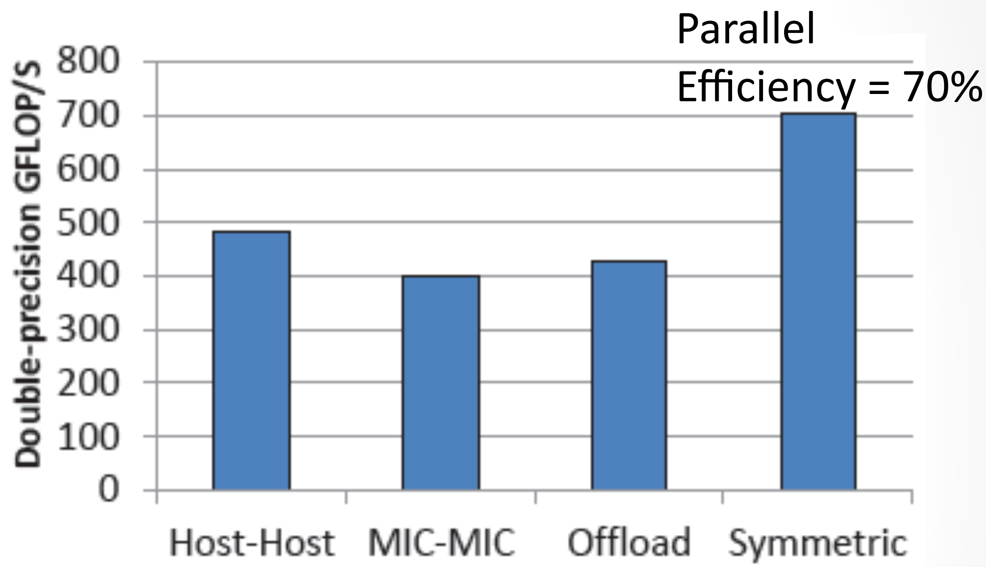
- 2 tier programming MPI+OpenMP
  - MPI to communicate across processors/nodes
  - 3D MPI decomposition
- Kernel optimizations
  - OpenMP with collapse clause on Z and Y dimensions
  - SIMDize along X dimension
  - Loop tiling
  - Stencil unrolling on the time domain *[Chipeperekwa’s MS Thesis,12]*

MPI code for Jacobi solver

```
for step = 1 to num_steps/2{
    #pragma omp for
        Pack data to ghostcells
    MPI_Isend (to left/right/up/down/north/west)
    MPI_Irecv (from left/right/up/down/north/west)
    #pragma omp for
        Unpack data from ghostcells
    MPI_Waitall
    #pragma omp for
        unrolled stencil update
}
MPI_Reduce (residual, MPI_SUM, root= 0)
```

Evaluating the computation modes

- Weak scaling on 16 nodes
- Problem size/node
  - 256x512x512
- "Basic" MPI code variant

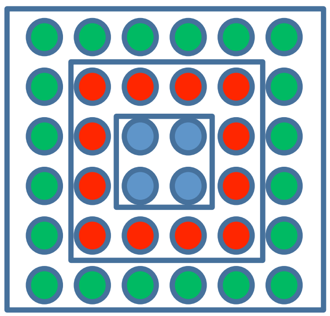


MPI Performance of Jacobi solver on 16 nodes

- Symmetric mode provides the highest performance
- From now on we use **symmetric mode**

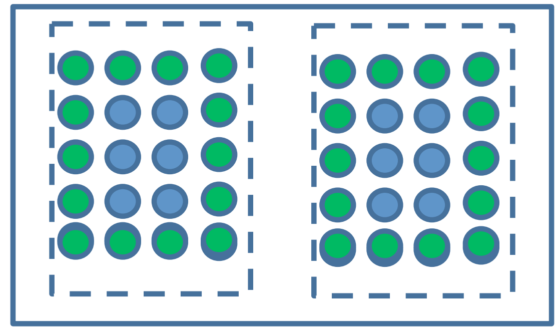
Hiding inter-node communication

- Approach #1:
  - Overlap communication with computations in the inner-most region

