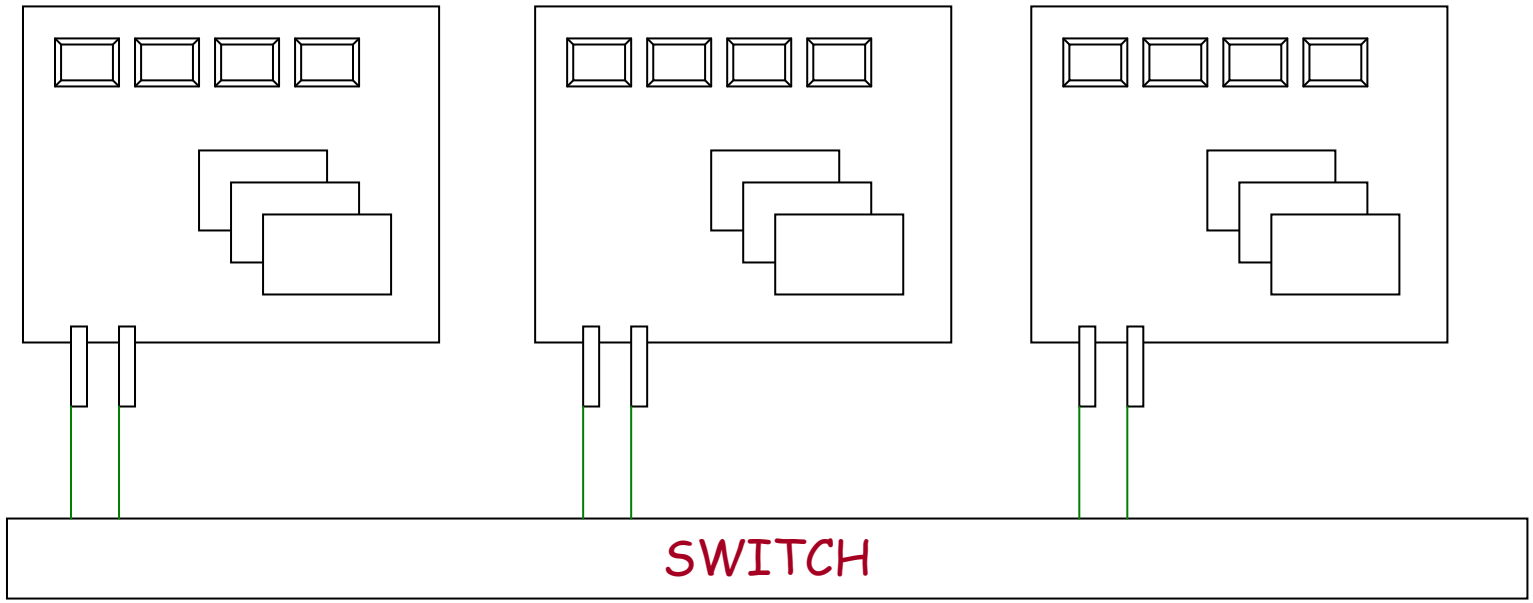
Communication on SMP clusters

Communication architecture

- SMP clusters are built out of multi-processor nodes
  - Memory and communication hardware is shared by all processors in a node



Adapters

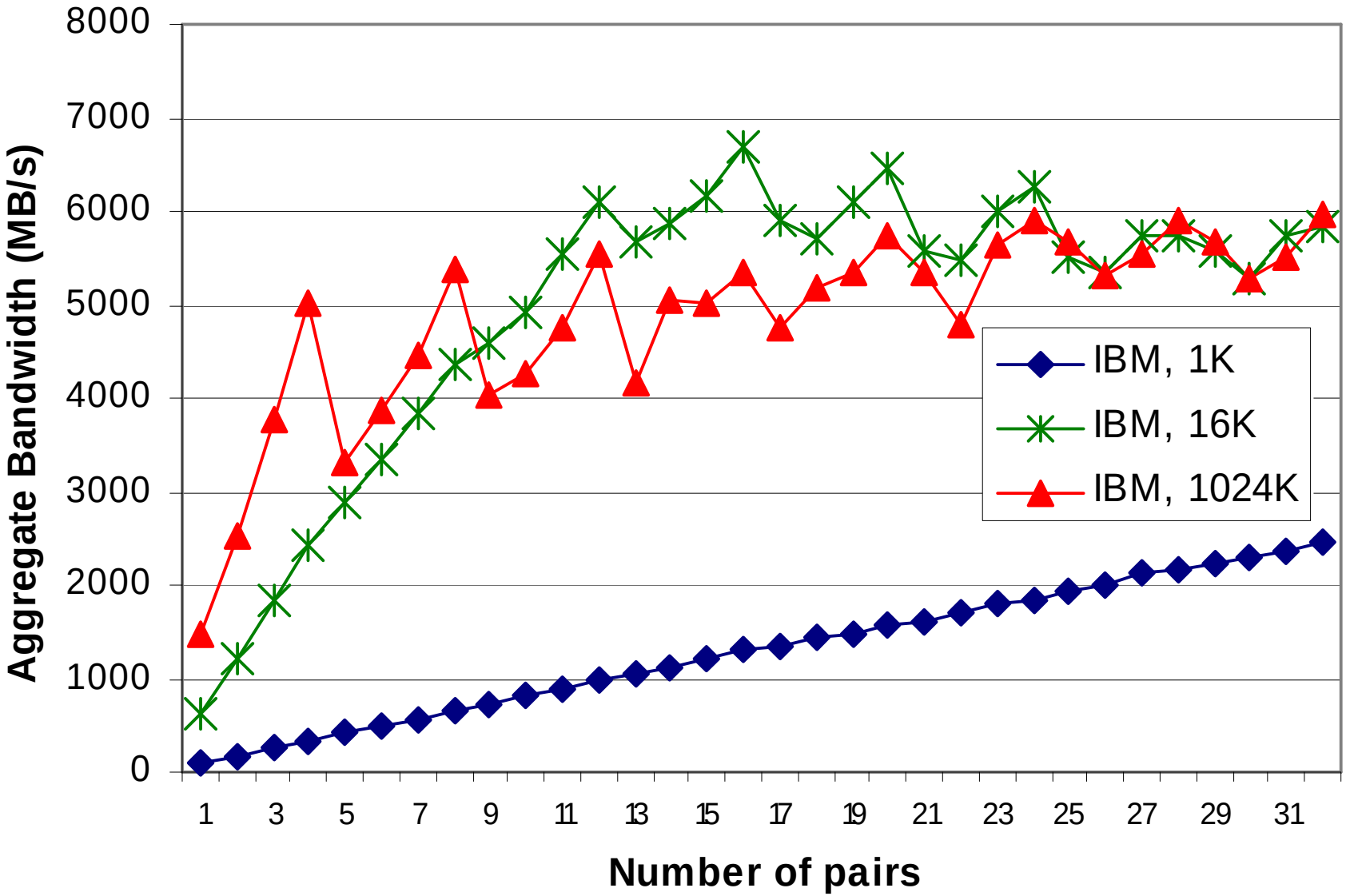
- Nodes usually connect to the communication network via some form of network adapter
- May be multiple adapters for increased bandwidth
  - All CPUs usually have access to all adapters
- Network adapters may be directly connected to the memory system or may be connected to a device bus (e.g. PCI)
  - Device buses usually slower than memory

