Improved Performance
Scaling on HPCx

Mixed mode programming
Motivation

• HPCx, like many current high-end systems, is a cluster of SMP nodes

• Such systems 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?
Styles of mixed code

• **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
Issues

We need to consider:

• Development / maintenance costs
• Portability
• Performance
Development / maintenance

- 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
Portability

• 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 HPCx:

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

• 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.
Intra-node MPI overheads

• 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

```fortran
!$omp parallel do
  DO I = 1,N
    A(I) = B(I) + C(I)
  END DO

CALL MPI_BSEND(A(N),1,.....)
CALL MPI_RECV(A(0),1,.....)

!$omp parallel do
  DO I = 1,N
    D(I) = A(I-1) + A(I)
  ENDDO
```

- **Implicit OpenMP barrier added here**
- **Single MPI task may not use all network bandwidth**
- **Other threads idle while master does MPI calls**
- **Cache miss to access message data**
- **Cache miss to access data written by other threads**
Mixed mode styles again...

- 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
Competition for network

- 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)
- *May* be 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
  - on HPCX this is the dominant effect
• On Phase 2 machine (HPS switch), off-node bandwidth depends on the number of MPI tasks.
  - a single message cannot use more than one adapter

• Test: ping-pong between 2 nodes.
  - vary the number of task pairs from 1 to 32.
  - measure aggregate bandwidth for medium size messages (16 Kbytes)
Communication bandwidth – IBM p690+

Aggregate Bandwidth (MB/s) vs. Number of pairs

Graph showing the aggregate bandwidth in MB/s for different numbers of pairs.

- IBM, 1K
- IBM, 16K
- IBM, 1024K

Legend:
- IBM, 1K
- IBM, 16K
- IBM, 1024K

Y-axis: Aggregate Bandwidth (MB/s)
X-axis: Number of pairs
Poor MPI implementation

- 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.
Optimising collectives

- MPI on Phase 2 system is cluster aware

- Only a subset of collective communications are optimised, and only in 64-bit mode.

- In some cases, using split communicators, shared memory segments or mixed-mode can improve performance
Poorly scaling MPI codes

• 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)

• May be more attractive on Phase 2 system
  - 32 processors per node instead of 8
Replicated data

- 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.
• On HPCx, the amount of memory available to a task is ~29Gb divided by the number of tasks per node.
  - for large memory jobs, can request less than 32 tasks per node
  - since charging is by node usage this is rather expensive
  - mixed mode codes can make some use of the spare processors, even if they are not particularly efficient.
Some cases studies

- Simple Jacobi kernel
  - 2-D domain, halo exchanges and global reduction

- ASCI Purple benchmarks
  - UMT2K
    - photon transport on 3-D unstructured mesh
  - sPPM
    - gas dynamics on 3-D regular domain
Simple Jacobi kernel

Time (s)

Threads per process

- Compute
- Collective
- Point-to-point
• 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
UMT2K

• Nested parallelism
  - OpenMP parallelism at lower level than MPI

• Master-only style
  - Implemented with a single OpenMP parallel for directive.

• Mixed mode is consistently slower
  - OpenMP doesn’t scale well
  - Results from Phase 1 system
UMT2K performance

Number of tasks X number of threads

Execution time

- 16x1
- 2x8
- 32x1
- 4x8
- 64x1
- 8x8
• 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
What should you do?

• 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)
Conclusions

- 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.