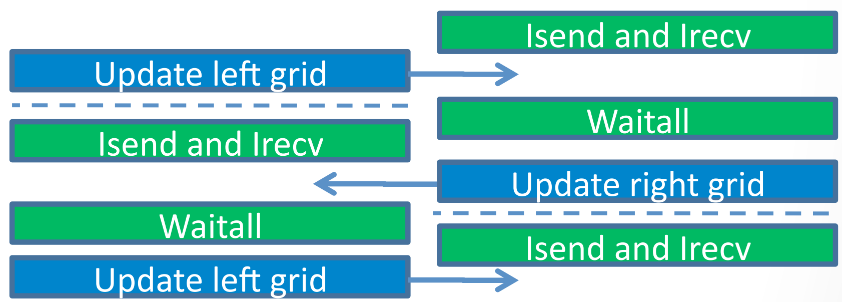


This approach hurts locality

- Approach #2:
  - Over-decompose for pipelining



For each of 3 spatial dimensions



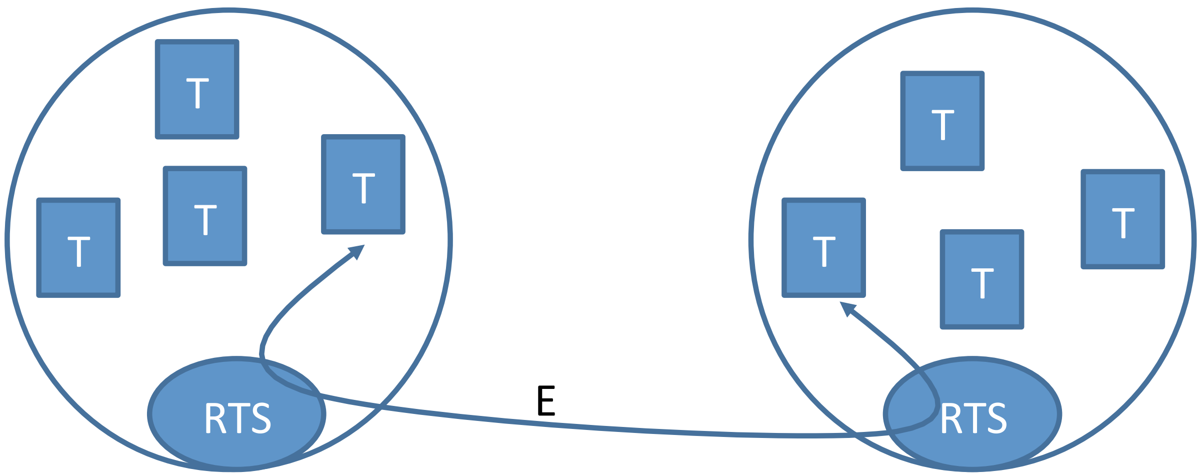
Need system support to:
- Over-decompose the problem
- Schedule comp/comm at runtime

Graph-based, data-driven execution

Overview

- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

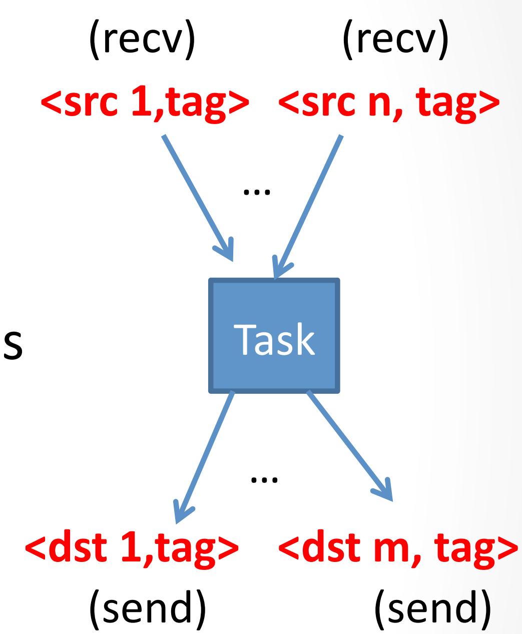
A graph-based execution model



- Program Graph <T, E> and runtime system (RTS)
- A tasks may have input and output edges
- Outputs are delivered by the runtime system
- Tasks idle when waiting for inputs
- When an input arrives, the RTS evaluates task state
  - A task exposes its inputs to the RTS via a firing rule
  - Task state becomes ***runnable*** when all inputs are available
- A task will be scheduled by the runtime system when
  - Task is in runnable state
  - There are available computing resources

Representing MPI program as task graph

- Vertices: virtualize a process to multiple tasks
  - Process id -> task id
  - Number of processes -> graph size
  - Task id and graph size are determined at runtime
- Edges: treat MPI send/recv as task dependencies
  - Input edge: message <source, tag>
  - Output: message <dest, tag>
  - Cycles are allowed
  - Inputs and outputs may change during execution



Overview

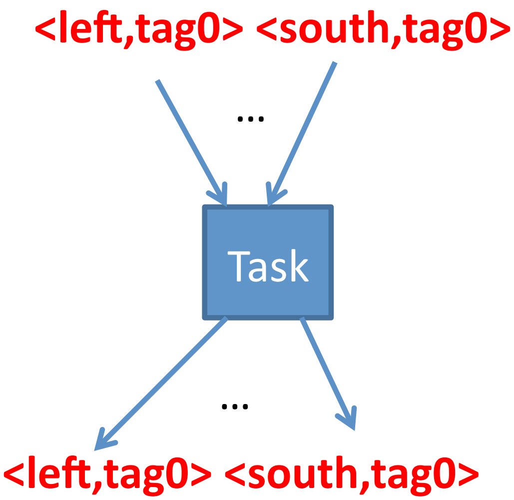
- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