Shared devices

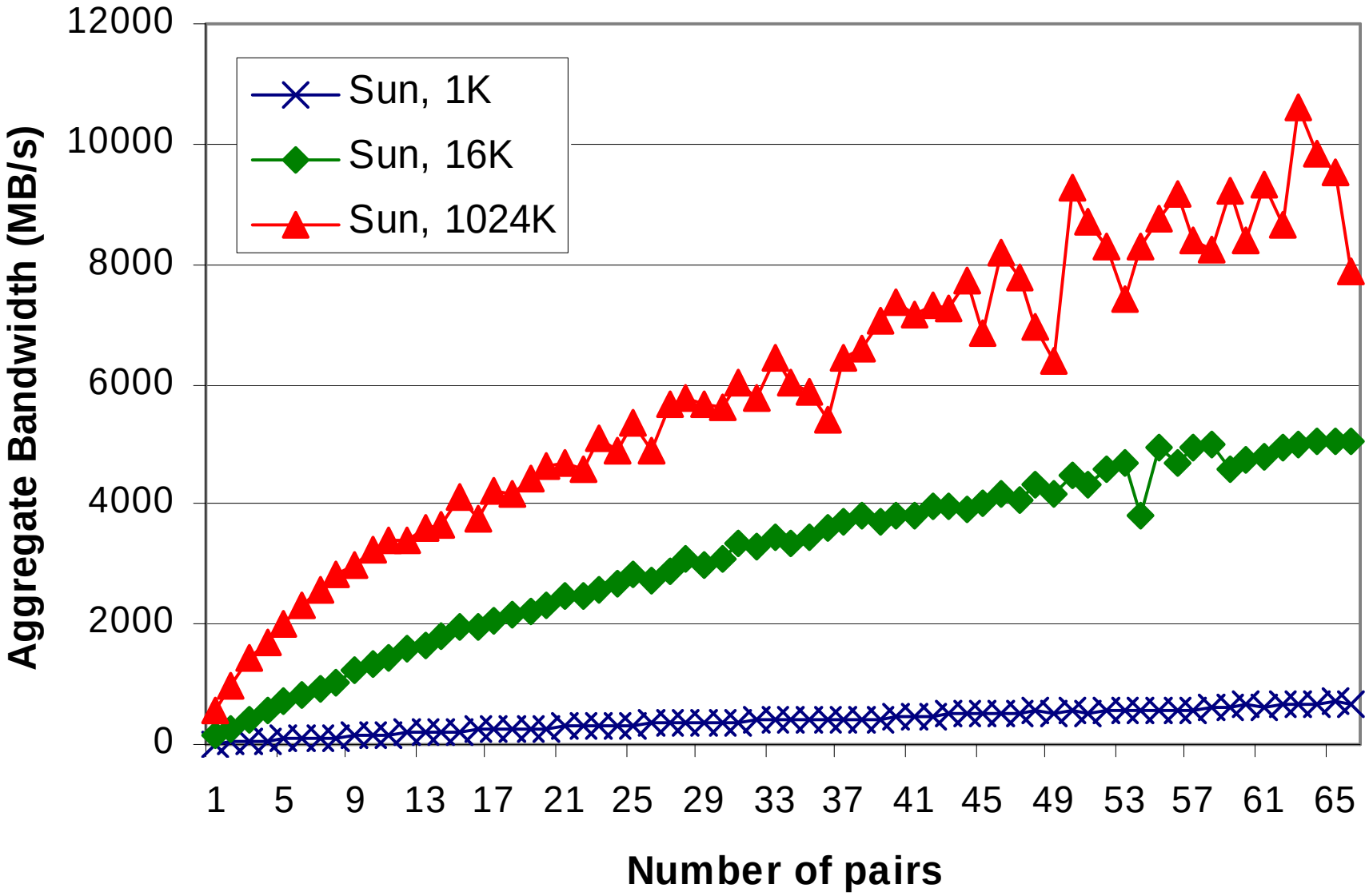
- Network devices are shared (like memory)
  - Similar impact on achievable memory/communication bandwidth
- Peak bandwidth for a single CPU may be very high
- Average bandwidth when all CPUs are contending for the resource may be much less
  - However the aggregate bandwidth (N times the average bandwidth) may not be the same as the peak bandwidth of a single CPU
  - Usually:

