Minimal Aggregated Shared Memory Messaging on Distributed Memory Supercomputers

Ben Jamroz [1,2]   John M. Dennis [1]

[1]National Center for Atmospheric Research

[2]DigitalGlobe Inc.

IPDPS Conference
Chicago - May 24, 2016

Point-to-Point Communication

High-performance scientific applications on supercomputers

- Perform calculations in parallel on many nodes
- Each node has its own distributed memory space
- Connected by a network
- Processes communicate by sending messages through the network
  - scatter, gathers, broadcasts, reductions, point-to-point

Point-to-point communication is ubiquitous in high-performance applications

- Spatial domain decomposed onto multiple processes
- Some spatial correlation

Numerical discretization of Partial Differential Equations

- Finite Volume (FV), Finite Element (FE), Discontinuous Galerkin (DG)
- Apply forward operators - derivatives, explicit time stepping

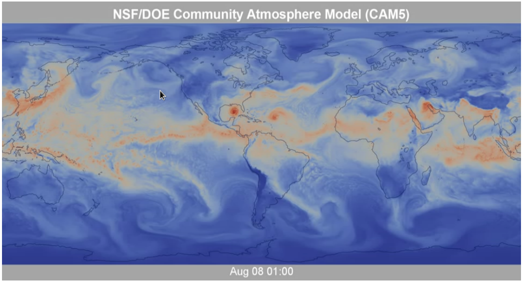
Point-to-Point Communication in Climate Applications

Community Earth System Model (CESM) NCAR and DOE climate model

- Fully coupled climate model
- Many components: atmosphere, ocean, land model, etc.

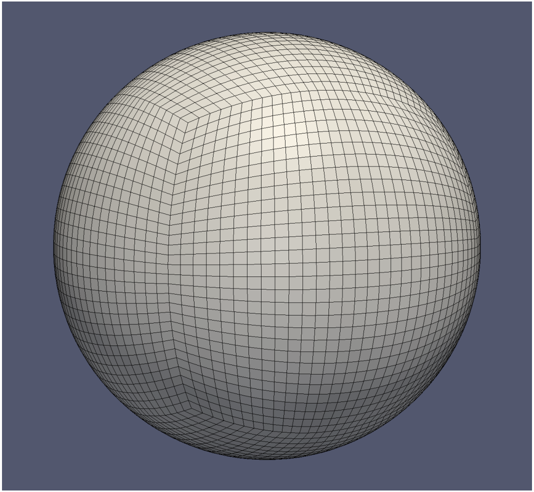
Community Atmosphere Model - Spectral Element (CAM-SE)

- Significant percentage of time spent in climate simulations
- Finite element discretization
- Explicit time stepping in production climate simulations



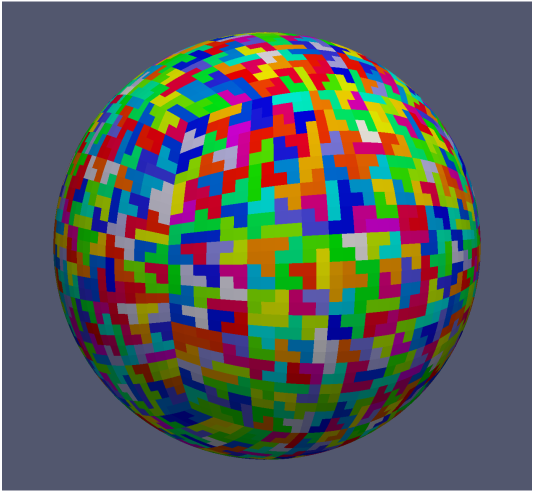
CAM-SE Discretization and Domain Decomposition

Cubed-sphere grid: Ne=30, 1° atmosphere resolution, 1800 processes



CAM-SE Discretization and Domain Decomposition

Cubed-sphere grid: Ne=30, 1° atmosphere resolution, 1800 processes



Increase Scalability of Climate Applications

Climate applications often want fast wall clock turnaround

- Faster throughput of scientific results
- Use as much of the supercomputer as reasonable
- More cores + fixed problem size: strong scaling

Many applications have limited scalability

- Wait longer for results - less science
- Use less of the machine
- Inefficient use of resources

