

---

# Decomposition Strategies

.



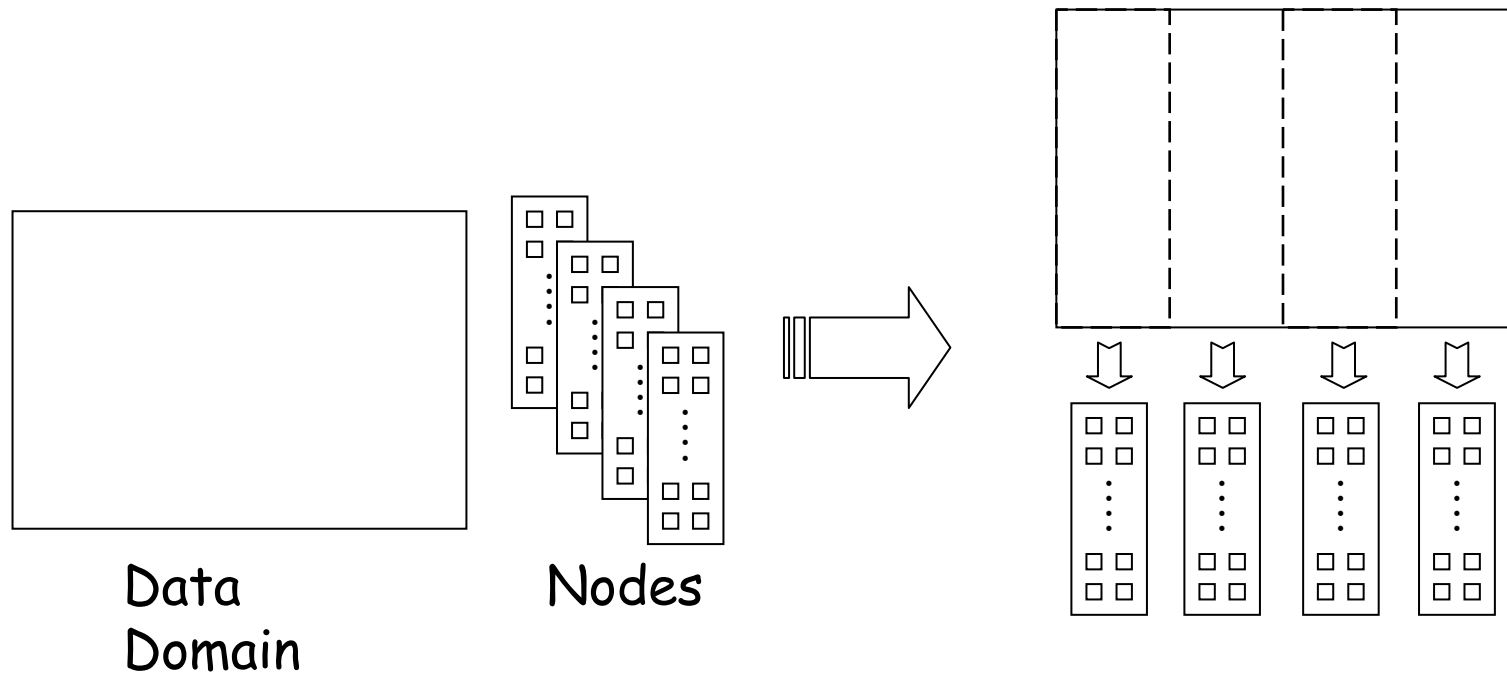
|epcc|

- Decomposition patterns
  - Domain decomposition
  - One Stage and two stage optimization
  - Irregular decompositions
- Data replication
- Underpopulation
- I/O
  - SMP cluster optimized
  - Asynchronous
  - MPI-IO

- Parallel codes require data to be distributed
  - Regular or irregular data distribution
  - Data replication
- Decomposition of data across processors impacts performance
  - Data distribution affects communication patterns
  - Looking to maximise computation to communication ratio
- SMP clusters introduce level of non-uniformity to the decomposition problem
- Decomposition maybe dependent on the data set

