



# Programming next-generation HPC systems: the mixed-mode model

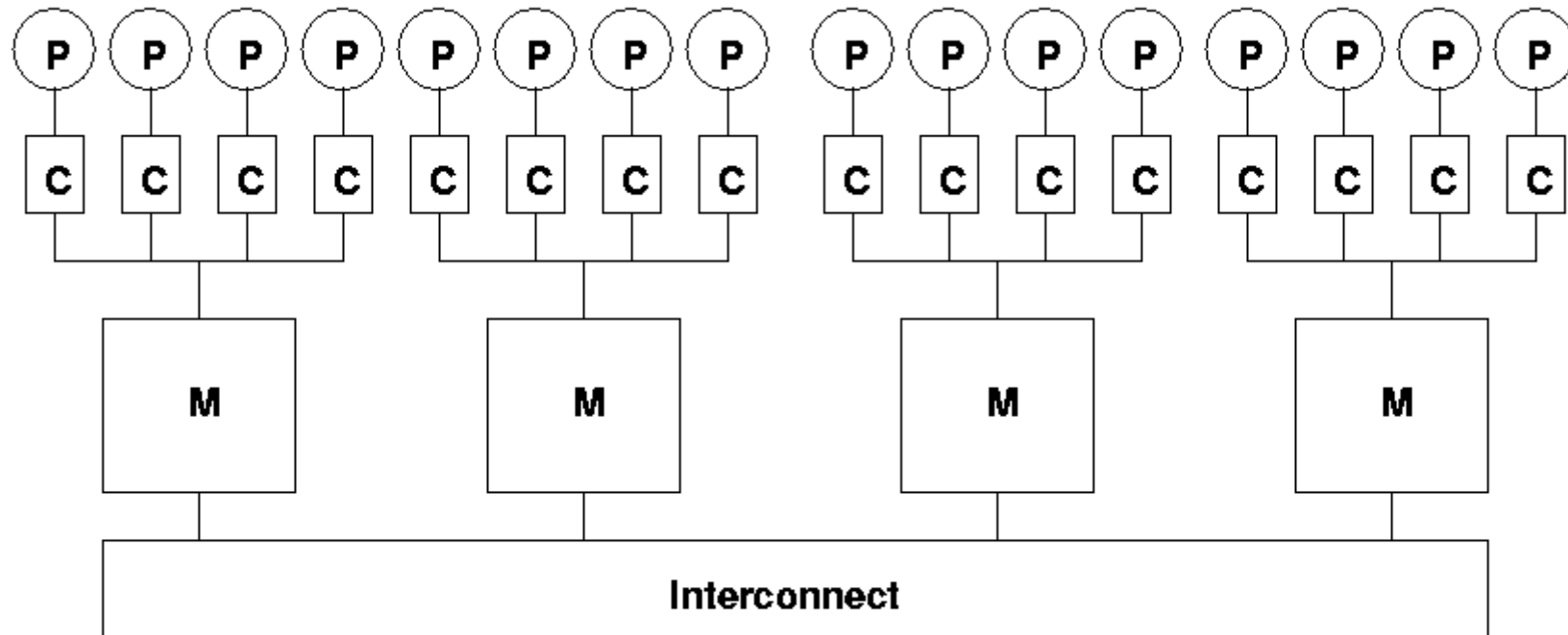
Mark Bull and Lorna Smith

EPCC

`{m.bull,l.smith}@epcc.ed.ac.uk`

---

- ▶ In recent years there has been a trend towards *clustered architectures*
  - ▶ Distributed memory systems, where each node consist of a traditional shared memory multiprocessor (SMP).
  - ▶ Single address space within each node, but separate nodes have separate address spaces.
  - ▶ Commodity based solution: SMP nodes are mass market products.
-



IBM SP, NEC SX-5, Compaq Alphaserver SC, Fujitsu VPP5000,  
Sun Enterprise/SunFire clusters, Cray SV2,.....