Multi- and Many-Core architectures

- Hardware trend: more low frequency cores
- Require even more scalable algorithms to take advantage

Exascale supercomputers

Desire:

Increase the scalability of climate applications on supercomputers

Latency and Bandwidth Costs

Point-to-point messages incur latency and bandwidth costs

- Latency: total overhead of sending and receiving a message
- Bandwidth: the rate at which data is sent

Strong scaling (fixed problem size - increase # processes)

- Less computation - **More communication**
- Latency costs fundamentally don't scale
- More messages flowing through the network
- Contention for hardware resources

Problem:

Point-to-point messaging limits scalability

MPI 3.0 Standard Supports Shared Memory

In the past MPI typically meant fully distributed memory

- Pro: Don't need to worry about race conditions (with correct API calls)
- Con: Sending a lot of messages can limit scalability

OpenMP used to (mainly) reduce the number of messages

- Pro: Retain some scalability
- Cons:
  - Race conditions
  - False sharing
  - Complexity of behavior

Other shared memory options:

- Posix Inter Process Comm. (IPC), Unified Parallel C/Software (UPC/UPS)

These two are combined in many applications to get the advantages of both

- Efficient MPI+"X" parallelism requires expertise in MPI and X

MPI 3.0 Standard Supports Shared Memory

MPI 3.0 added support for the explicit use of shared memory

- Processes which do share memory **can**
- Processes can read and write to a common shared location
- Similar to one-sided communication

One-sided MPI communication (PGAS)

- Remote Distributed Memory Access (RDMA)
- MPI_Win_allocate allocates "windows" available to other processes
- Get/Put calls to read/write data
- MPI_Win_fence to synchronize data

MPI 3.0 Standard Supports Shared Memory

Shared Memory MPI follows similar construct

- Remote Memory Access (RMA)
- Implementation aware of which processes share physical memory
- MPI_Comm_split_type creates sub-communicator of processes on a node
- MPI_Win_allocate_shared creates windows available to processes sharing the same physical memory
- MPI_Win_shared_query pointer access to shared memory
- MPI_Win_fence synchronize shared memory

MPI shared memory characteristics

- Continuous or “noncontinuous”
- Need to be aware of page boundaries: “page-aligned” buffers

T. Hoefler *et al*. JoC 2013

Benefits and Use of MPI Shared Memory

Shared memory features allow

1. Grouping of MPI processes (sub-communicators) which share memory
2. Allocation and use of data shared across MPI processes on a node
3. Synchronization of processes and data on a node

MPI shared memory instead of OpenMP

- OpenMP parallel regions
- OpenMP data races
- Simpler parallelism (one level of parallelization at computation)

Need to handle interprocess communication (shared memory)

- Simple in point-to-point applications

Solution:

Use MPI shared memory features to implement an efficient point-to-point communication scheme to reduce latency costs

Point-to-Point Communication with MPI

Typical point-to-point MPI communication

1. Call MPI_Isend, MPI_Irecv on all messages
2. Call MPI_Waitall on all messages

Underlying MPI implementation

- Each message is independent
  - Cannot aggregate messages to reduce network communication
  - Limitation of the MPI_Isend/Irecv API
  - In contrast to reductions, broadcasts, scatter, and gathers
- Can utilize shared memory to send messages between processes on-node

We'll modify this paradigm for our new communication scheme

Aggregate Inter-Node Messages

Efficient hierarchical communication scheme

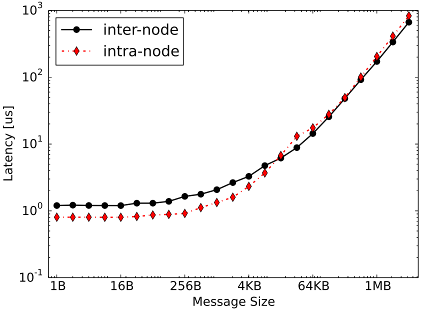
- Distributed memory messages - between nodes
- Shared memory communication - within nodes
- Reduce the number of inter-node messages