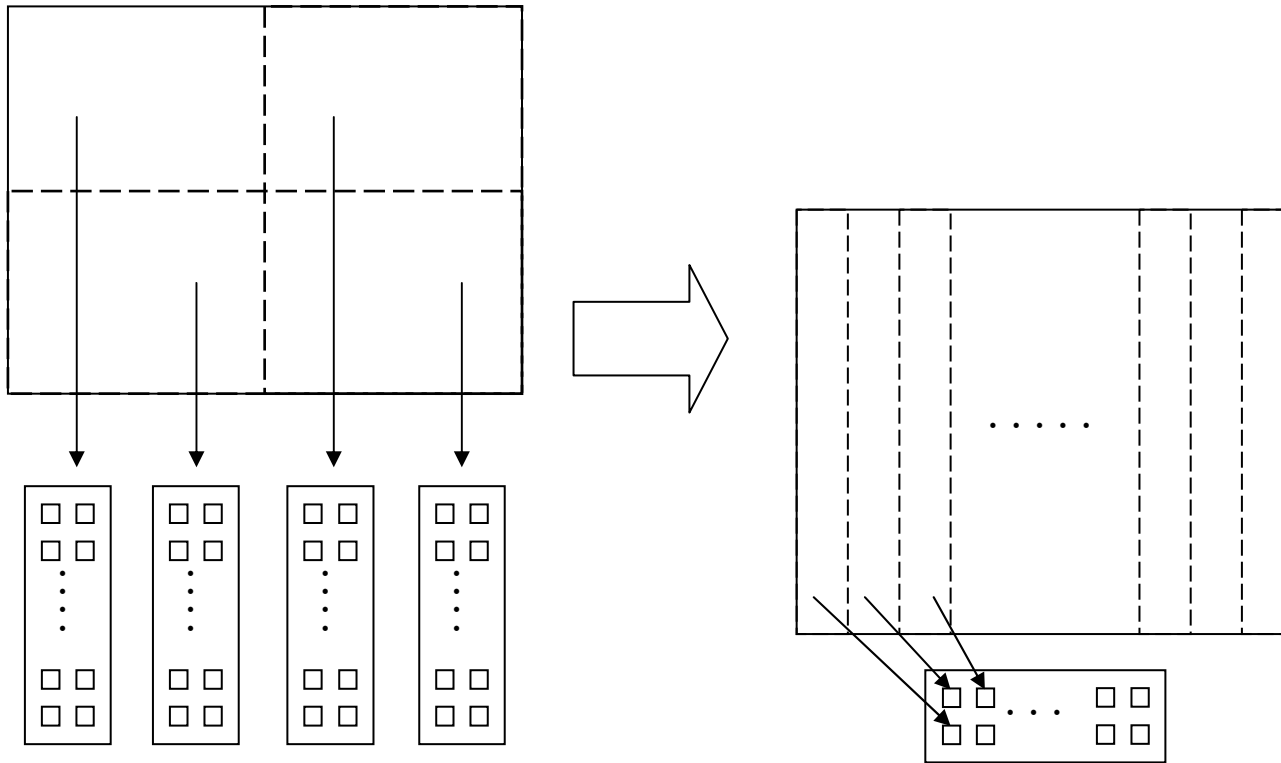
- **Mapping N dimensional dataset over processors**
  - Must match processor groups (nodes) to sections of the data
  - Exploit fast "on-node" communications
    - Reducing impact of message latency
- **Simplest optimization is to map dimensions to nodes**
  - Removes a direction of "off-node" communications for each dimension mapped
  - 2D dataset: decompose one dimension across nodes, and fit one into the nodes
  - 3D dataset: match 2 dimensions to the node size, removing two directions of communications
- **3D decomposition slicing**
  - Allows more regular decompositions

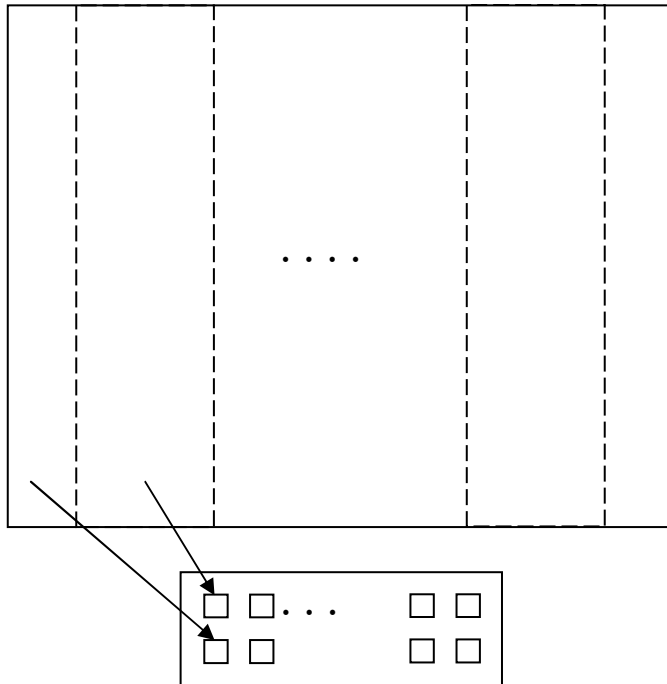
- Performance benefit dependent on communication pattern
  - Regular boundary swapping communications ideal
- May not be practical for many datasets
  - "Square" datasets could produce decompositions with poor surface area to volume ratios
- Depends on node sizes and number of nodes used