Bamboo Programming Model

- Matching-regions
  - Determine graph's inputs/outputs
  - Contain send/receive matchings
  - No matching is allowed across matching-regions
- Send/receive blocks
  - Statements in SBs are independent of those in RBs
  - If a send must go after a Recv, place both in a receive block
- Computations (optional)
- Treat collective as a set of point-to-point primitives

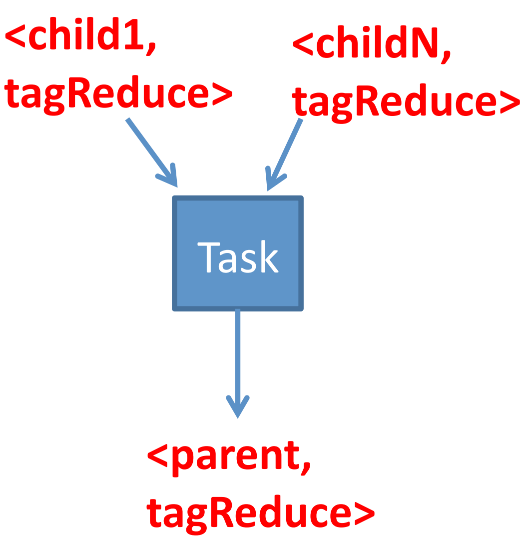
```
#pragma bamboo olap
for step = 1 to num_steps/2{
   #pragma bamboo send
   {  #pragma omp for
        Pack data to ghost cells
      MPI_Isend (to left/right/up/down/north/west)
   }
   #pragma bamboo receive
   {
      MPI_Irecv (from left/right/up/down/north/west)
      #pragma omp for
        Unpack data from ghost cells
    }
    MPI_Waitall
    #pragma omp for
       unrolled stencil update
}//end for
MPI_Reduce (residual, MPI_SUM, root= 0)
```

MPI code annotated with Bamboo



Jacobi update

```
#pragma bamboo olap
 for step = 1 to num_steps/2{
    #pragma bamboo send
    {}
    #pragma bamboo receive
    {}
 }
```