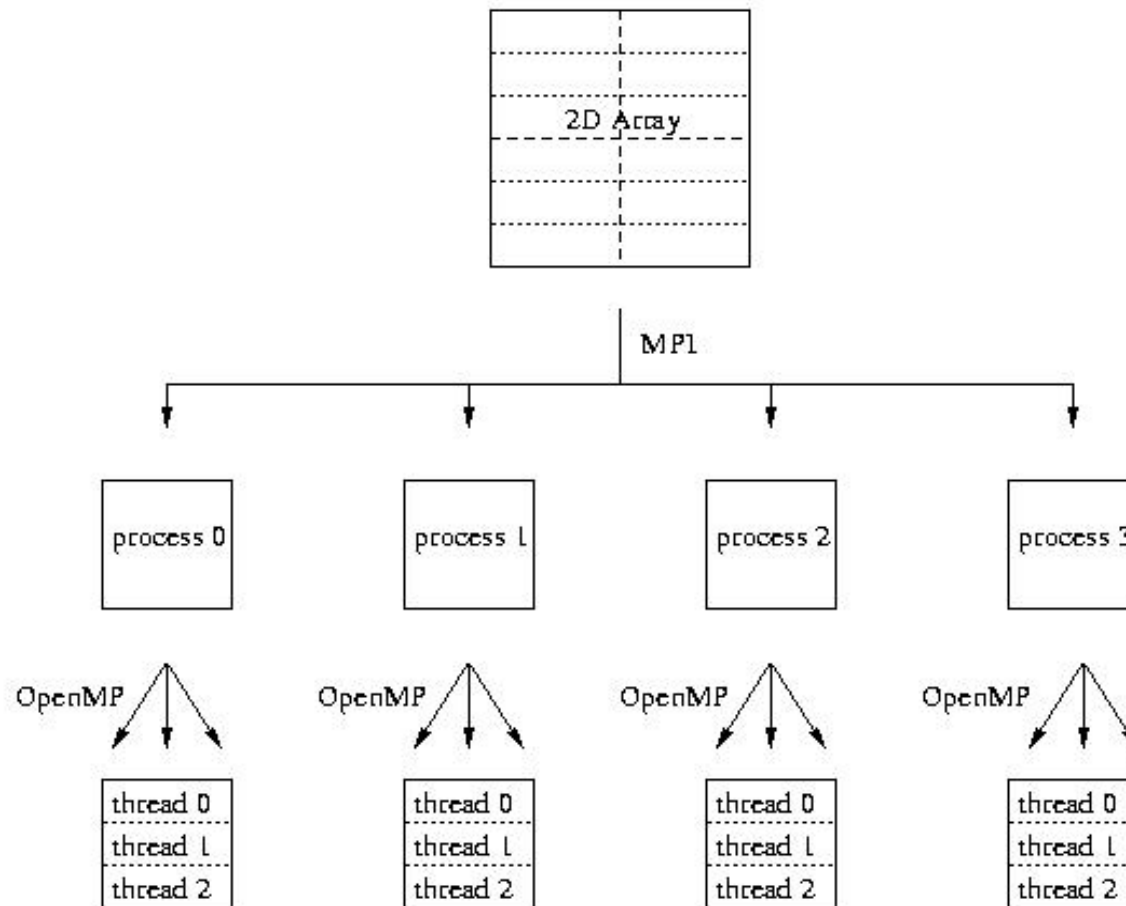
- ▶ Could use message passing (MPI) across whole system.
  - ▶ Cannot (in general) use OpenMP across whole system
    - requires support for single address space
    - possible in hardware (e.g. SGI Origin), but expensive.
    - possible in software, but inefficient.
  - ▶ Could use something else (HPF, Co-array Fortran, HPC++)
    - user inertia
    - standardisation/portability issues.
  - ▶ Could use OpenMP *within* a node and MPI *between* nodes
    - is there any advantage to this?
-

## ▶ MPI

- characteristics
  - distributed memory model
  - explicit control parallelism
- advantages:
  - data placement problems rarely observed
  - implicit synchronisation with subroutine calls
  - portable to distributed and shared memory machines
- disadvantages:
  - development and debugging: time consuming
  - communications overhead
  - large message granularity often required to minimise latency
  - global operations can be expensive