Send only one message between node pairs

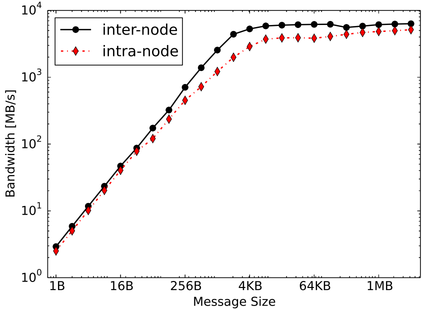
- All messages from node A to node B written to shared buffer
  - Aggregation
  - All processes in node A which send a message to a process in node B
- One MPI process (on node A) sends message A->B
  - Reduces latency costs
- One MPI process (on node B) receives message A->B
  - Message received in MPI shared buffer
  - Other intranode processes read data from shared buffer
- Minimal amount of network messages
  - Fewer messages $\rightarrow$ less contention (interference)

Fewer But Larger - Less Latency? Better Bandwidth?

The Ohio State University (OSU) point-to-point benchmarks on Yellowstone



osu_latency



osu_bandwidth

Larger messages

- Better bandwidth $\geq$ 4KB
- Potential for less latency? sub-linear below $\approx$ 4KB

Fewer messages

- Less contention for resources - A. Bhatele *et al.* SC 2013

Minimal Aggregated SHared Memory Communication

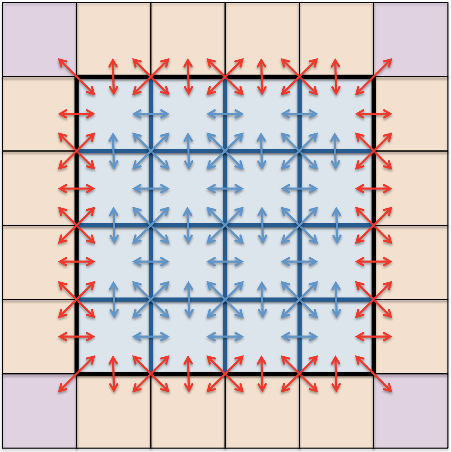
Minimal Aggregated SHared Memory (MASHM) Communication

- Precalculation
  - Form intra-nodal sub-communicator groups
  - Sum sizes of all intra-node messages
  - Sum size of each inter-node message
- Allocate shared memory buffers
  - Intra-node communication
  - Inter-node communication
- Communication
  1. Write data to shared memory (inter-node send buffer or intra-node buffer)
  2. Synchronize the inter-node send buffer
  3. Selected processes send inter-node messages
  4. Synchronize the intra-node buffer
  5. Selected processes wait for inter-node messages to be received
  6. Synchronize the inter-node receive buffer
  7. Read data from shared memory (inter-node recv buffer or intra-node buffer)

MASHM Reduces the Number of Messages

MASHM Communication

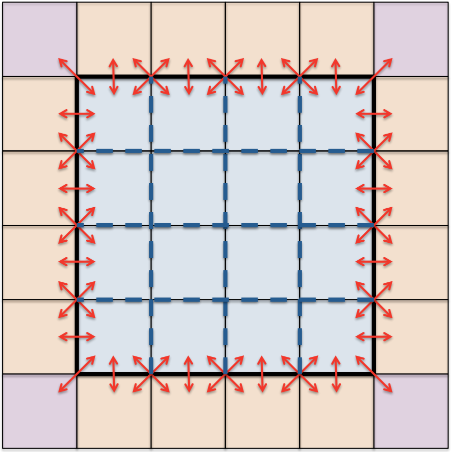
- Eliminates intra-node messages
- Minimized inter-node messages (one per node-pair)



MASHM Reduces the Number of Messages

MASHM Communication

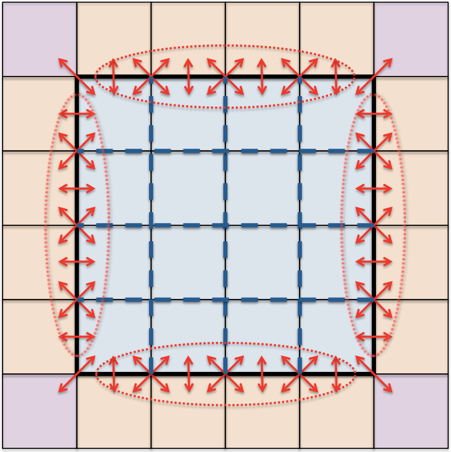
- Eliminates intra-node messages
- Minimized inter-node messages (one per node-pair)



