Mixed Mode Programming: A Case Study on an IBM Power5 Cluster

Mark Bull
Michał Piotrowski

EPCC, University of Edinburgh
Email: m.bull@epcc.ed.ac.uk, mpiotrow@epcc.ed.ac.uk
Motivation

• Clusters of shared memory nodes have become a system of choice for many research and enterprise projects.

• The mixed mode programming is a combination of shared and distributed programming models.

• It can potentially exploit features of the SMP cluster architecture, resulting in a more efficient parallelisation strategy.

Fig 1. SMP cluster model.
Mixed Mode

Mixed mode naturally matches the SMP cluster architecture.

Message exchange within a node is reduced and replaced by faster direct reads and writes from memory.

Typically, the number of inter-node messages will be reduced, but the average message size will increase.
Benchmark Code

Benchmark codes are based on the 3D Jacobi Relaxation Algorithm.

Benchmark code involves:
- 3D data decomposition
- Jacobi computation
- Point-to-Point communication (nearest neighbours)
- Global reduction (Delta)

while $\Delta$ is greater than tolerance

exchange halos

perform Jacobi relaxation

calculate global $\Delta$

end while
Master Only: all MPI communication takes place outside of OpenMP parallel regions.

while $\Delta$ is greater than tolerance
  
  if my thread id == 0 then
    exchange halos
  end if

  perform Jacobi relaxation
  
  OpenMP barrier

  calculate global $\Delta$

  update old array

end while
Funnelled: communication may occur inside parallel regions, but is restricted to a single thread.

distribute data between threads (load balance)

while $\Delta$ is greater than tolerance

if my thread id == 0 then

exchange halos

perform Jacobi relaxation

else

perform Jacobi relaxation (non-boundary space)

end if

OpenMP barrier

calculate global $\Delta$

update old array

end while
Multiple: all threads participate in the communication, independently of other threads

- distribute data between threads
- while is greater than tolerance
  - exchange 'top', 'bottom', 'front' and 'back' halos
  - if my thread id == 0 then
    - exchange 'left' halo
  - else if my thread id == max thread id
    - exchange 'right' halo
  - end if
- perform Jacobi relaxation
- OpenMP barrier
- calculate global \( \Delta \)
- update old array
- end while
Multiple - Communication

In order to recognise which message is send by which thread, tagging system is used.

A tag is an unique integer value which is based on the thread id and direction in which the message is being sent.
A blocking MPI call means that the program execution will be suspended until the message buffer is safe to use.
Communication – nonblocking

An MPI non-blocking call returns immediately after the call is initiated.

By moving the communication to the background the processors can use the former waiting time for computation.
## HPCx Phase 3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System size</strong></td>
<td>160 IBM eServer 575 nodes</td>
</tr>
<tr>
<td><strong>Node size</strong></td>
<td>16 processors, 32 Gbytes of main memory</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>2560 processors</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>IBM Power5, 1.5GHz 64-bit RISC</td>
</tr>
<tr>
<td><strong>Cache:</strong></td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>64KB instruction, 32 KB data</td>
</tr>
<tr>
<td>L2</td>
<td>1.9MB shared between 2 proc</td>
</tr>
<tr>
<td>L3</td>
<td>36MB shared between 2 proc</td>
</tr>
<tr>
<td><strong>Interconnect</strong></td>
<td>IBM High Performance Switch (HPS)</td>
</tr>
</tbody>
</table>

**HPCx system features**
Number of processors is fixed: 256 processors, that is 16 SMP nodes.

Problem size is fixed per processor:
- 96x96x96
- 192x192x192 array of doubles independent of the balance between number of threads and processes

Factors connected with domain decomposition shape are excluded.
None of the mixed-mode versions was able to outperform pureMPI.

Larger message size

4x4 is the most effective topology

IBM MPI does not handle multiple messages very well

IBM MPI does not handle multiple messages very well.

Point-to-Point

Delta

Jacobi

MPI communication

Global Delta

Jacobi relaxation algorithm

Mixed Mode Programming
Communication

Communication and computation overlapping doesn’t work

Time (seconds)

- Point-to-Point
- Delta
- Jacobi

Mixed Mode Programming
Basic non-blocking 96x3

Point-to-Point communication overhead is more visible for smaller problem size.
Derived data types

Manual message assembly and disassembly - more effective
What’s going wrong?

• 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.
What’s going wrong?

• 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 inter-thread synchronisation than a pure OpenMP version.
  ▫ Implicit point-to-point synchronisation in MPI 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.
Simple 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)
    ENDO
```

Implicit barrier added here

Intra-node messages overlapped with inter-node

Inter-thread communication occurs here
Conclusions

• None of the mixed mode versions managed to outperform the pure MPI version.
  ▫ Funnelled version 20% slower than pure MPI
  ▫ Ineffective point-to-point communication (Master Only, Funnelled)
  ▫ Limited multi-threading support provided by the MPI library (Multiple)
  ▫ Slow OpenMP Barrier

• There are several scenarios where mixed mode could show its potential e.g.:
  ▫ Codes with “significant” data replication