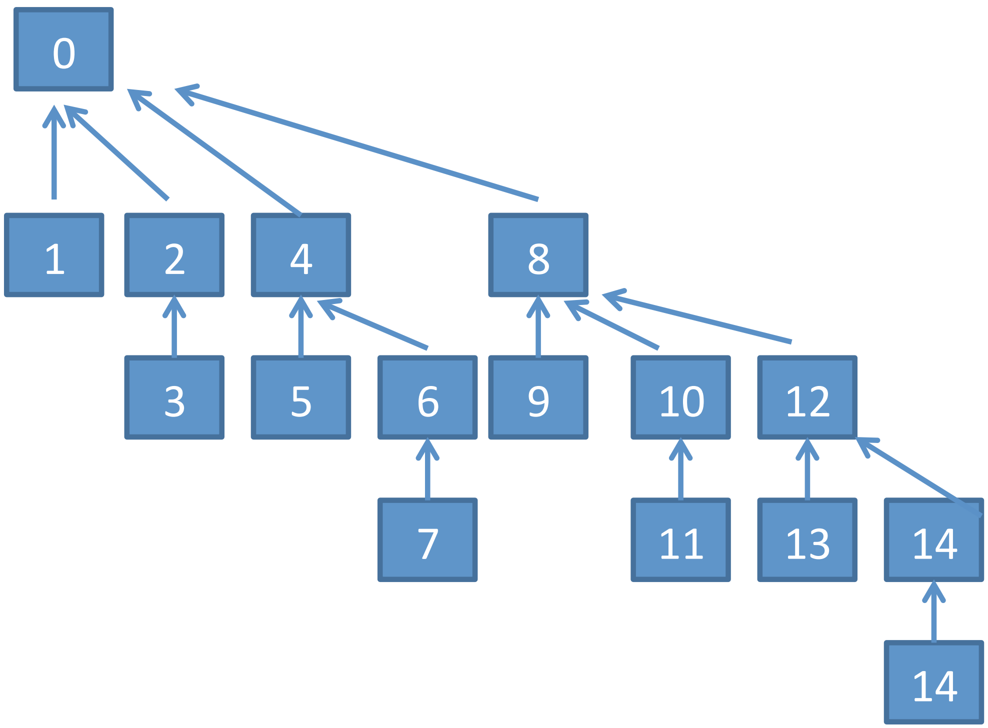


MPI_Reduce

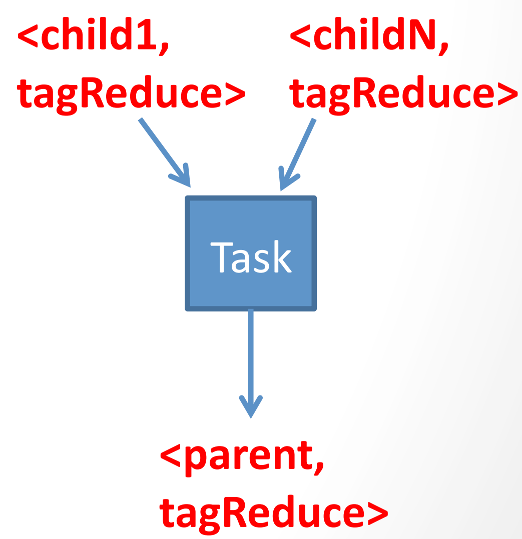
```
#pragma bamboo olap
 {
    #pragma bamboo send
    {}
    #pragma bamboo receive
    {}
 }
```

MPI_Reduce

- Binomial tree



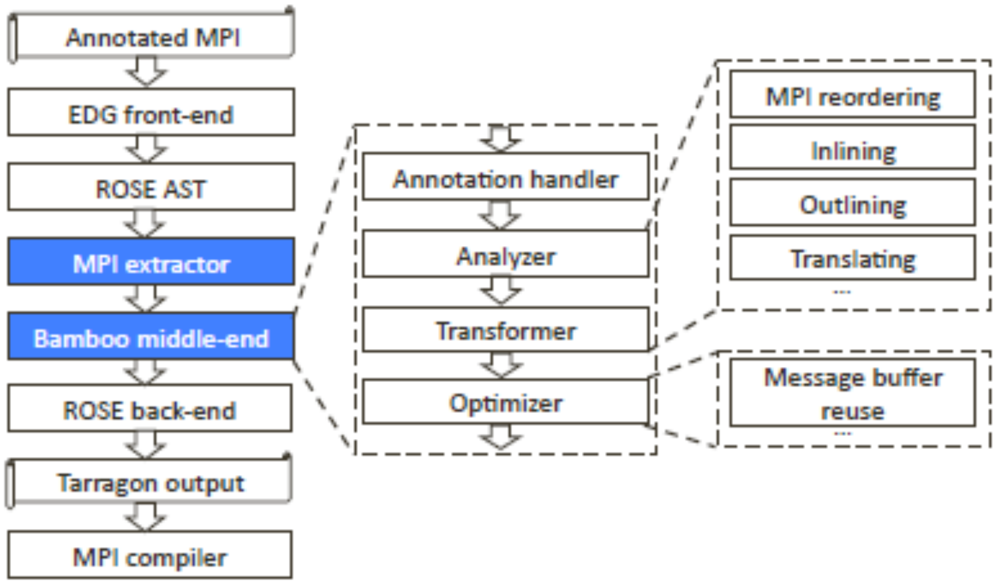

```
#pragma bamboo olap
{       #pragma bamboo send
        if(leaves) Send  to parent
        #pragma bamboo receive
        {
          if(innernodes){
            for (smallest to largest children)
                Receive data from children
            if(hasParent) Send to parent
          }
        }
}
```



- Bamboo automatically replaces MPI_Reduce by an implementation based on point-to-point
- The programmer **doesn't have to implement** collective **nor annotate** the code

Bamboo implementation

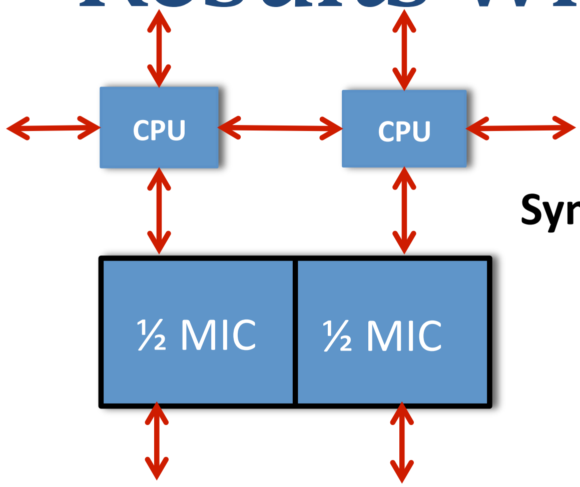
- Tarragon runtime system *[SC 06, DFM 11] [Cicotti's PhD thesis ,11]*
  - Task graph library
  - Runtime system
- Bamboo translator *[SC 12]*
  - Built on top of the ROSE framework http://rosecompiler.org