- Require a strategy for data decomposition within a node
- Simplest mechanism is to use a 1D decomposition within a node
  - Simple to program and understand
- Regular communication pattern can cause load-balance problems using simple strategy
  - Communications non-uniform latency and bandwidth

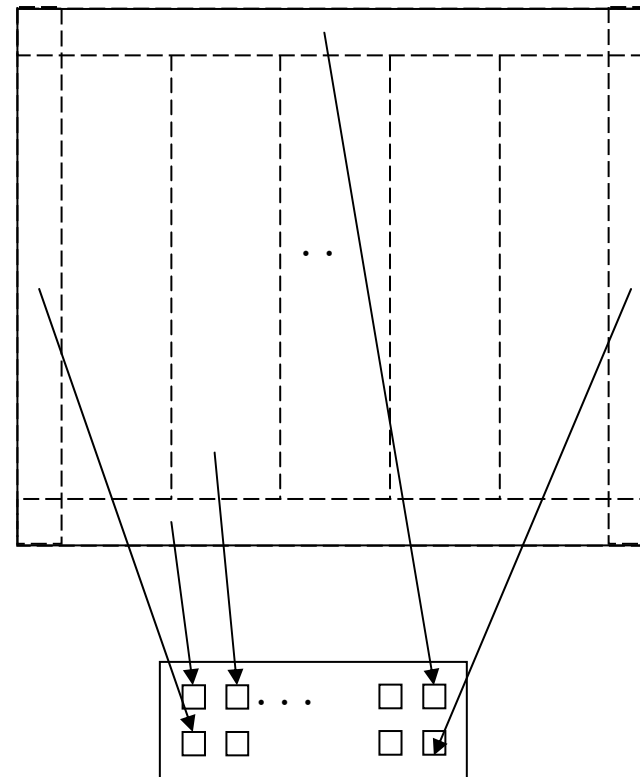
# Two stage decomposition

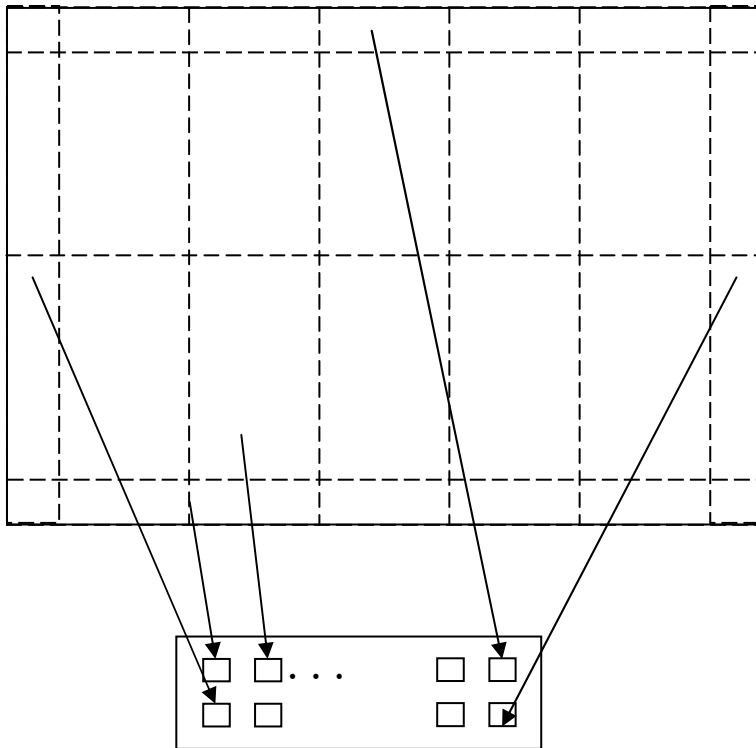




- Decompose data over nodes in a balanced fashion
- Use a non-uniform pattern within a node to ensure load balance
- 1D decomposition can be altered to take into account communication load