    Aggregate > Peak-single > Average

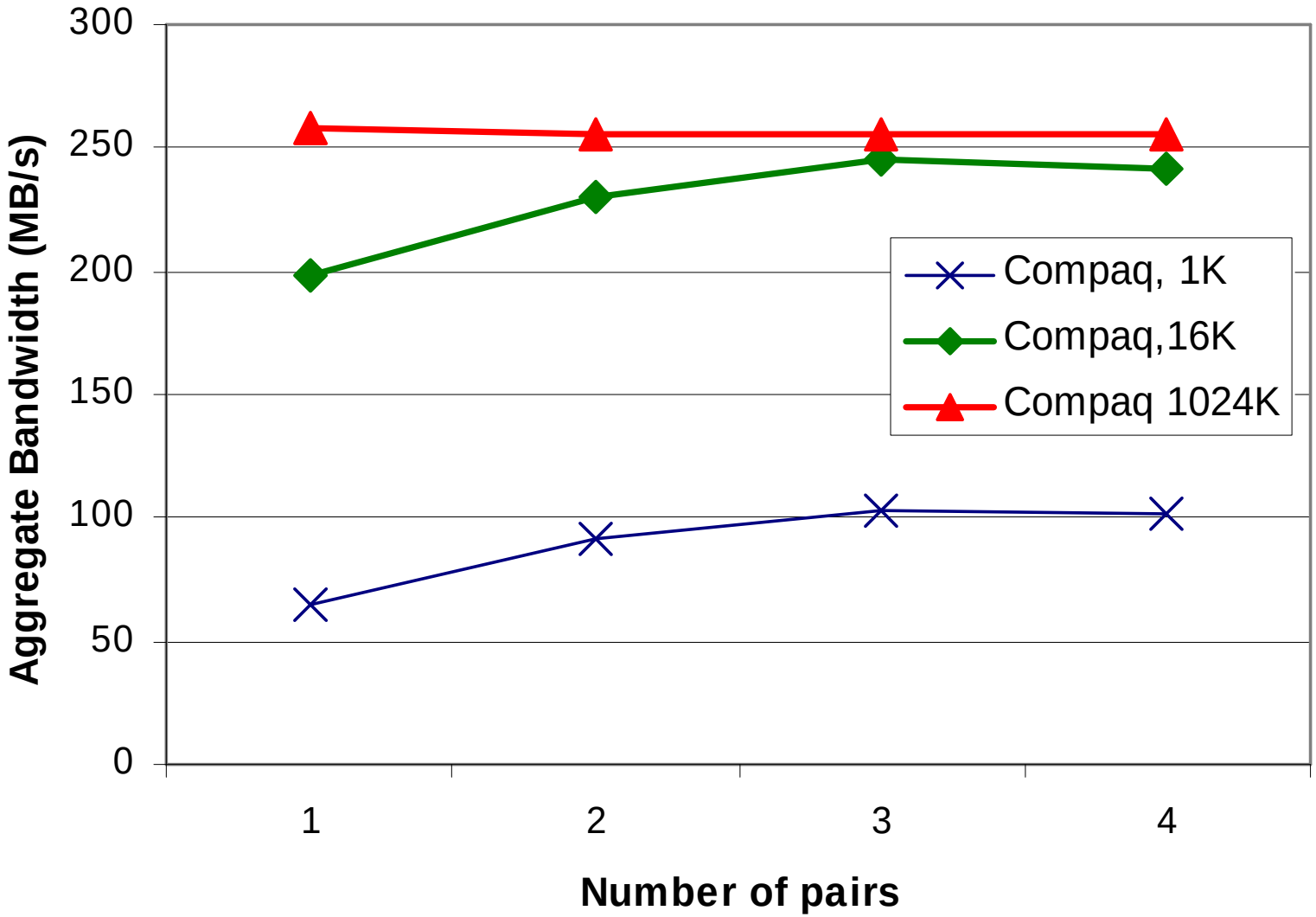
Communication bandwidth – IBM p690+



Communication bandwidth – Sun Fire 15K



Communication bandwidth – AlphaServer



Shared memory

- Normally each process on a SMP system has a separate address space and uses disjoint areas of memory
  - A single process can still utilise multiple CPUs by using threading
  - However read-only memory pages (e.g. code) may be automatically shared between processes by the OS
  - Processes can also create shared memory segments
    - Multiple process can access these at the same time
    - Live until destroyed (not tied to the lifetime of any process)
    - May be read/write

Shared memory segments

- Shared memory segments are important because:
  - Understanding them help you to understand the MPI library
  - They may be a useful tool when optimizing codes for SMP clusters

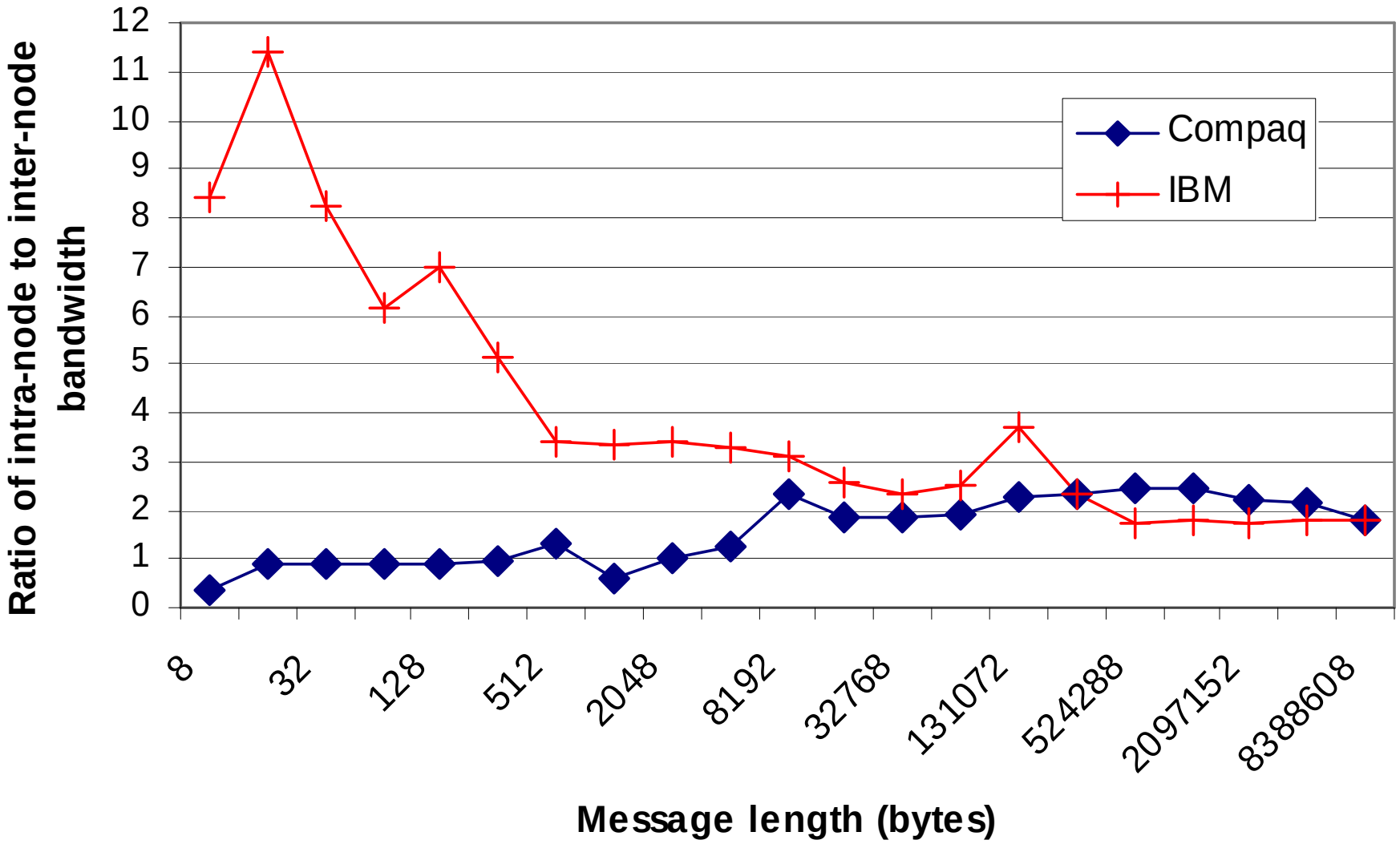
- Syntax for creating shared memory segments is fairly portable across Unix systems
  - Memory consistency rules less so, but things are usually OK if you stick to standard features like mutex/semaphore

