
Mixed mode programming



- SMP clusters can generally be adequately programmed using just MPI
- It is also possible to write mixed MPI/OpenMP code
 - Seems to be the best match of programming model to hardware
 - What are the (dis)advantages of using this model?

- **Master only**
 - All communication is done by the OpenMP master thread, outside of parallel regions
 - Other threads are idle during communication
- **Funnelled**
 - All communication is done by one OpenMP thread, but this may occur inside parallel regions
 - Other threads may be computing during communication
- **Multiple**
 - Several threads (maybe all) communicate

We need to consider:

- Development / maintenance costs
- Portability
- Performance

- In most cases, development and maintenance will be harder than for an MPI code, and much harder than for an OpenMP code
- If MPI code already exists, addition of OpenMP may not be *too* much overhead
 - Easier if parallelism is nested
 - Can use master-only style, but this may not deliver performance: see later
- In some cases, it may be possible to use a simpler MPI implementation because the need for scalability is reduced
 - E.g. 1-D domain decomposition instead of 2-D

- Both OpenMP and MPI are themselves highly portable (but not perfect)
- Combined MPI/OpenMP is less so
 - Main issue is thread safety of MPI
 - If full thread safety is assumed (multiple style), portability will be reduced
 - Batch environments have varying amounts of support for mixed mode codes
- Desirable to make sure code functions correctly (with conditional compilation) as stand-alone MPI code and as stand-alone OpenMP code

Possible reasons for using mixed MPI/OpenMP codes on SMP clusters:

1. Intra-node MPI overheads
2. Contention for network
3. Poor MPI implementation
4. Poorly scaling MPI codes
5. Replicated data

- Simple argument:
 - Use of OpenMP within a node avoids overheads associated with calling the MPI library
 - Therefore a mixed OpenMP/MPI implementation will outperform a pure MPI version

- **Complicating factors:**

- The OpenMP implementation may introduce additional overheads not present in the MPI code
 - e.g. synchronisation, false sharing, sequential sections
- The mixed implementation may require more synchronisation than a pure OpenMP version
 - especially if non-thread-safety of MPI is assumed
- Implicit point-to-point synchronisation may be replaced by (more expensive) OpenMP barriers
- In the pure MPI code, the intra-node messages will often be naturally overlapped with inter-node messages
 - Harder to overlap inter-thread communication with inter-node messages

Example

```
!$omp parallel do
```

```
DO I = 1,N
```

```
    A(I) = B(I) + C(I)
```

```
END DO
```

Implicit OpenMP barrier
added here

Single MPI task may not use
all network bandwidth

other threads idle while
master does MPI calls

```
CALL MPI_BSEND(A(N),1,.....)
```

```
CALL MPI_RECV(A(0),1,.....)
```

cache miss to access
message data

```
!$omp parallel do
```

```
DO I = 1,N
```

```
    D(I) = A(I-1) + A(I)
```