- Further optimization - isolate the off-node communication load to the fewest possible processors
- Using a 2D decomposition, data that relies on off-node processes can be allocated to one processor
- "Internal" processes have a larger computation load





- More practical but greater latency
- Overlapping communications necessary to gain benefit
  - Non-blocking MPI: IRecv, ISend, Wait

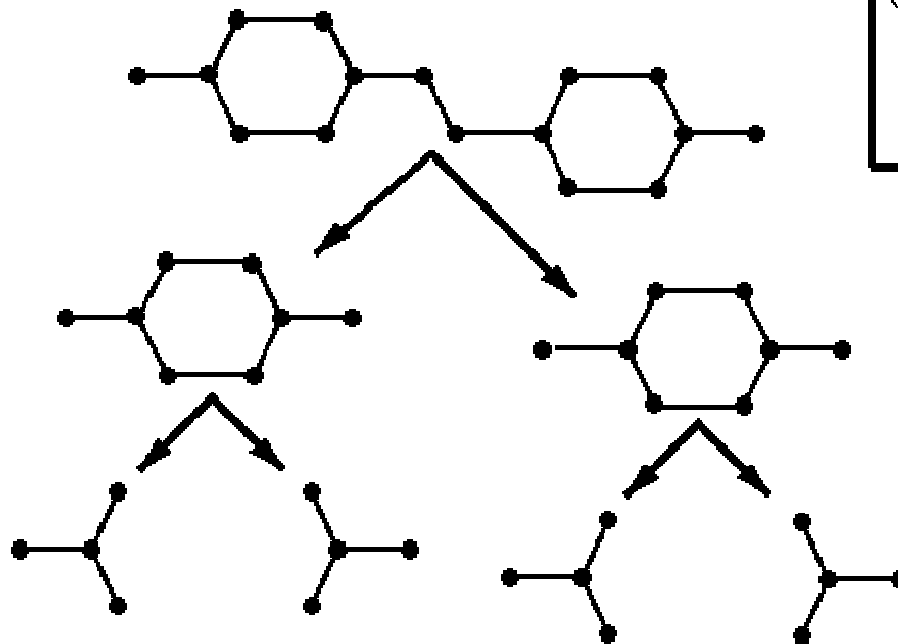
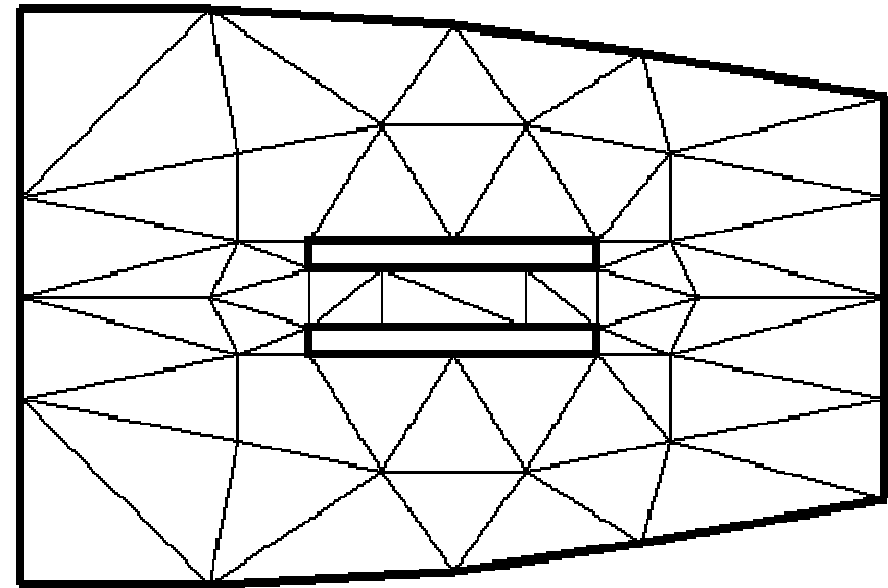
- Same algorithm with different datasets may need different decompositions
  - May need to experiment to get optimum decomposition
  - Self-tuning may be an a helpful tool
- Interaction between across-node decomposition and on-node decomposition