How message passing works

- On SMP clusters there are usually two modes of operation
  - One optimized for use within a node
  - One for communicating between nodes

- Message passing performance is usually quite different in the two cases

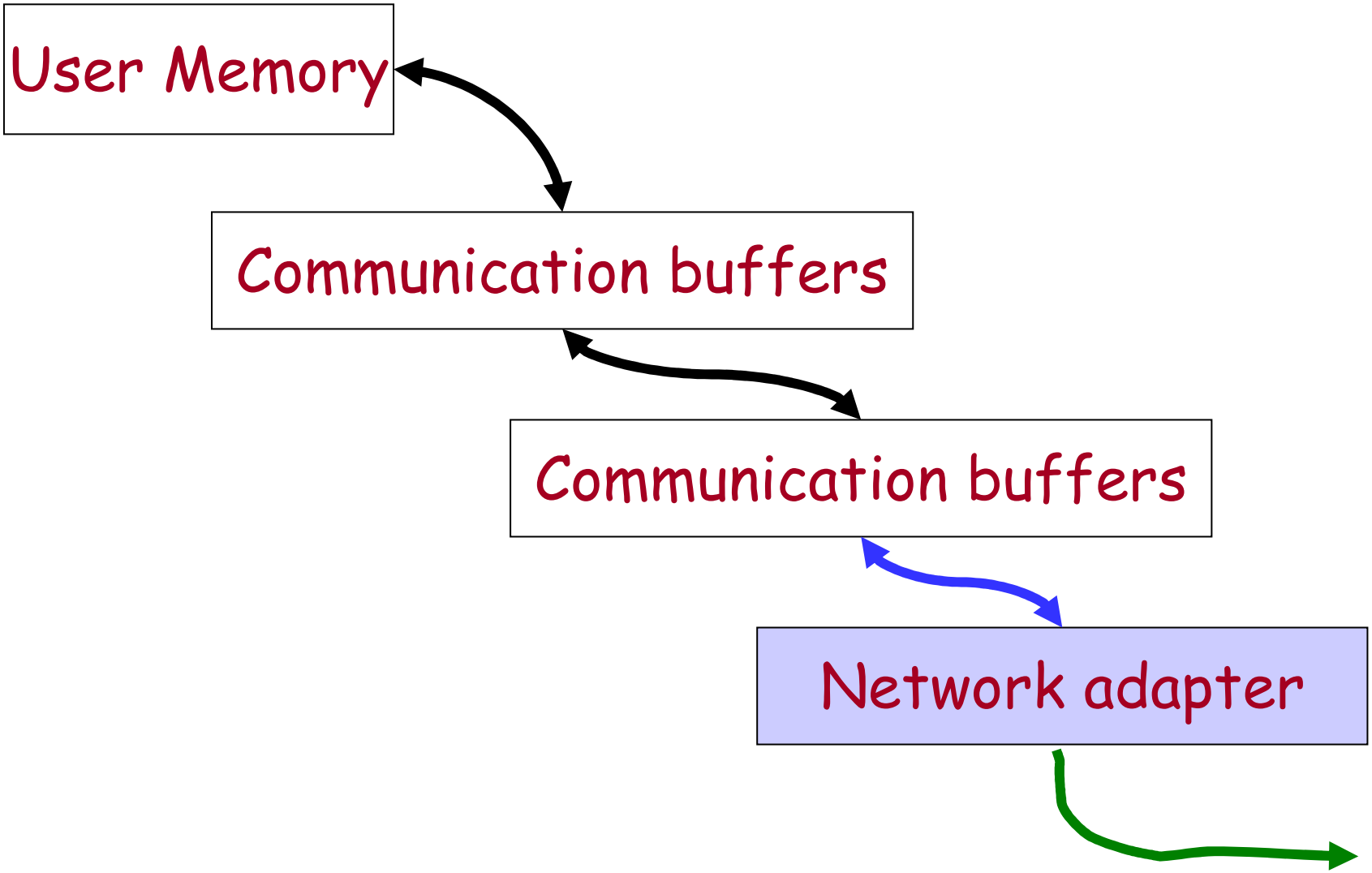
Message passing performance



Communication between nodes

- Data may need to be copied several times within a node. E.g.
  - Copy to internal library buffers
    - Especially if non contiguous MPI data-types are used
  - Copy to system communication buffers
  - Copy into communication adapter
- Same process in reverse for receive
- These are not necessarily single copies. Each stage may be performed in chunks based on buffer sizes or network packet sizes
- Copies to network adapter may be via a device bus