Overview

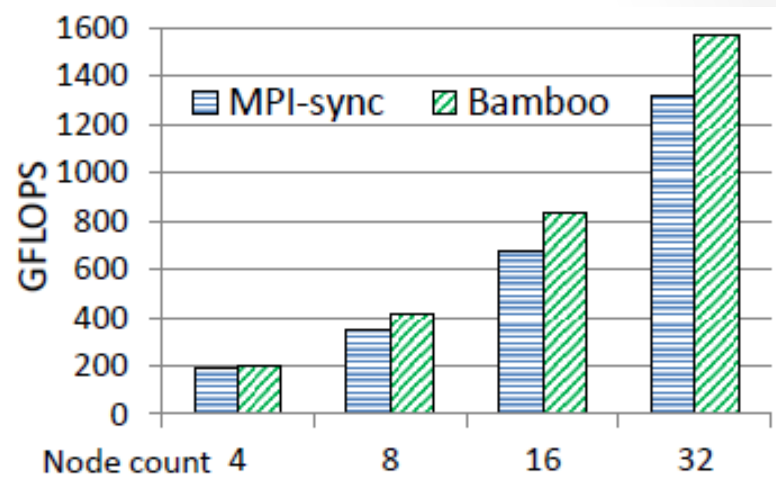
- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

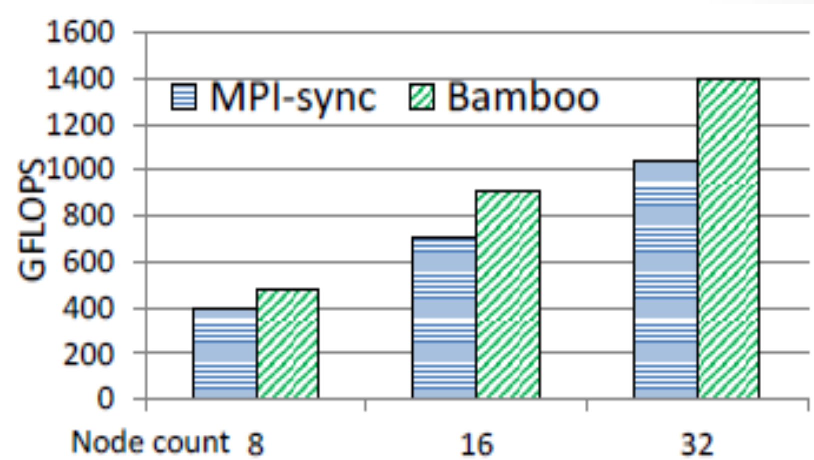
Results with latency hiding



Symmetric mode

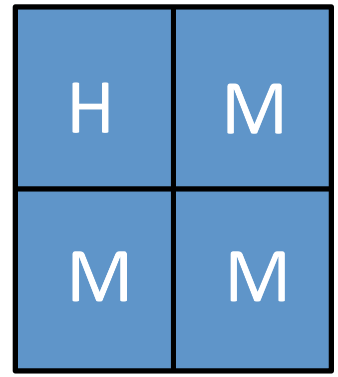
- Weak scaling study (384 $P^{1/3}$)^3
  - Performance improvement over MPI_Sync
    - 24% on 16 nodes
    - 20% on 32 nodes
- Strong scaling study 1024^3
  - Performance improvement over MPI_sync
    - 29% on 16 nodes
    - 32% on 32 nodes



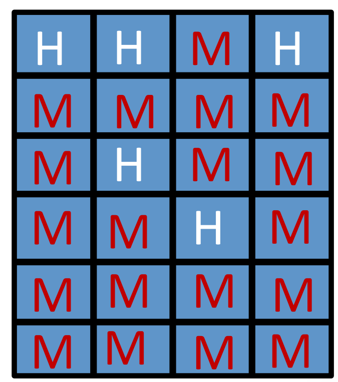


Load balancing: static v.s. dynamic scheduling

- What if
  - Technological changes
  - Node configuration changes
  - Application changes
  - Network traffic always changes
- Static scheduling
  - Low overheads
  - Poorly adaptive
- Dynamic scheduling
  - Higher overheads
  - Highly adaptive