- May optimize performance where data is stored locally or resources attached locally
- Good for irregular decomposition
- By default many systems number linearly
  - This means that communication patterns may not be optimized
- By creating a cartesian communicator we may achieve an optimized communication pattern
  - By setting `Reorder=True`
  - However some systems may not do something sensible

- Graph-optimization based decomposition can be optimized for SMP clusters
- Mapping elements to nodes is key
- Basic decomposition involves computing a graph that minimises the number of links between elements to reduce computation
- Also attempting to load-balance the number of elements per processor

- For SMP clusters, the graph optimization algorithm should take into account two levels of links
  - Slow "off-node" links, and fast "on-node" links
  - Optimize to reduce number of slow links between large groups of elements connected by fast links
- This allows for off-node and on-node communication to be load-balance again each other
- Must also distributing work within nodes

- Simplest way to optimize for cluster is to perform the decomposition twice



- Run same algorithm in two stages to give:
  - 1<sup>st</sup> Global decomposition
  - 2<sup>nd</sup> Local decomposition

- Replicated data problems can also be optimized for clusters:
  - One copy of data shared per node
- Dramatically reduces memory used
  - For 32 processor nodes, consuming a  $1/32^{\text{th}}$  of memory
  - Can allow much larger problems to be computed
- Can be costly to computation
  - Memory bandwidth to shared array can restrict performance
  - Maybe synchronisation issues when updating shared memory

- Performance improvement can be achieved with under populated nodes
  - E.g. for the AMBER\* code on an IBM p690:

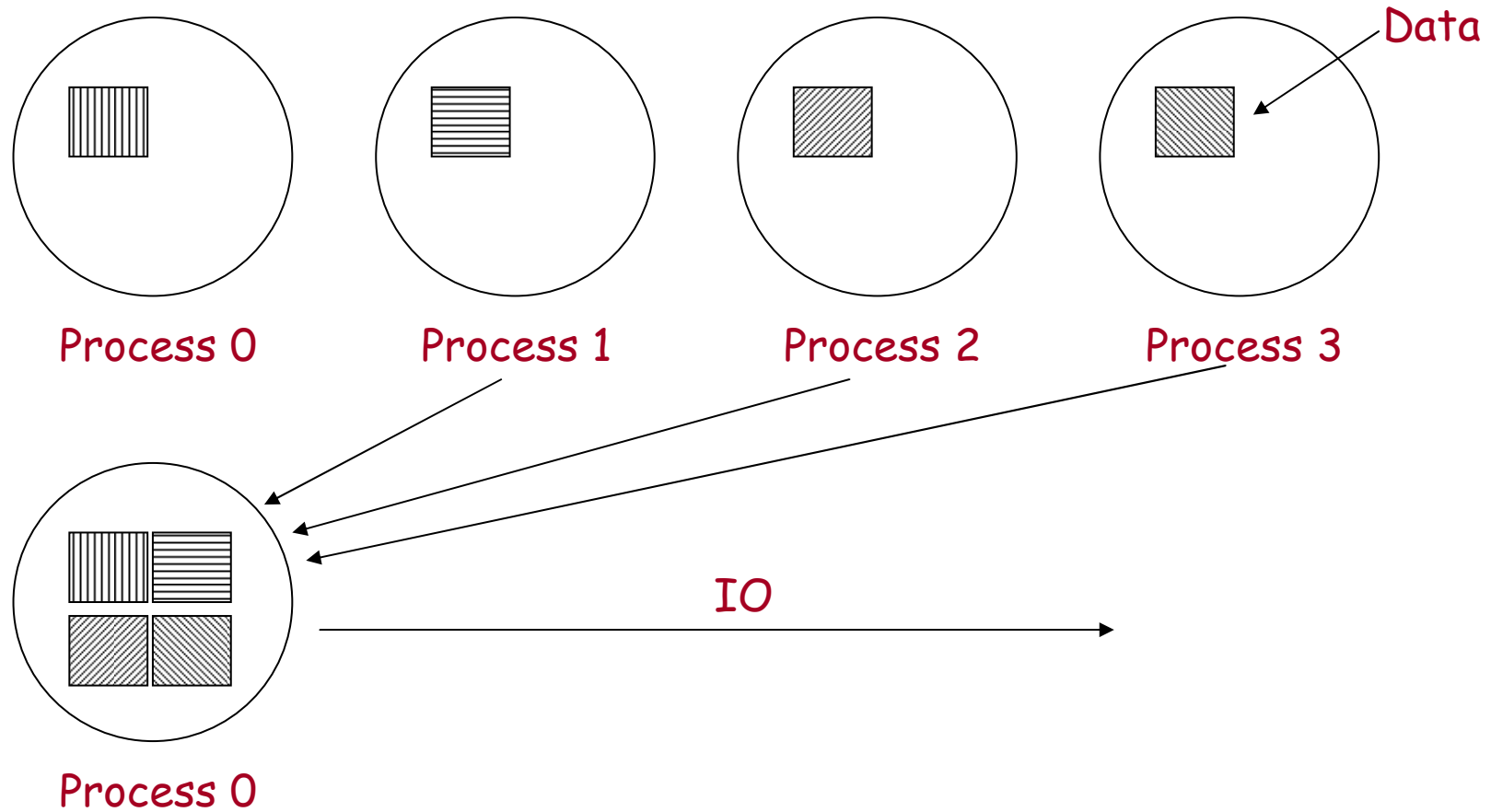
8 procs (1 node, 8 procs per node)	247s
8 procs (2 nodes, 4 procs per node)	259s
16 procs (2 nodes, 8 procs per node)	191s
16 procs (4 nodes, 4 procs per node)	170s
32 procs (4 nodes, 8 procs per node)	171s
32 procs (8 nodes, 4 procs per node)	133s
64 procs (8 nodes, 8 procs per node)	162s
64 procs (16 nodes, 4 procs per node)	139s