## ▶ OpenMP

- characteristics:
    - shared memory model
    - implicit communication
  - advantages
    - relatively easy to implement
    - better use of the shared memory environment (in theory)
    - both course and fine grain parallelism are effective
  - disadvantages
    - limited to shared memory machines
    - data placement may cause problems on NUMA systems
-



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.
  - ▶ 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 thread safety is assumed, portability will be reduced
    - system level control number of threads/processes not standardised
    - batch environments have varying amounts of support for mixed mode codes.
  - ▶ Need to make sure code functions correctly (with conditional compilation) as stand-alone MPI code and as stand-alone OpenMP code.
-

Six possible performance reasons for mixed OpenMP/MPI codes:

1. Intra-node MPI overheads
  2. Poorly scaling MPI codes
  3. Replicated data
  4. Restricted MPI process numbers
  5. Limited MPI process numbers
  6. Computation power balancing
-

- ▶ 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, if non-thread-safety of MPI is assumed.
    - Implicit point-to-point synchronisation may be replaced by (more expensive) barriers.
    - If, 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.
-

```
!$omp parallel do
```

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

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

```
END DO
```

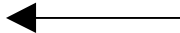
Implicit barrier added here



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

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

Intra-node messages overlapped with inter-node



```
!$omp parallel do
```

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

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

```
ENDDO
```

OpenMP communication occurs here



- ▶ 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.
  
  - ▶ Should improve performance on large data sizes through reduced swapping.
-

- ▶ 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
  - ▶ Most likely to be successful on fat node clusters (few MPI processes)
-

- ▶ Some MPI codes cannot run on arbitrary numbers of processors
    - e.g. may be restricted to powers of 2
  - ▶ Some SMP's don't have power-of-two numbers of processors
  - ▶ May be easier to exploit these numbers of processors with mixed code than to re-engineer the MPI code.
-

- ▶ MPI library implementation may not be able to handle 100's of processes adequately.
  - ▶ Only likely to be an issue on very large systems.
  - ▶ Mixed MPI/OpenMP implementation will reduce number of MPI processes.
-

- ▶ Dynamic load balancing technique
    - Huang and Tafti (1999)
  
  - ▶ Mixed mode code
    - work is initially partitioned equally between MPI processes
    - when a process becomes overload, it increases the number of OpenMP threads
    - requires flexible job scheduling policy
    - better suited to shared memory machines than to clusters.
-