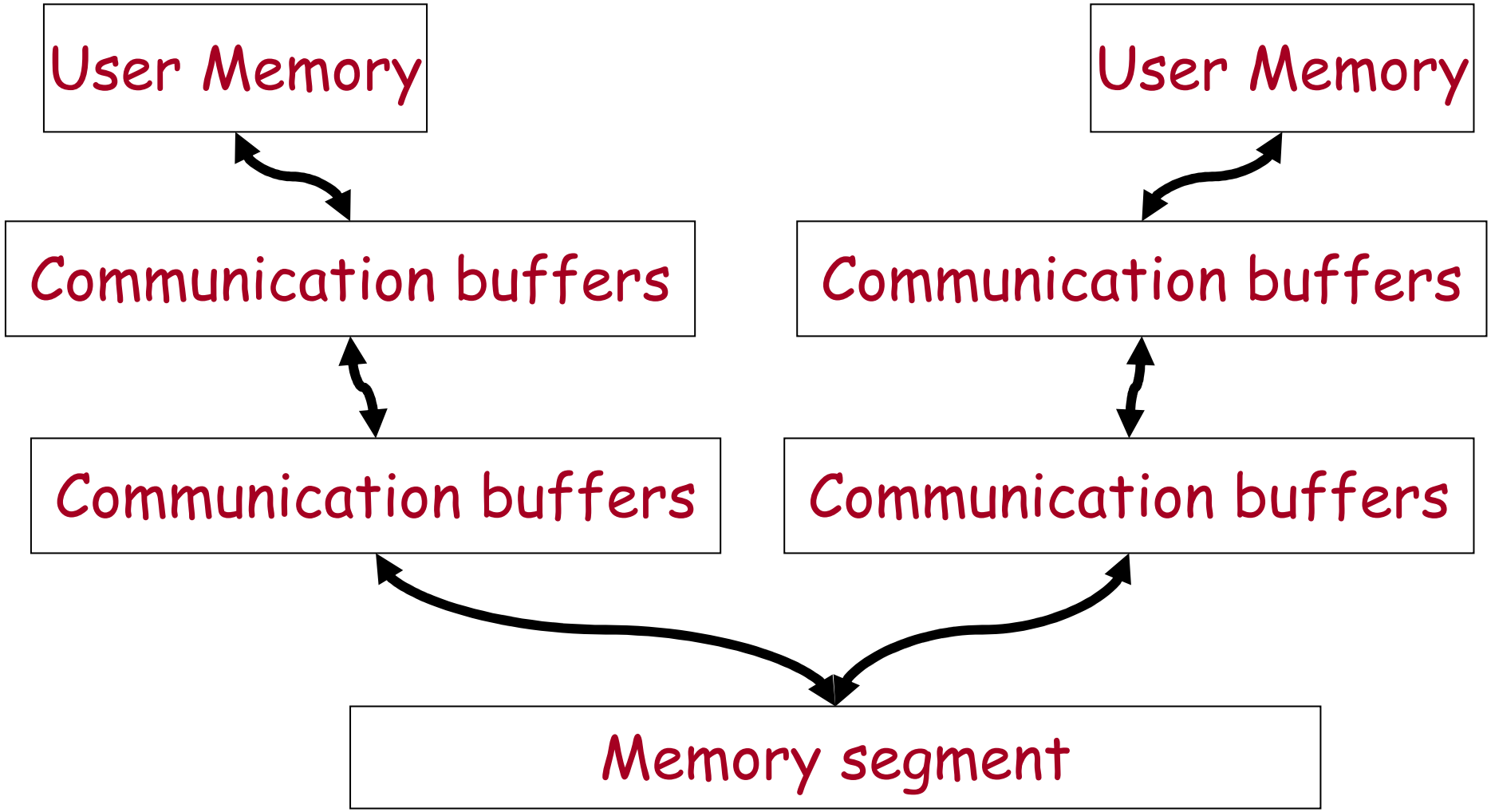
Communication between nodes



Communication within a node

- Between processes in the same node data is copied through a shared memory segment rather than using the communication fabric
  - All copies are at memory speed

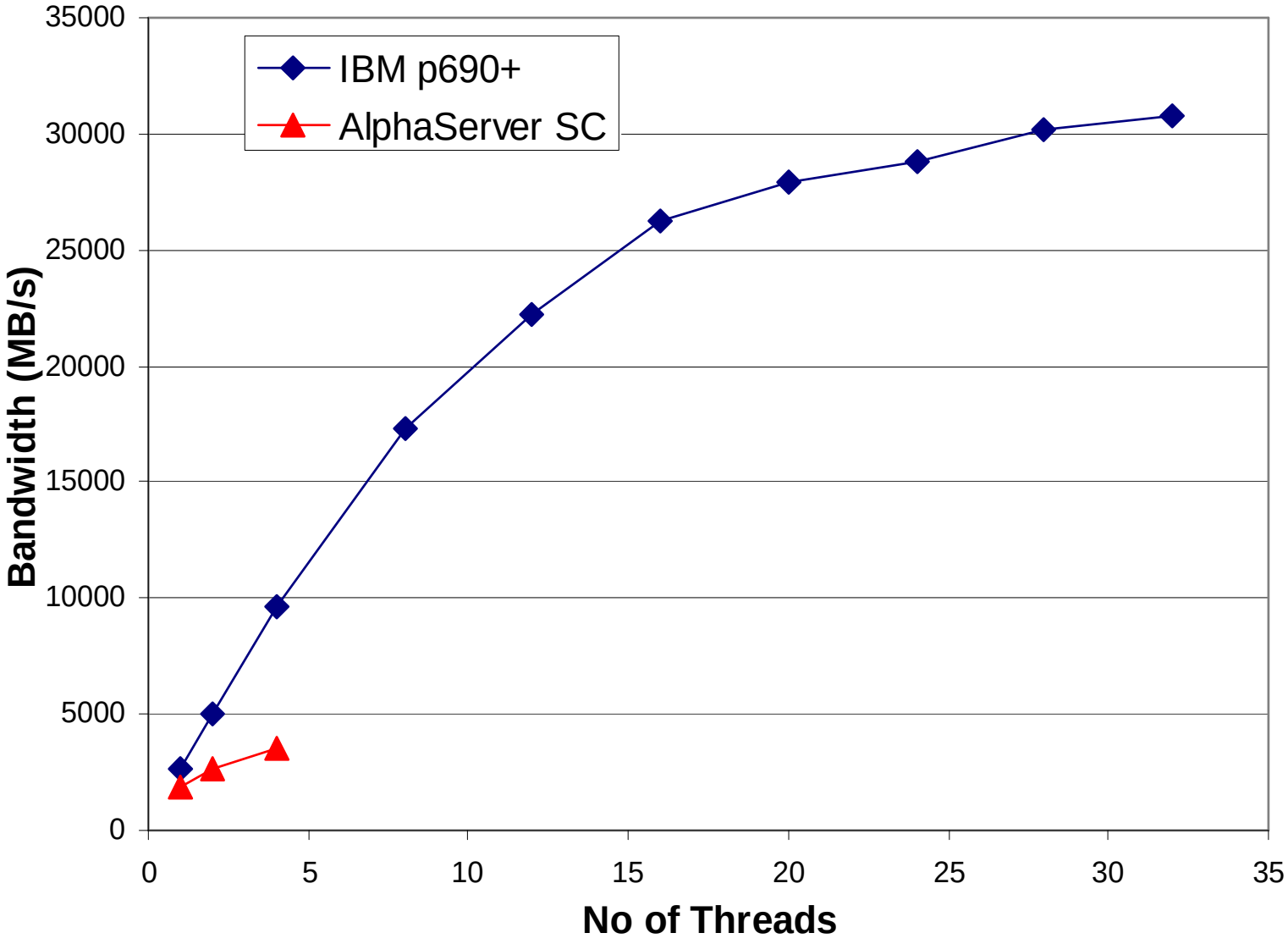
Communication within a node



Memory bottlenecks

- Both Inter and Intra node communications involve memory to memory copying
  - This takes time and therefore contributes to communication cost
  - Bandwidth may be reduced if lots of memory contention (including that from other communication)
- Memory copying is performed by the CPU
  - Cannot be usefully overlapped with work. For non blocking communications two possibilities:
    - Copies performed by a separate thread
    - Copies preformed by the main thread in send/recv/wait