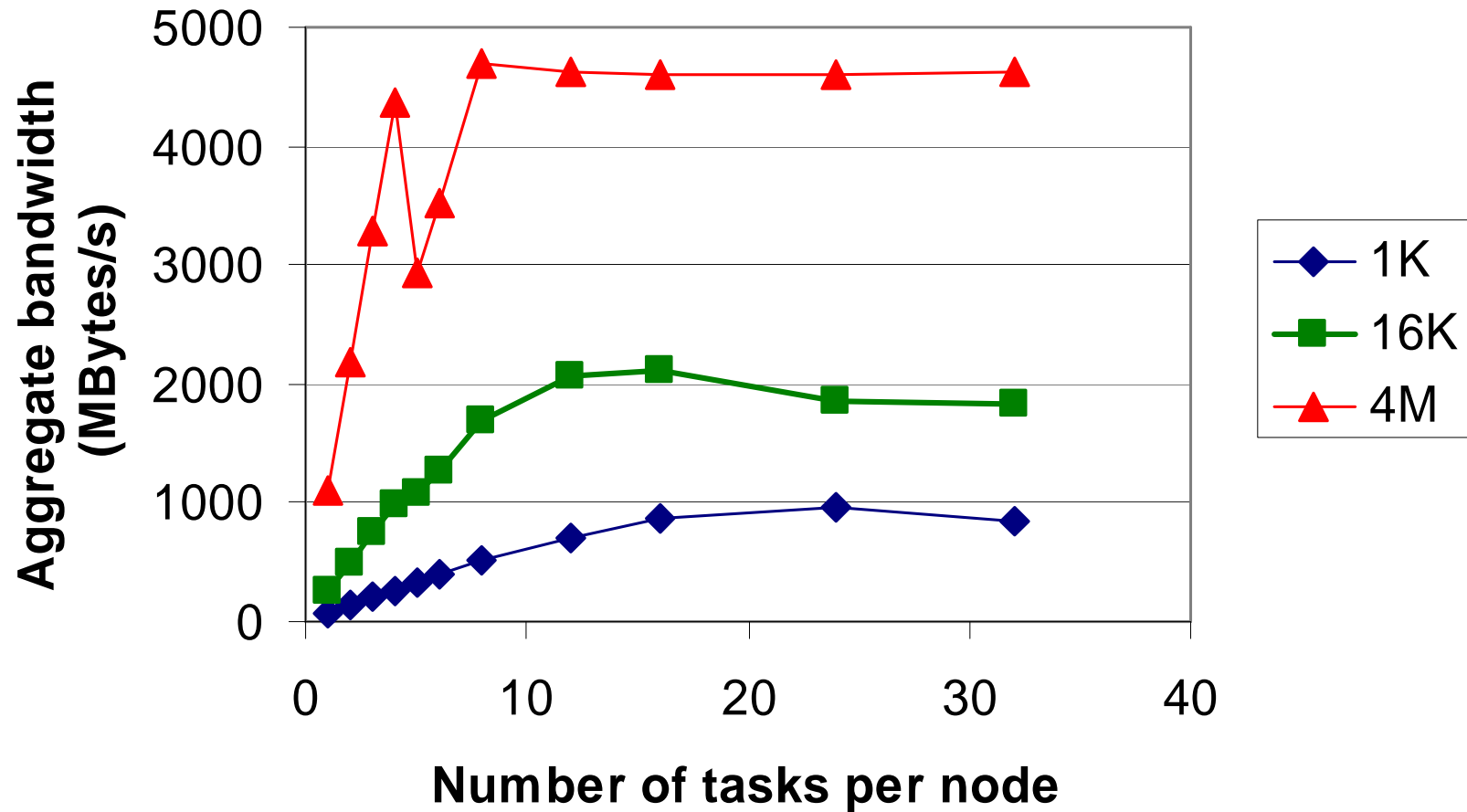
```
END DO
```

cache miss to access data
written by other threads

- Master-only style of mixed coding introduces significant overheads
 - often outweighs benefits
- Can use funnelled or multiple styles to overcome this
 - typically much harder to develop and maintain
 - load balancing compute/communicate threads in funnelled style
 - mapping both processes and threads to a topology in multiple style

- On a node with p processors, we will often have the situation where all p MPI processes (in a pure MPI code) will send a message off node at the same time
- This may cause contention for network ports (or other hardware resource)
- *Maybe* better to send a single message which is p times the length
- On the other hand, a single MPI task may not be able to utilise all the network bandwidth

Aggregate bandwidth



- If the MPI implementation is not cluster-aware, then a mixed-mode code may have some advantages
- A good implementation of collective communications should minimise inter-node messages
 - e.g. do reduction within nodes, then across nodes
- A mixed-mode code would achieve this naturally
 - e.g. OpenMP reduction within node, MPI reduction across nodes

- Only a subset of collective communications are optimised
- In some cases, using split communicators, shared memory segments or mixed-mode can improve performance