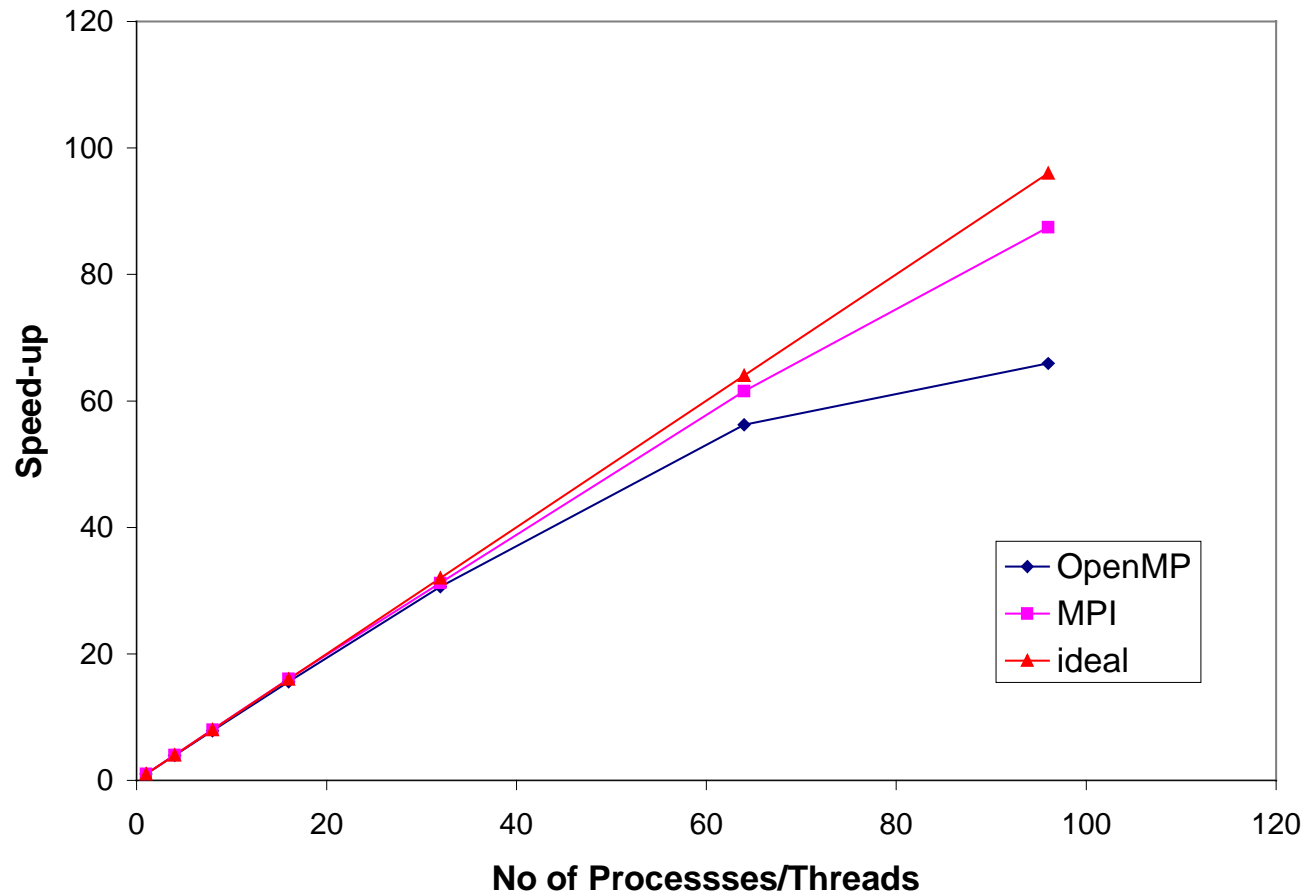
- ▶ Aim
    - to develop a real mixed-mode application
    - to assess the performance benefits
    - to exploit SMP clusters effectively
  - ▶ Quantum Monte Carlo code
    - Electronic structures HPCI consortium
    - Computationally intense: successful MPI version already exists
    - Predicting the electronic structure and properties of real materials
    - f77 / f90 code
  - ▶ Quantum Monte Carlo techniques
    - Variational Monte Carlo (VMC)
    - Diffusion Monte Carlo (DMC)
-