MASHM Reduces the Number of Messages

MASHM Communication

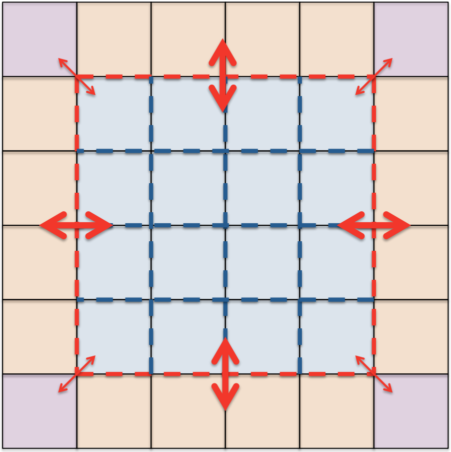
- Eliminates intra-node messages
- Minimized inter-node messages (one per node-pair)



MASHM Reduces the Number of Messages

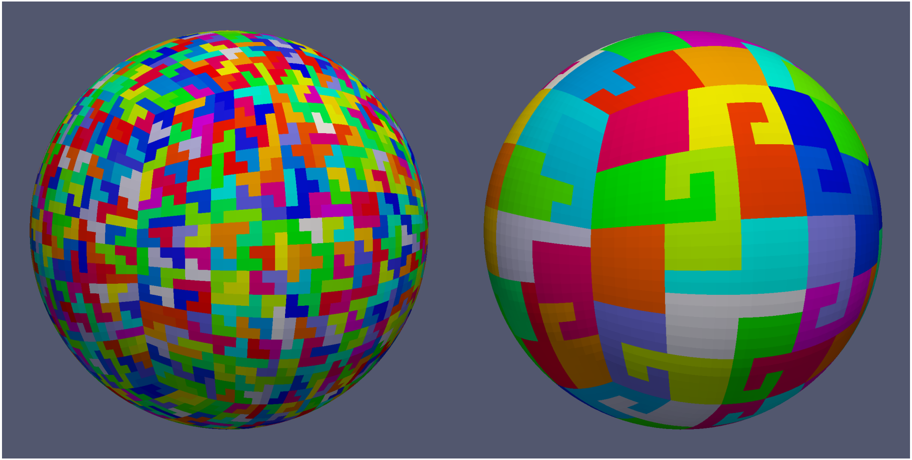
MASHM Communication

- Eliminates intra-node messages
- Minimized inter-node messages (one per node-pair)



MASHM Reduces the Number of Messages

Reduction in number of MPI messages: 1800 processes, 113 nodes



MASHM Overlapped Data Movement

Slight extension to optimize MASHM communication

- Send inter-node messages first
- Move intra-node data while waiting on inter-node messages
- Communication
  1. Write **inter-node** data to shared memory
  2. Synchronize the inter-node send buffer
  3. Selected processes send inter-node messages
  4. **Write intra-node data to shared memory**
  5. Synchronize the intra-node buffer
  6. **Read intra-node data from shared memory**
  7. Selected processes wait for inter-node messages to be received
  8. Synchronize the inter-node receive buffer
  9. Read **inter-node** data from shared memory

This extensions overlaps some data movement: MASHM-ODM in the results

MASHM Open Source Library

Minimal Aggregated SHared Memory (MASHM) Messaging library

- Simplify the use of MASHM communication method and MPI 3.0
- Abstracted details of the methods
- Built on top of MPI
  - Tested with MPICH, MVAPICH, Intel MPI, OpenMPI
- C and Fortran bindings
- Available on github (https://github.com/benjamroz/MASHM)
  - Simple C and Fortran examples
  - Application in the results section is included

Useful for targeting other applications

- Only requires an existing point-to-point communication
- MPI_Irecv, MPI_Isend, MPI_Waitall $\rightarrow$ MASHM library calls