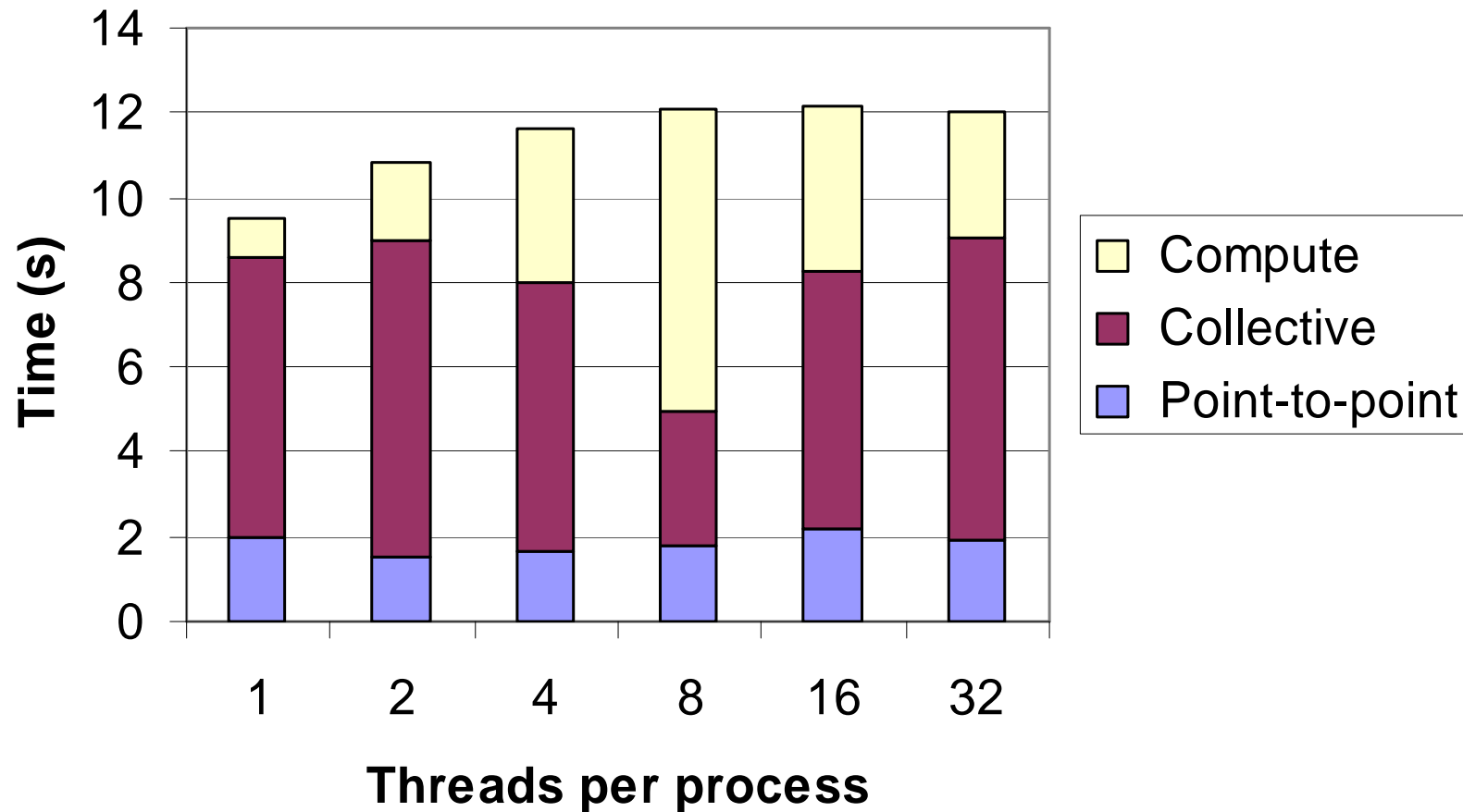
- If the MPI version of the code scales poorly, then a mixed MPI/OpenMP version *may* scale better
- May be true in cases where OpenMP scales better than MPI due to:
 1. Algorithmic reasons
 - e.g. adaptive/irregular problems where load balancing in MPI is difficult
 2. Simplicity reasons
 - e.g. 1-D domain decomposition
 3. Reduction in communication
 - often only occurs if dimensionality of communication pattern is reduced

- Most likely to be successful on fat node clusters (few MPI processes)
- Load balance with threads, how much trade off, batch system, limited flexibility

- Some MPI codes use a replicated data strategy
 - all processes have a copy of a major data structure
- A pure MPI code needs one copy per process(or)
- A mixed code would only require one copy per node
 - data structure can be shared by multiple threads within a process
- Can use mixed code to increase the amount of memory available per task