\*AMBER See: <http://amber.scripps.edu/>

- For some parallel codes
  - IO can be a time consuming bottleneck
- IO installations on SMP clusters vary widely
  - Highly customised
  - IO system on SMP clusters can be SMP clusters themselves
- Speed vs Usability and Portability
- Reading and Writing may need different strategies

- Two main I/O setups
  - Global File System
  - Local File System
- Key difference is the level at which communication occur
  - O/S level -> Global File System
  - Software level -> Local File System
- Performance depends on Interconnect
  - Special purpose: GPFS,
  - General: NFS, Ethernet, ...
- Usability and scalability dependent on configuration

- Single processor master I/O is a common approach

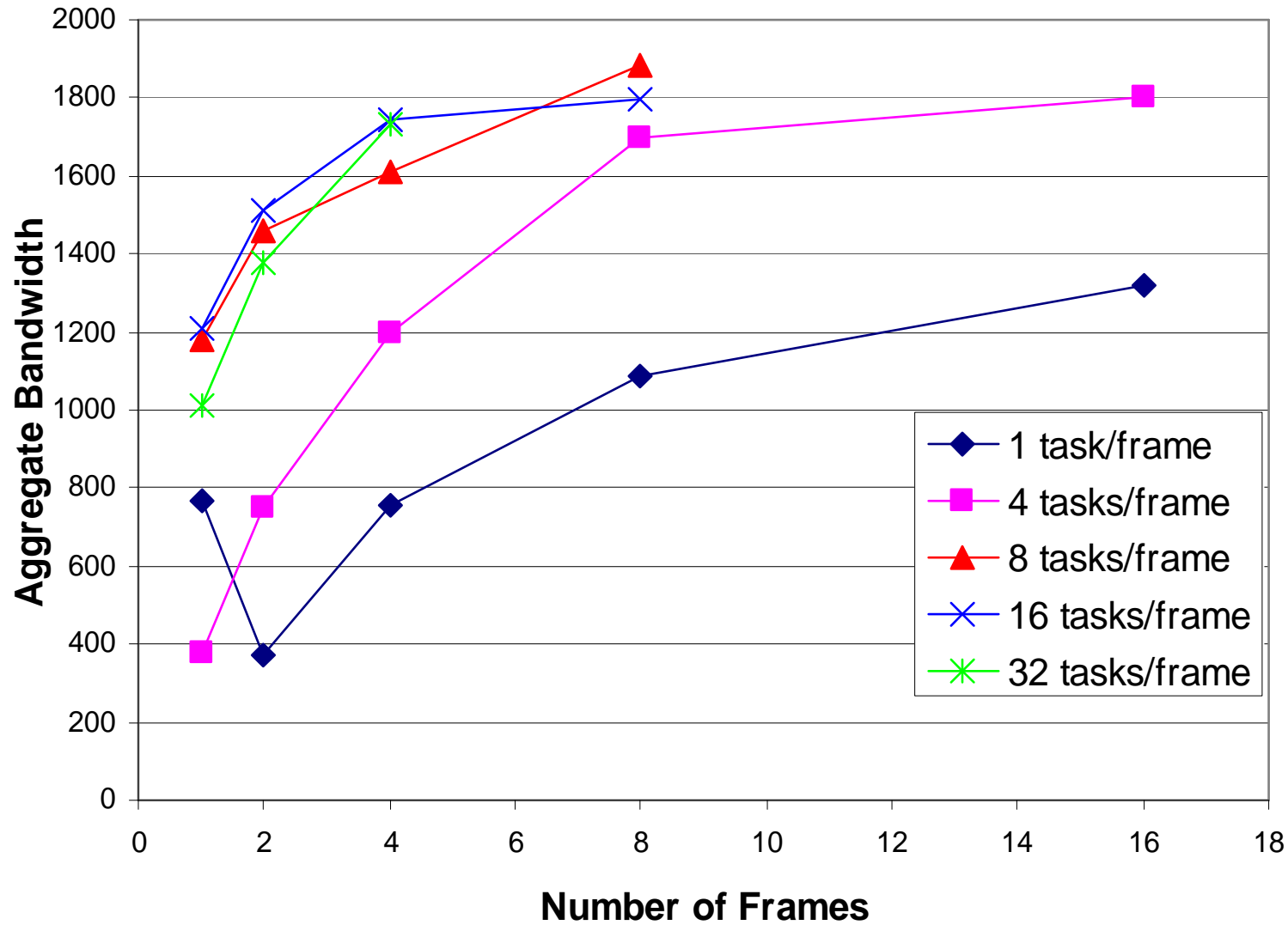


- Single processor I/O master approach can severely limit large processor jobs
  - Involves two stage communication process
  - Heavy use of the network
  - Master node is doing a large amount of work - load imbalance
  - Master node may run out of memory

- Each processor can write to its own file, removing one communication stage
  - Local I/O caching, less contention for switch and memory bandwidth
  - Pre/post-processing of files required
  - Checkpointing and restarting on a different number of processors may not work

- An optimized clustered Master I/O approach is possible
- One master process per node gathers /scatters data and performs the I/O
  - Reduced synchronisation as only "fast" communications used
  - Improved usability as only a small number of files needed
- Gather/scatter performed either using MPI communications or a shared-memory segment
- May need to use a small group of processors per node to get improved performance

# Clustered master I/O



- Each process reads and writes to the same file
    - Use "random access" file to write or read from different positions in one file
    - Removes large number of files, improves usability