Streams data



Adapter bottlenecks

- Usually nodes are connected to the switch infrastructure using "network adapters"
- There will be a limit to the rate that adapters can process data
  - especially if attached to a slow device bus like PCI
- Adapters are shared so may have contention
- There may be significant latency (cost independent of the packet size) for transferring a packet to/from an adapter
  - This will show up in MPI latency
  - May also in MPI bandwidth if MPI message uses multiple packets

Switch bottlenecks

- Communication performance may also be limited by the network switch

- As SMP clusters consist of a smaller number of more powerful nodes it can be easier to build switch networks for them
  - Can use smaller switch networks and run multiple networks in parallel to increase bandwidth

Bottlenecks

- Overall communication performance depends on all of the above factors

- Limits due to contention for a resource may not be visible if some other part of the system is more restrictive
  - E.g. IBM p690+ system was limited by the PCI bus, then by the switch microcode

Optimizing for SMP clusters

- There are several classes of optimization specific to SMP clusters:
  - Using threads within a MPI process
    - Considered elsewhere in this tutorial
  - Using explicit shared memory segments within a node
  - Optimizing the communication pattern to make best use of Inter/Intra node communication

Explicit shared memory segments

- You can use shared memory segments to create areas of memory accessible by all processes on a node
- Don't try to do this to send messages
  - MPI should already be doing this and will almost certainly do a better job
- However may be useful for operations that don't map well onto MPI messages
  - If all processors need read-only look-up tables they can share a single copy
  - If the program caches previously computed values the processes can share a single cache (values computed once per node instead of per process)

Optimizing communication pattern

- Inter/Intra node communications have different performance and contention characteristics
- We want to modify the communication pattern to take advantage of this. For example:
  - Introduce additional Intra-node communication to reduce the amount of data sent between nodes (reduce Inter-node bandwidth use)
  - Introduce additional Intra-node communication to reduce the number of messages sent between nodes (reduce Inter-node latencies)
  - Modify problem decomposition so less communication happens between nodes

Tree based communication

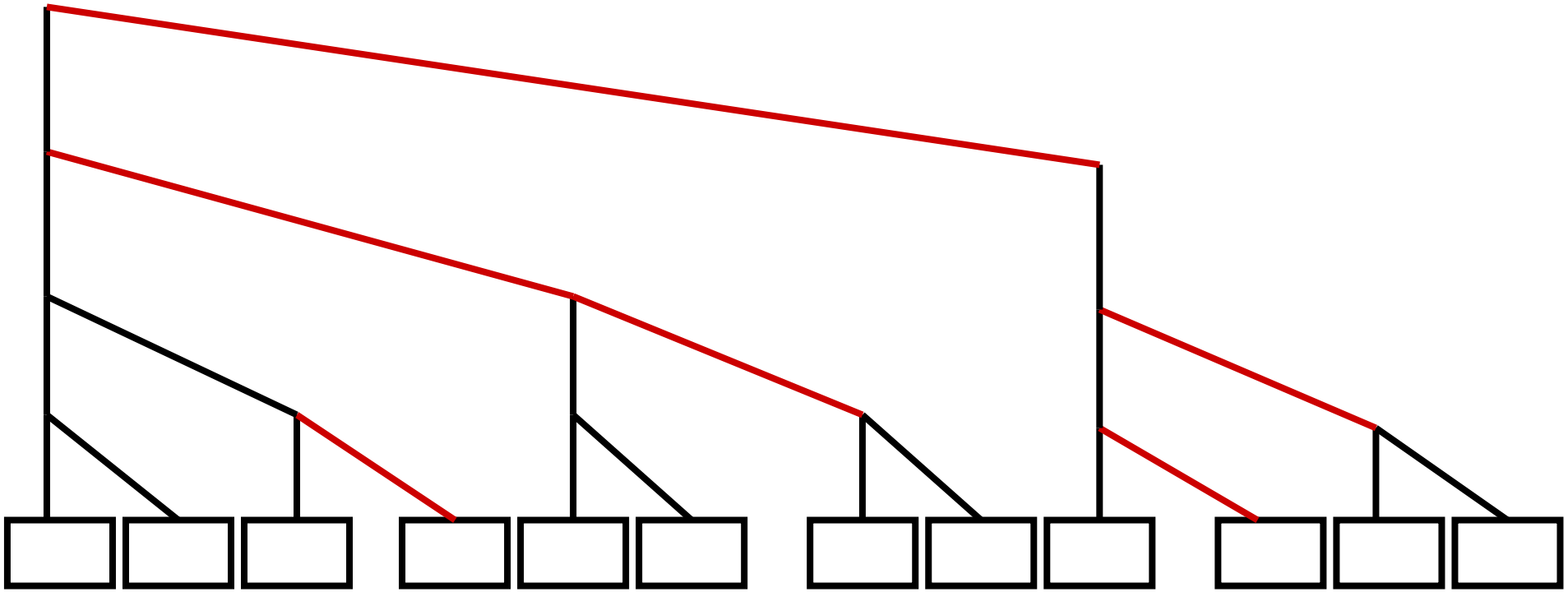
- A number of collective operations utilise communication up and down trees
  - Broadcast
  - Reduce
  - Barrier
- These communication patters can be optimized for SMP clusters
  - Hopefully your MPI library already has this optimization
  - If not you can use the MPI profiling interface to substitute an optimized version

Optimized trees

- When communicating up and down trees communication takes place in a series of steps
- Want to construct the tree so that the minimum number steps contain communications that cross node boundaries
- Essentially what we want is a tree of trees
  - Processes within a node form a tree
  - The root nodes of these trees are formed into a tree across nodes
- In MPI we need to split communicators