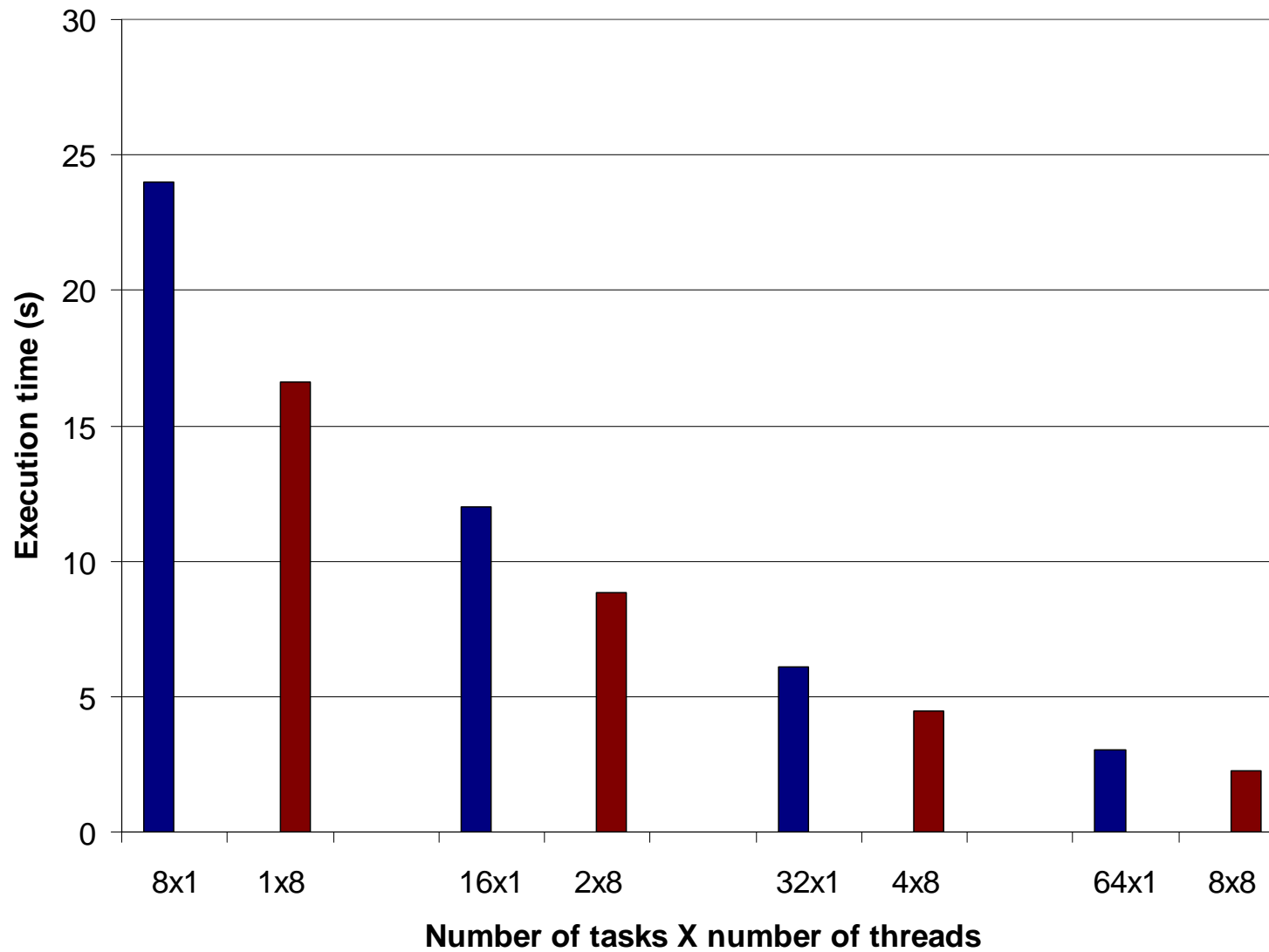
- Simple Jacobi kernel
 - 2-D domain , halo exchanges and global reduction
- ASCI Purple benchmarks
 - sPPM
 - gas dynamics on 3-D regular domain

Simple Jacobi kernel



- Results are for 128 processors (4 nodes)
- Mixed mode is slower than MPI
- Collective communication cost reduced with more threads
- But: offset by increased computation cost
 - Includes reduction operation in OpenMP

- **Parallelism is essentially at one level**
 - MPI decomposition and OpenMP parallel loops both over physical domain
- **Funnelled style**
 - Overlapped communication and calculation with dynamic load balancing
- **Mixed mode is significantly faster**
 - Main gains appear to be reduction in inter-node communication
 - In some places, avoids MPI communication in one of the three dimensions



- Don't rush: you need to argue a very good case for a mixed-mode implementation
- If MPI scales better than OpenMP within a node, you are unlikely to see a benefit
 - Requirement for large memory is an exception
- The simple "master-only" style is unlikely to work
- It may be better to consider making your algorithm/MPI implementation cluster aware (e.g. use nested communicators)

- Clearly not the most effective mechanism for all codes
- Significant benefit may be obtained in certain situations:
 - MPI code doesn't scale well
 - replicated data codes
- Unlikely to benefit well optimised existing MPI codes
- Portability and development / maintenance considerations

- *Mixed mode programming on a clustered p690 system, L. Smith, M. Bull, J. Duthie, Scicomp 9, 2004. See:*
<http://www.spscicomp.org/ScicomP9/program.html>
- *Development of mixed mode MPI / OpenMP applications, L. Smith and M. Bull, Scientific Programming, Vol. 9, No 2-3, 2001, 83-98.*