MASHM Library API - Setup

```
/* Initialize the MASHM object */
MashmInit(&myMashm, MPI_COMM_WORLD);

/* Set the number of messages */
MashmSetNumComms(myMashm, numNeighbors);

/* Set the size of each message */
for (i = 0; i < numNeighbors; i++)
  MashmSetComm(myMashm, i, neighbors[i], msgSizes[i]);

/* Set up the communication method */
MashmCommFinish(myMashm);

/* Retrieve pointers for buffers */
for (i = 0; i < numNeighbors; i++) {
  mashmSendBufferPtrs[i] = \
    MashmGetBufferPointer(myMashm, i, MASHM_SEND);
  mashmRecvBufferPtrs[i] = \
    MashmGetBufferPointer(myMashm, i, MASHM_RECEIVE);
}
```

MASHM Library API - Communication

```
/* Fill the buffers with data to be sent */
for (i = 0; i < numNeighbors; i++)
  for (j = 0; j < msgSizes[i]; j++)
    mashmSendBufferPtrs[i][j] = inData(i,j)

/* Send internode messages */
MashmInterNodeCommBegin(myMashm);

/* Exchange intra-node data */
MashmIntraNodeCommBegin(myMashm);
MashmIntraNodeCommEnd(myMashm);

/* Finish receiving inter-node data */
MashmInterNodeCommEnd(myMashm);

/* Read from mashmRecvBufferPtr */
for (i = 0; i < numNeighbors; i++)
  for (j = 0; j < msgSizes[i]; j++)
    outData(i,j) = mashmRecvBufferPtrs[i][j]
```

Anisotropic Laplace's Equation

Laplace's equation is common in many scientific and engineering disciplines

$$-\nabla \cdot \nabla \phi \equiv -\nabla^2 \phi \equiv -(\partial_x^2 + \partial_y^2 + \partial_z^2)\phi = \rho$$

Discretized on a finite difference grid

- Represent derivatives with stencil over adjacent grid points
- 5-point stencil in 2D - 9-pt stencil 3D

Anisotropic Laplace's equation $-\nabla \cdot \mathbf{K} \nabla \phi = \rho$

- Conductivity tensor **K**
- Introduces mixed derivatives: $\partial_x \partial_y, \partial_x \partial_z, \partial_y \partial_z$
- 9-point stencil in 2D - 19-pt stencil 3D

Relaxation of Anisotropic Laplace's Equation

Common method for solving is to apply weighted Jacobi relaxation

- Grid points iterated over and the stencil is applied to the adjacent points
- In a distributed calculation requires a point-to-point message

"Original" point-to-point communication

- Communicate neighboring values (MPI_Isend, MPI_Irecv, MPI_Waitall)
- Apply stencil
- Repeat

Use MASHM or MASHM-ODM communication

- Write to/read from shared memory pointers (inter- or intra-node buffers)
- MASHM-ODM write and send inter-node data first

Experiments: Determine Scalability Improvement

Can the MASHM communication methods increase strong scalability?

How does the total time and time spent in communication change as we add more processes to a fixed problem size?

- Yellowstone at NCAR
  - Two sockets Intel Xeon E5-2670 (Sandy Bridge) 16 cores
  - Intel MPI v5.0.1.035
- Edison at NERSC
  - Two sockets Intel Xeon e5-2695 v2 (Ivy Bridge) 24 cores
  - Cray's MPI Library MPT v7.1.1 (based on MPICH)

Experiment set up

- Apply the communication methods within the weighted Jacobi relaxation
  - Anisotropic Laplace's equation in 3D (19 point stencil)
- Relaxation on a 3D grid with 96 points in each dimension
  - Vary the number of nodes $(2, 4, 8, \ldots, 512)$
  - 16 processes per node on Yellowstone
  - 24 processes per node on Edison

Fewer, Larger Messages

Message statistics on Yellowstone (16 processes per core)

		Original		MASHM	
Resources		Inter-node	Avg. Message	Inter-node	Avg. Message
Nodes	MPI ranks	Messages	Size [B]	Messages	Size [B]
2	32	240	2852.6	2	79872.0
4	64	336	1296.0	6	82944.0
8	128	672	1296.0	14	82944.0
16	256	1344	680.2	72	19498.7
32	512	2304	361.7	188	9787.9
64	1024	4608	312.9	396	8494.5
128	2048	9216	194.3	884	4986.8
256	4096	7680	111.0	1860	3493.2
512	8192	15360	86.4	3812	2710.2