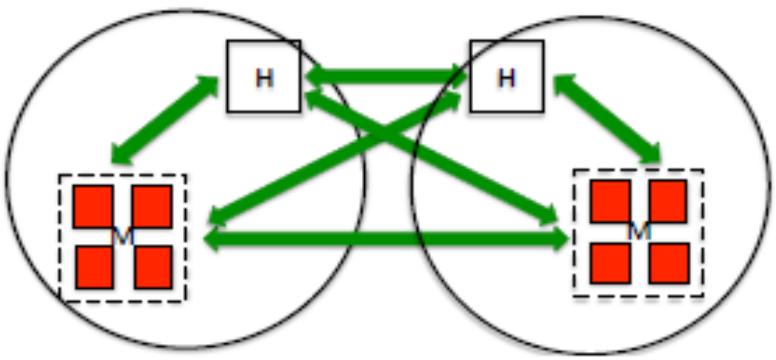
Static scheduling



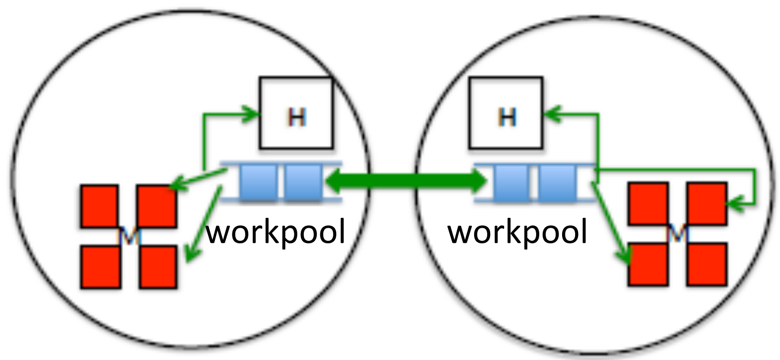
Dynamic scheduling

A rectified symmetric mode to load balance at runtime

- Treat host and device as workers
  - Each processor can act as multiple workers
- Work-pool model
  - Dynamic task distribution
  - A single shared queue or multiple queues with a work stealing policy



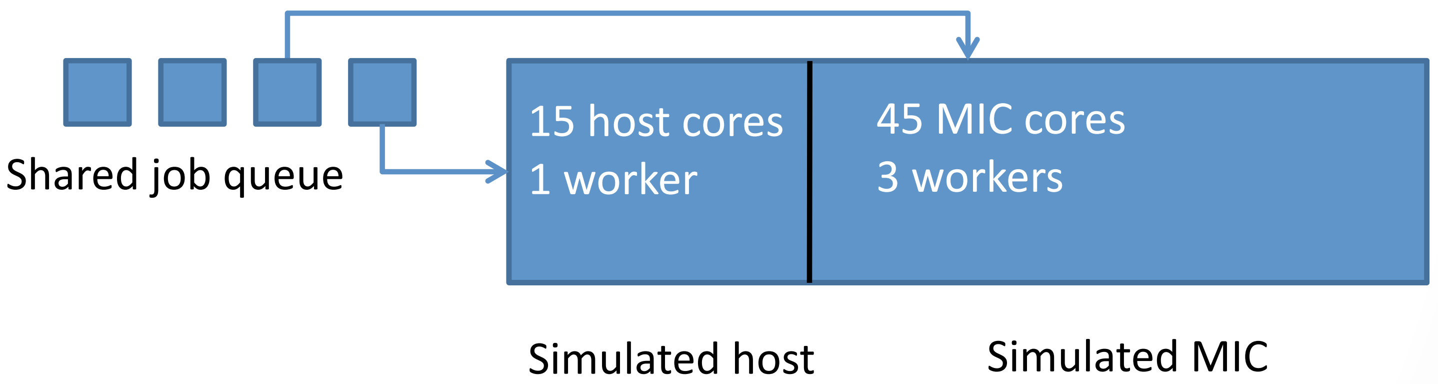
(a) Symmetric mode



(b) A proposed scheme to rectify symmetric mode

Setup to generate unbalanced processor speeds in symmetric mode

- ISAs on host and device are different
  - We use a Knights Corner to simulate a node with host and device
- A simulated hybrid node
  - Simulated device has more cores than simulated host
    - We use ¼ cores as host and ¾ cores as device

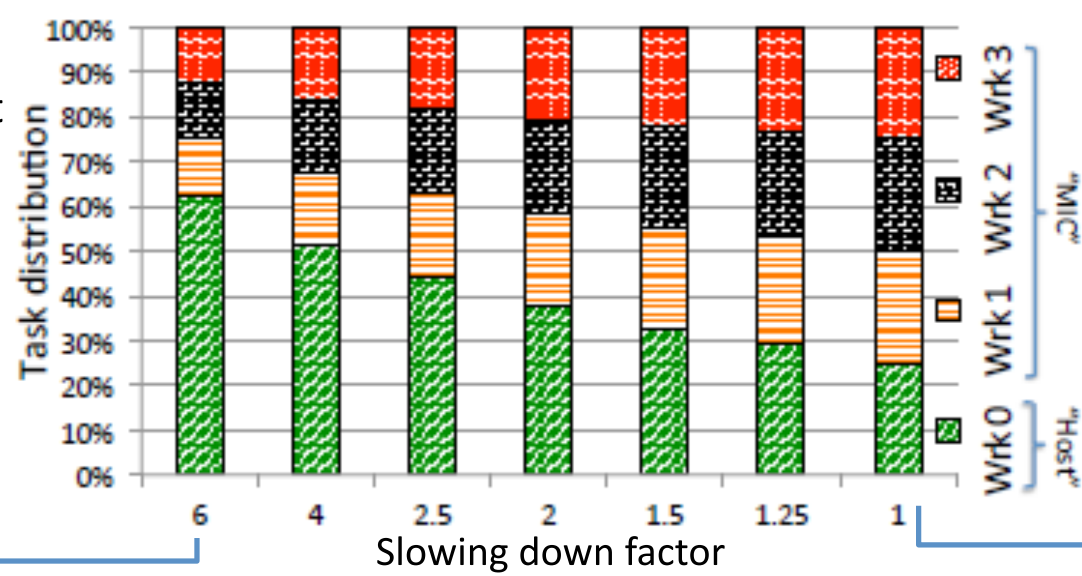


  - A core of simulated device is slower than that of simulated host
    - We add redundant work if a task is scheduled on device to make the core on device slower