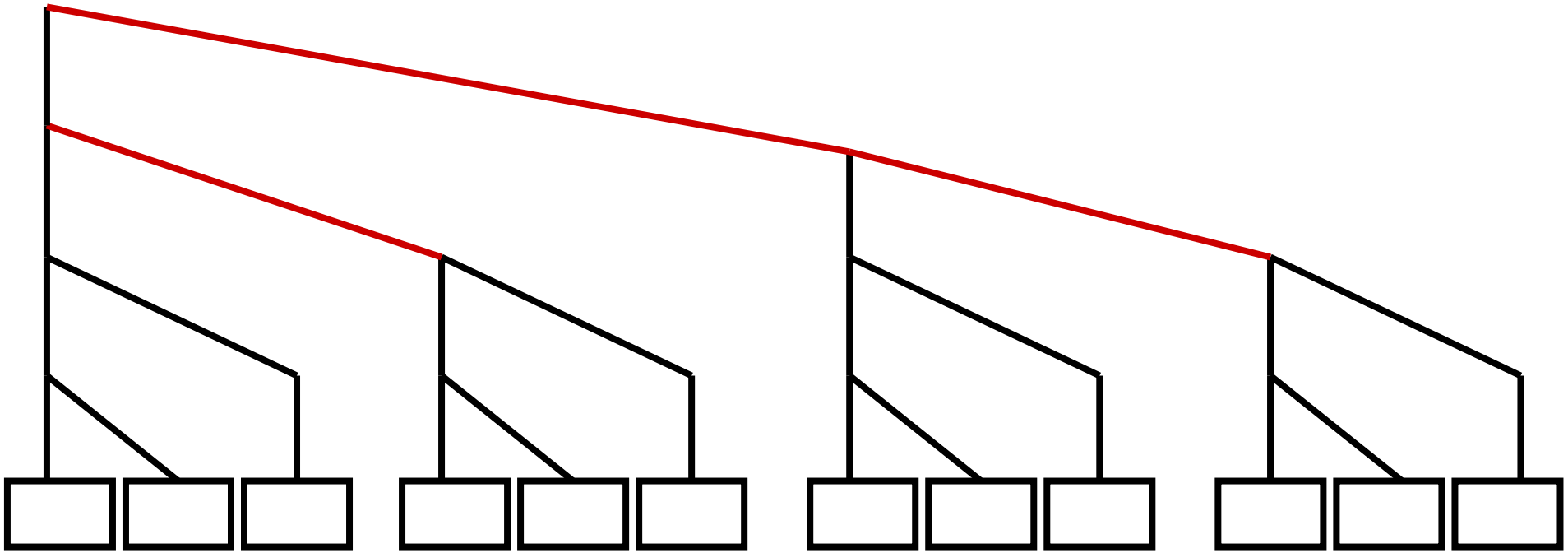
Unoptimized trees

- All steps have some off node communication



Optimized trees

- Only 2 steps contain off node communication



MPI process mapping

- Your MPI or system libraries may give you some control over where **processes** are allocated within a system
- SMP clusters require strategy for processor grouping
  - Grouping local processes on nodes
  - Grouping nodes sensibly
- Even if can't control need to know what happens
- Allows optimization of communication patterns
  - Can reduce the amount of off-node communication

Split communicators

- We want to split communicators on node boundaries
- **MPI_Get_processor_name** gives us the name of the current node
  - Generate a hash-value to turn this into a unique number e.g.

```
hash = 0
For each byte c in name
  hash -> 131*hash + c
```

  - Most systems use naming scheme where node names only differ by numerical digits

Split communicator II

- Use this number to split communicators

```
MPI_Get_processor_name(procname,&len);
node_key = name_to_colour(procname);
MPI_Comm_split(input,node_key,0,&local);
```

- Local is now a communicator for the local node
- Now we can make communicators across nodes

```
MPI_Comm_rank(local,&lrank);
MPI_Comm_split(input,lrank,0,&cross);
```

- This works even if input is not **COMM_WORLD** but must use cross communicator from local rank-0

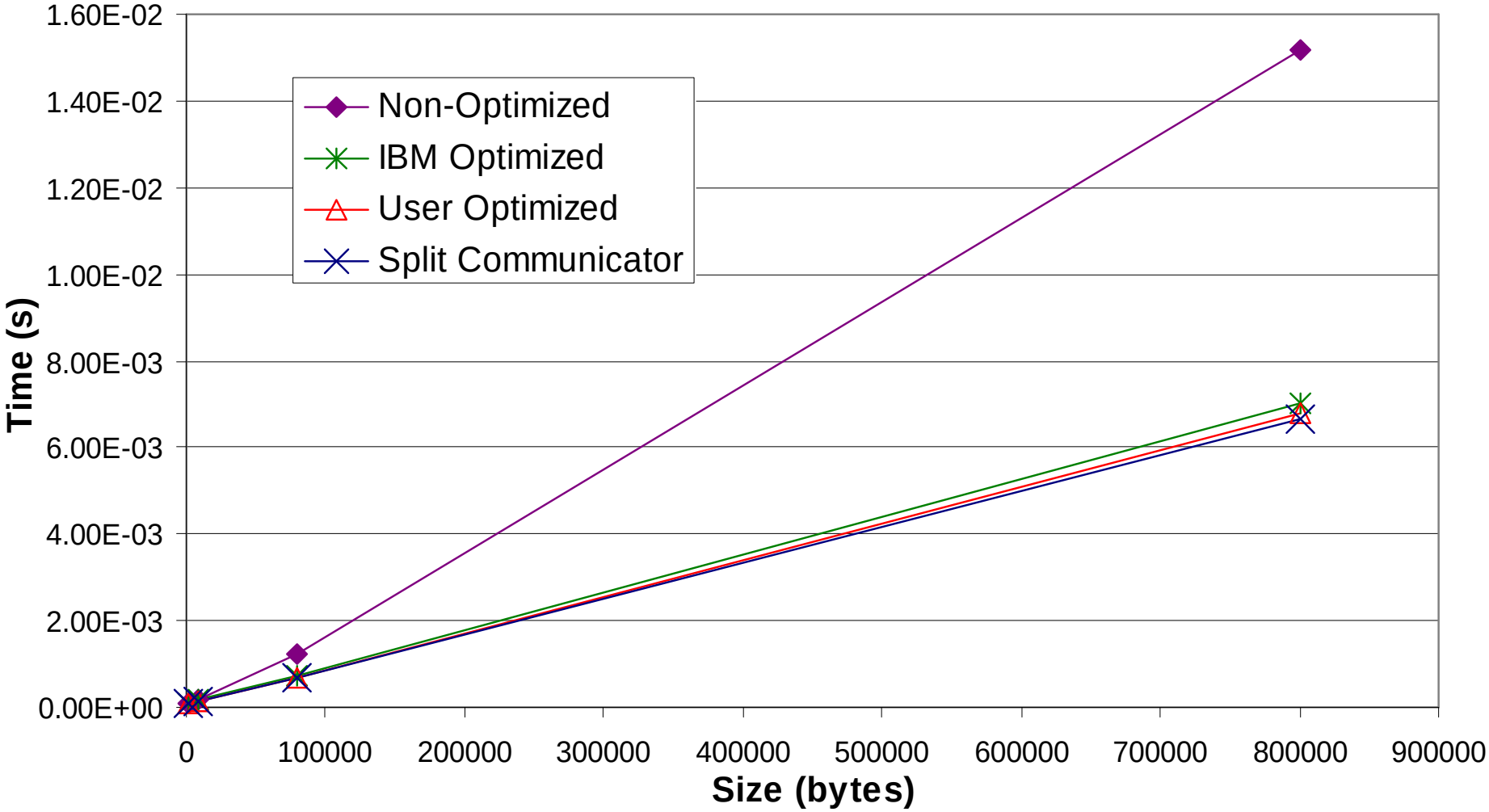
Optimized AllReduce

- Allreduce can then be implemented as follows:

```
MPI_Reduce(sendbuf,scratch,count,dt,op,0,local);
If(local_rank == 0)
 MPI_AllReduce(scratch,recvbuf,count,dt,op,cross);
MPI_Bcast(recvbuf,count,dt,0,local);
```