  - Can complicate programming and dependent on hardware setup
    - Less portable than using separate files
    - May not be able to make full use of I/O system performance
  - Requires knowledge of number of processor and data decomposition used
    - Robust code should handle different datasets
-

- Writing or reading large amounts of data can be optimized using Asynchronous I/O (i.e. non-blocking)
  - When the I/O function is called, the function finishes immediately, allowing the program to continue
- Asynchronous I/O may still impact computation / communication performance as it's using system resources
  - May spawn threads that will cause context switch
- Again, dependent on hardware setup and OS implementation
  - Reduces portability

- MPI library provides Parallel I/O functionality (MPI-2)
  - Simple, portable access to files
  - Portability depends on MPI implementation
- Allows the reading of a single file without worrying about how many processors created it
  - Achieve decomposition-independent file formats for 2 and 3D data or processor distributions
- Greater portability than machine specific I/O
- Greater performance than master serial I/O
- Often optimized for large datasets and file sizes, and large numbers of processors

- Data decomposition and communication patterns can significantly impact performance for large numbers of processors
  - Aim to reduce synchronisation costs and maximise load-balance
- Any decomposition pattern must take the different communication performance into account
  - Two levels of communication cost
  - Non-blocking communications essential to mask latency
- I/O load can be reduced depending upon your requirements

- *AMBER Performance on HPCx, L. Smith, C. Johnson, HPCx Technical Report, 2003. See:*  
<http://www.hpcx.ac.uk/research/hpc/>
- *File I/O from multi processor jobs, Joachim Hein, HPCx Technical Report, 2003. See:*  
<http://www.hpcx.ac.uk/research/hpc/>