Original communication scheme

- Large number of small messages being sent at high node counts

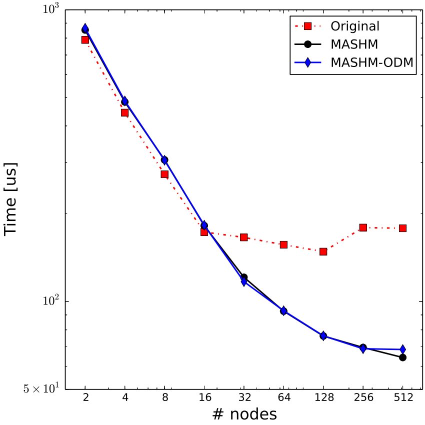
MASHM communication method

- Much fewer inter-node messages
- Message sizes have increased
- Still modest at high node counts $\approx$ 4KB

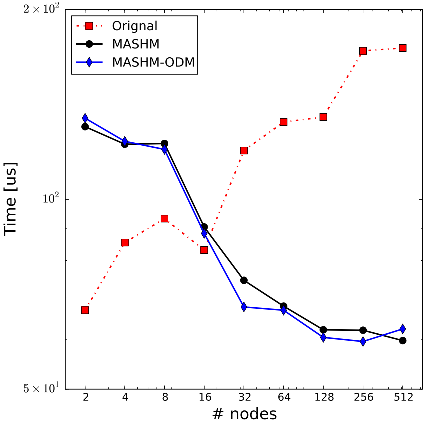
Results on Yellowstone

Average time per relaxation iteration, communication cycle

- All methods scale well out to 16 nodes.
- MASHM scales to 64 nodes, always reduces wall time
- 2.2x faster on 128 nodes