- Also possible to optimize for cases where local or cross communicators only contain one processor
- MPI collectives should be optimized by vendor but you may need to do something similar in application code

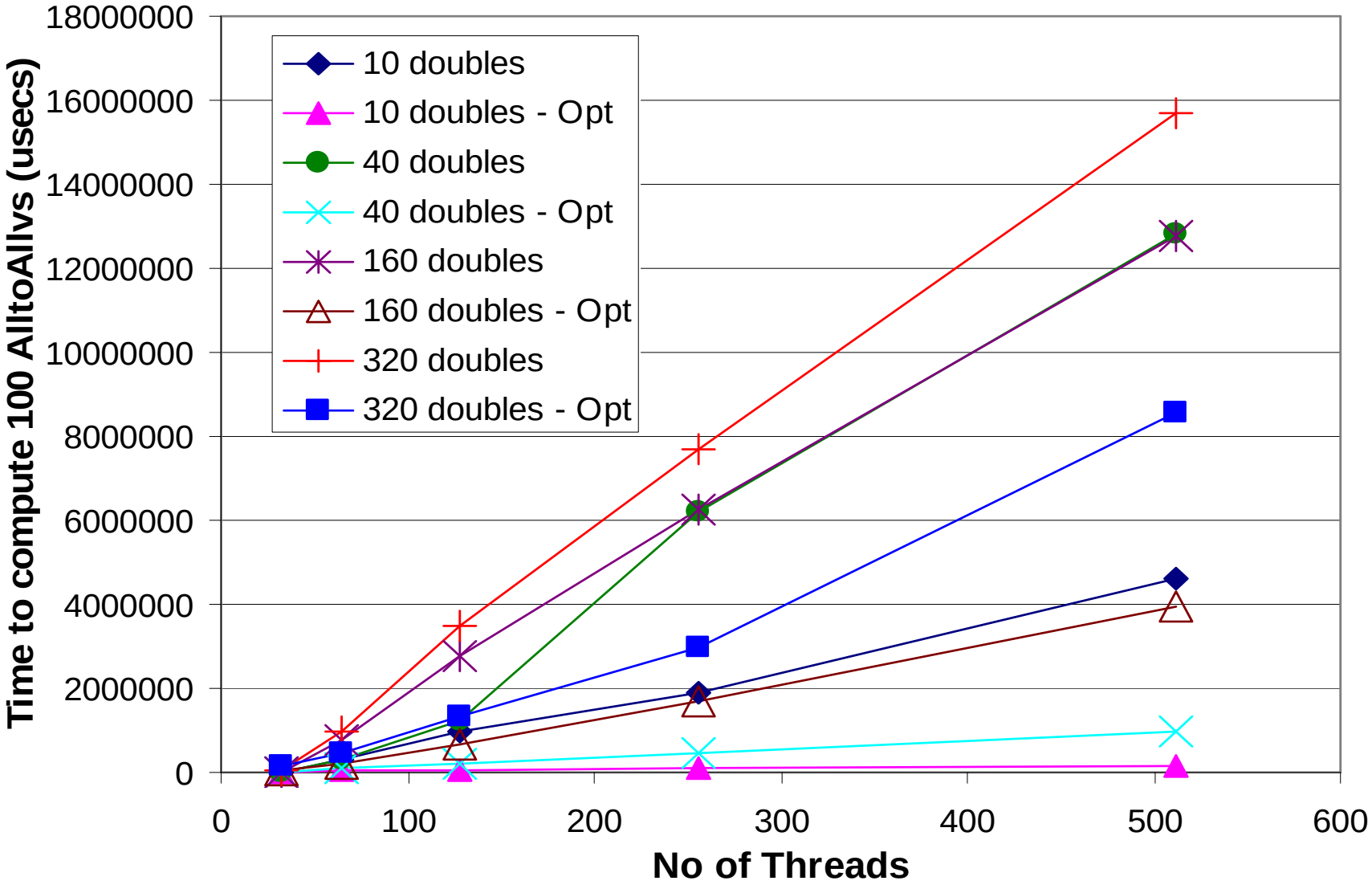
Optimized AllReduce



Optimizing AlltoAll

- The **`MPI_Alltoall`** call is equivalent to every process sending a message to every other process
- This involves a very large number of messages
- Can reduce the impact of inter-node latencies by having each SMP node only send one (larger) message to each other SMP node
- This requires additional local communication within a node to collect/distribute the data before/after
- May still be worthwhile depending on the Inter-node latency cost

Optimized AlltoAllv



Remove order constraints

- Many applications have communication phases made up from several different messaging operations
  - For example boundary swap
- In general it is good practice not to introduce unnecessary constraints on the order of communication
  - Implement all boundary swaps together using non-blocking calls rather than E-W followed by N-S
- This is especially important for SMP clusters where communication speeds vary

Order constraint example

- Consider a 2D regular decomposition
  - N-S dimension decomposed within a SMP node
  - E-W dimension decomposed across nodes

- The N-S boundary swap only uses shared memory and can probably overlap with the E-W swap without contention
- If code is written to force E-W swap to complete first overlap cannot happen and runtime will be increased

Conclusions

- The main difference between SMP clusters and uni-processor clusters is the degree that hardware is shared by CPUs
  - This decreases cost and complexity but increases the risk of resource contention
- SMP clusters do not have uniform communication performance
  - Inter and Intra node communications behave differently
  - To gain the maximum performance codes may need to be modified to take account of this difference

References

- Stream Benchmark. STREAM: Sustainable Memory Bandwidth in High Performance Computers, *J. McCalpin*. See:

  http://www.cs.virginia.edu/stream/

- Planned AlltoAllv: A Cluster Approach, *A. Jackson and S. Booth*, HPCx Technical Report 2004. See:

  http://www.hpcx.ac.uk/research/hpc/

- Cluster aware collective MPI communications, S. Booth. Source code available from:

  http://www.epcc.ed.ac.uk/~spb/shm_mpi/fast_mpi.c