Task distribution results with load balancing

- 3 workers on MIC and 1 worker on host
- Slowing down MIC workers to generate load unbalance
- The problem of processor speed variation is eliminated
  - Faster workers takes more tasks
- We need more virtualized tasks when the speed differential among workers increases

1 task to MIC:
2 tasks to host



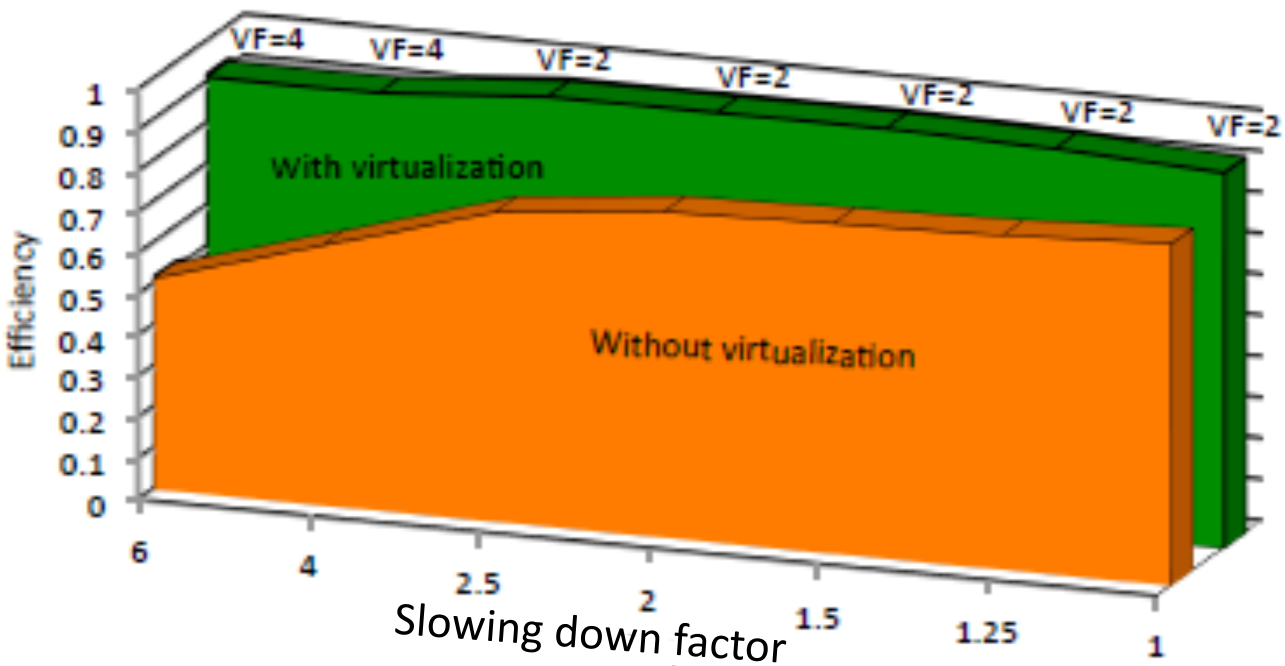
3 tasks to MIC:
1 task to host

11/18/2013

bamboo.ucsd.edu

Load balancing and latency hiding on multiple nodes

- Communication hiding and load balancing can happen at the same time



(b) Efficiency (performance/sustainablePerformance) with and without virtualization on 64 simulated nodes