Yellowstone - Relaxation

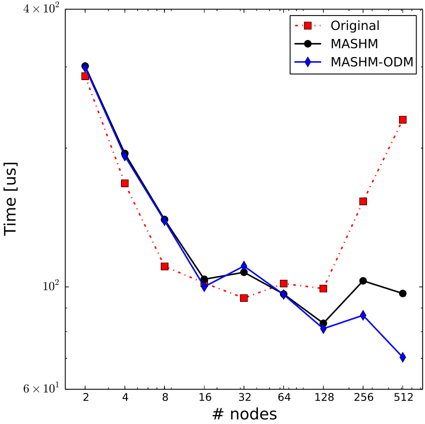


Yellowstone - Communication

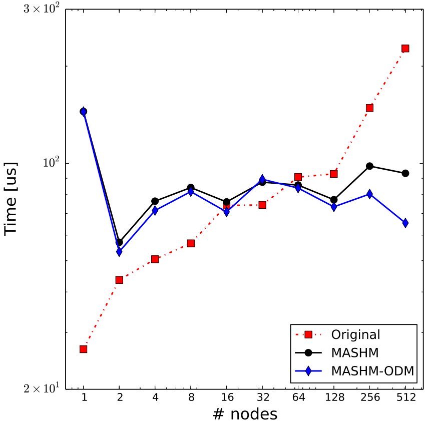
Results on Edison

We see similar results on the Edison supercomputer

- MASHM-ODM faster per by 3.5x
- MASHM faster by 2.4x



Edison - Relaxation

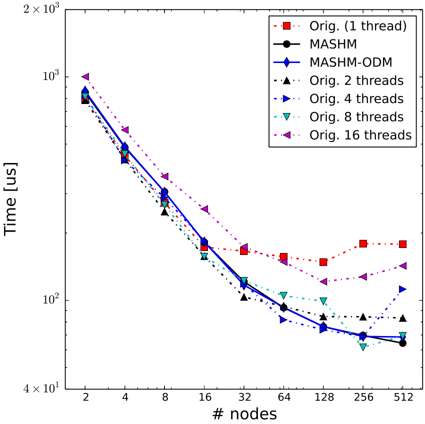


Edison - Communication

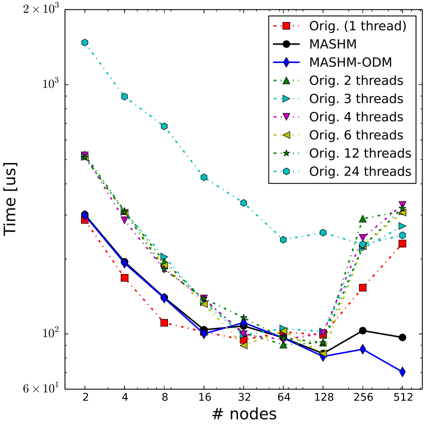
MASHM competitive with OpenMP

OpenMP threading: use all available cores

- OpenMP significantly improves Original performance
- MASHM is competitive or better than optimal OpenMP
- Edison 512 nodes: MASHM 2.6x, MASHM-ODM 3.5x



Yellowstone



Edison

Model Communication Costs

Model communication costs to attribute time to individual components

- Determine which components are most important
- Does MASHM reduce latency costs?

Cost per point-to-point cycle for a single MPI process

$$C = \sum_{i=1}^{N_{\text{inter}}} \left(l_i^{\text{inter}} + \frac{s_i^{\text{inter}}}{b_i^{\text{inter}}}\right) + \sum_{i=1}^{N_{\text{intra}}} \left(l_i^{\text{intra}} + \frac{s_i^{\text{intra}}}{b_i^{\text{intra}}}\right)$$

Approximate this function

- Average number and size of messages $\overline{N_{\text{inter}}}, \overline{s^{\text{inter}}}, \overline{N_{\text{intra}}}, \overline{s^{\text{intra}}}$
- OSU benchmarks latencies and bandwidths $\overline{l^{\text{inter}}}, \overline{b^{\text{inter}}}, \overline{l^{\text{intra}}}, \overline{b^{\text{intra}}}$
- Correction factors to scale bandwidth to "achieved": $\beta^{\text{inter}}, \beta^{\text{intra}} \in [0, 1]$
  - H. Gahvari *et al.* ICPP 2012

$$\overline{C} = \overline{N_{\text{inter}}} \left(\overline{l^{\text{inter}}} + \frac{\overline{s^{\text{inter}}}}{\beta^{\text{inter}}\,\overline{b^{\text{inter}}}}\right) + \overline{N_{\text{intra}}} \left(\overline{l^{\text{intra}}} + \frac{\overline{s^{\text{intra}}}}{\beta^{\text{intra}}\,\overline{b^{\text{intra}}}}\right)$$

Model Communication Costs

MASHM communication costs

- Replace the intra-node costs with data movement $\overline{m}$: 68GB/s $\approx$ 4.25$\mathrm{GB/s}$
- Add an intra-node synchronization cost S
- Correction factors $\beta^{\mathrm{M}}, \beta^{\mathrm{m}} \in [0, 1]$

$$\overline{C_{\mathrm{M}}} = \overline{N_{\mathrm{M}}}\left(\overline{l^{\mathrm{M}}} + \frac{\overline{s^{\mathrm{M}}}}{\beta^{\mathrm{M}}\overline{b^{\mathrm{M}}}}\right) + \overline{N_{\mathrm{intra}}}\frac{\overline{s^{\mathrm{intra}}}}{\beta^{\mathrm{m}}\overline{m}} + S.$$

Fit these cost models to gain insight into communication costs

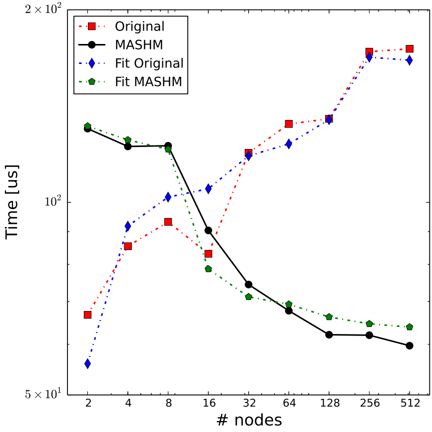
- Least squares using timing data, message statistics, and OSU data
- Parameters - Original
  - $1/\beta^{\mathrm{inter}}$, $1/\beta^{\mathrm{intra}}$
- Parameters - MASHM
  - $1/\beta^{\mathrm{M}}$, $1/\beta^{\mathrm{m}}$, S