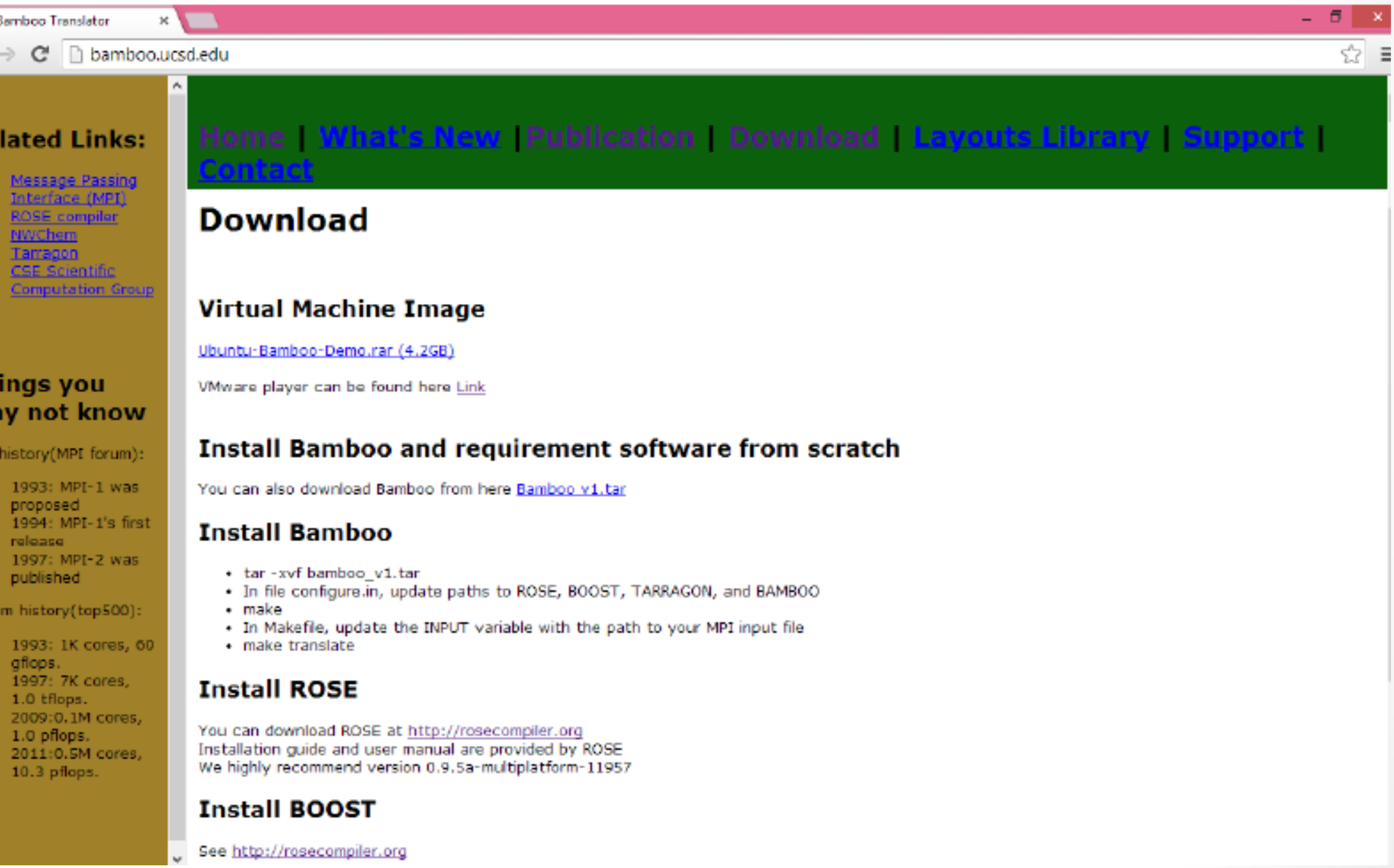
Overview

- Experimental testbed
- MPI programming and optimization
- A graph-based execution model
- Bamboo, a directive-based translator
- Experimental results
  - Latency hiding
  - Load balancing
- Conclusion

Conclusion

- We presented a new programming model that enables homogeneous computing on heterogeneous platforms
- Bamboo translates legacy MPI code into the task graph reprentation
- We demonstrated the benefits of latency hiding and load balancing
- Future work:
  - Implement the rectified symmetric mode
  - Evaluate this mode on real heterogeneous configuration
  - Apply to real code

Download and install



Acknowledgement

- This research was supported by the Advanced Scientific Computing Research, the U.S. Department of Energy, Office of Science, contracts No. DE-ER08-191010356-46564-95715 and DEFC02-12ER26118
- Tan Nguyen is a fellow of the Vietnam Education Foundation (VEF), cohort 2009, and was supported in part by the VEF
- This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575
- We would like to thank Stampede's consultants, who provided quick and thoughtful responses to our questions while conducting this research

References

- [Chipeperekwa's MS thesis, 12] http://cseweb.ucsd.edu/groups/hpcl/scg/papers/2013/TMChipeperekwa-MSReport.pdf
- [Bamboo website] http://bamboo.ucsd.edu/
- [SC 12] T. Nguyen, P. Cicotti, E. Bylaska, D. Quinlan and S. B. Baden, "Bamboo - Translating MPI applications to a latency-tolerant, data-driven form", Proceedings of the 2012 ACM/IEEE conference on Supercomputing (SC12), Salt Lake City, UT, 2012
- [SC 06] Pietro Cicotti and Scott B. Baden, «Asynchronous programming with Tarragon" in Proc. 15th IEEE International Symposium on High Performance Distributed Computing, 2006
- [DFM 11] Pietro Cicotti and Scott B. Baden, Data-Flow Execution Models for Extreme Scale Computing (DFM 2011), Galveston Island, Texas, pp. 28-37, Oct 10, 2011
- [Cicotti's PhD thesis, 11] Pietro Cicotti, Ph. D. Dissertation, Department of Computer Science and Engineering, University of California, San Diego, 2011