Improved Inter-Node Bandwidth

Least-squares solve for parameters

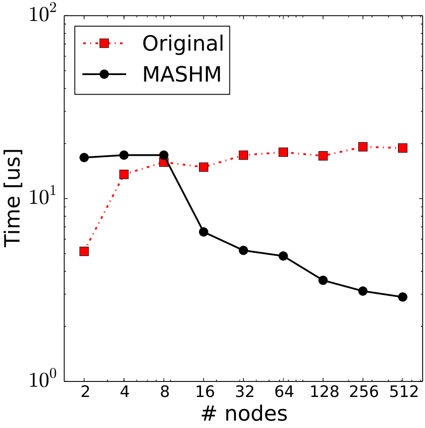
	Original		MASHM		
	β^{inter}	β^{intra}	β^{M}	β^{m}	S [μs]
Value	0.0364	.206	0.318	0.377	59.1

- Inter-node bandwidth correction
  - Original 3.64% $\rightarrow$ large effect of resource contention when many inter-node messages are sent at once
  - MASHM 31.8% $\rightarrow$ aggregation improves the achieved bandwidth rate
- Model implies MASHM improves bandwidth rate
- Break down individual communication costs

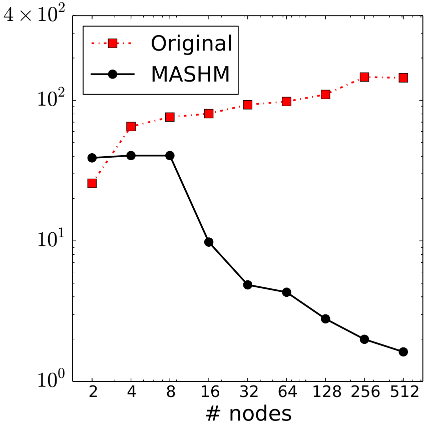


Reduced Latency and Improved Bandwidth

MASHM reduces both inter-node latency and bandwidth costs at moderate to high node counts



inter-node latency

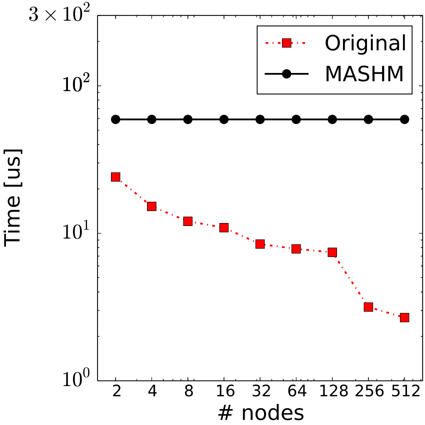


inter-node bandwidth

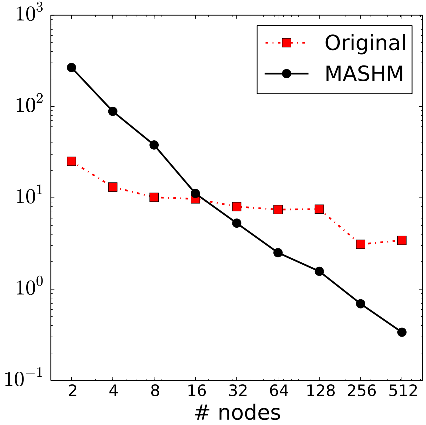
Faster Intra-Node Data Movement

Intra-node synchronization costs dominate at high core counts (3)

Intra-node data movement dominates at low core counts



intra-node latency/synchronization



intra-node bandwidth

Intuition From Model

Model predictions

- Reduced inter-node latency cost - by sending fewer messages
- Reduced inter-node bandwidth costs - by sending larger messages
- Intra-node data movement more efficient at scale
- Intra-node synchronizations

Model can give insight to where MASHM is more efficient

- Which applications?
- Size and number of messages

Conclusion

Minimally Aggregated SHared Memory (MASHM) Communication Method

- MPI 3.0 features to eliminate all intra-node point-to-point MPI messages
- Sends a single inter-node message between node pairs
- Strong scales much better than point-to-point messaging
- Competitive with using OpenMP
- Model of costs: reduces both inter-node latency and bandwidth costs
- Open Source library available at `https://github.com/benjamroz/MASHM`