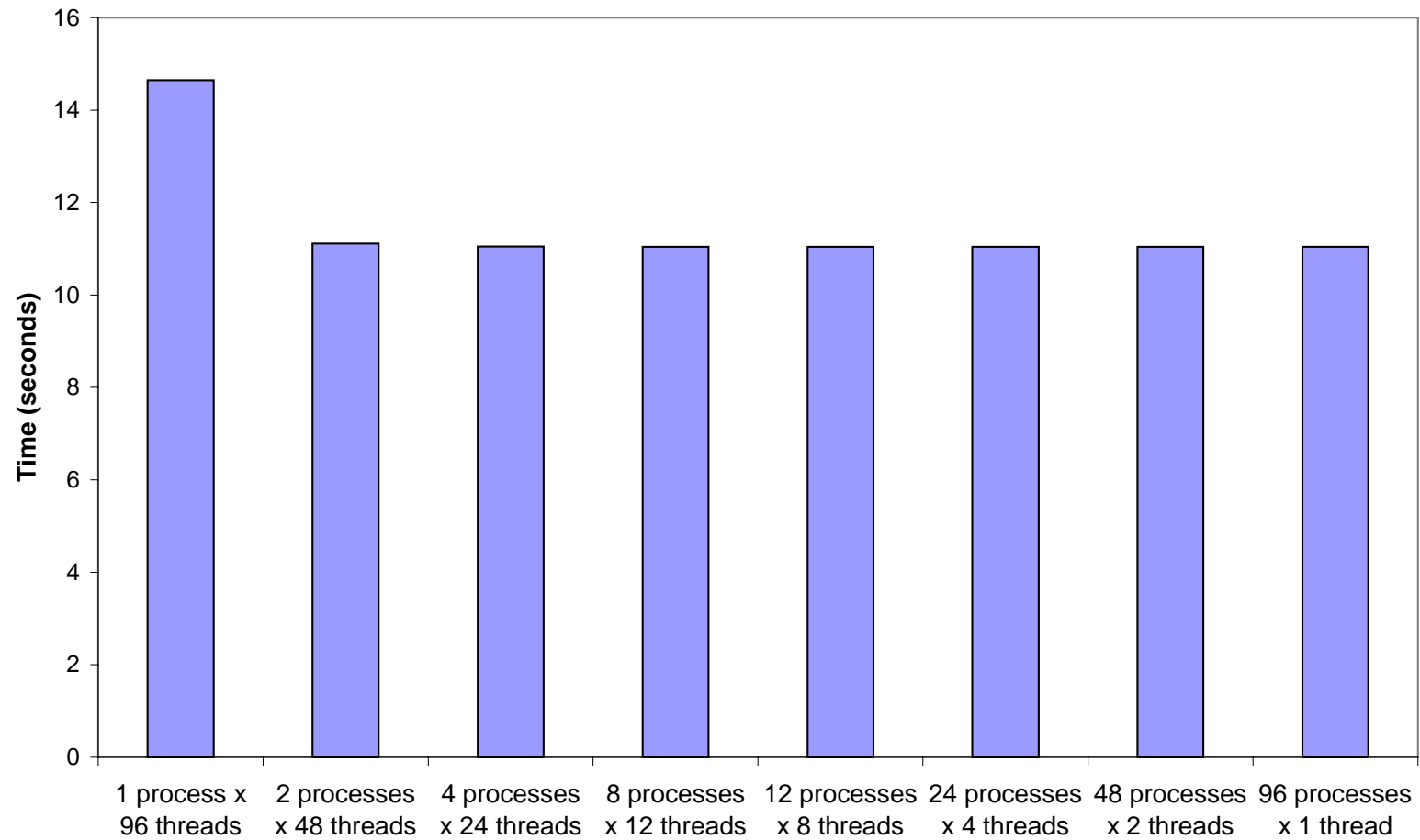
- ▶ 1) Initialise the ensemble of 'walkers'
  - ▶ 2) Update each walker in the ensemble. For each walker:
    - Move the electron
    - calculate the local energy and other variables of interest
    - calculate the new weight
    - accumulate the local energy contributions for this walker
    - breed or kill new walkers based on the energy
  - ▶ 3) Evaluate new generation averages
  - ▶ 4) After N generations calculate new 'block' averages
  - ▶ 5) Iterate steps 2-4 until equilibrium is reached
-

- ▶ Master-slave model
    - Electron configurations divided between slave processes
    - Master process broadcasts parameters to slave processes
    - Slave process evaluate properties dependant on subset of electron configurations: return properties to master process
    - Repeat until convergence achieved
  - ▶ Variational Monte Carlo
    - each process assigned a fixed no of electron configurations
    - inter-process communication only required for the final result
  - ▶ Diffusion Monte Carlo
    - electron configurations created/ annihilated: no. on each process may change after each 'block': poor load-balancing?
    - electron configurations redistributed between the processes after each block
-

- ▶ Majority of time spent in the loop over electron configurations
    - Compiler directives placed around this loop
    - Course grain parallelism
      - comparable level to MPI code: allows direct comparison
      - 150 variables, numerous modules and subroutine calls
    - Two principle shared arrays
      - start of loop: copied to temporary private arrays
      - end of loop: copied back to shared arrays
    - Electron configurations change: size of temporary arrays change
      - Ensure arrays copied back in the same order as sequential version: ORDERED statement
  - ▶ Mixed code
    - Hierarchical model: in general OpenMP beneath MPI
    - exceptions: MPI\_BCASTs during 1st iteration: CRITICAL section
-

- ▶ SGI Origin 2000, 300MHz R12000 processors





- ▶ OpenMP scaling
    - similar to MPI to 32 processors, tails off considerably above 64
  - ▶ Thread / process combinations
    - similar times, exception 96 threads / 1 process
  - ▶ MPI scaling
    - redistribution of electron configurations between processes
    - all-to-one + point-to-point communications
  - ▶ OpenMP scaling
    - cc-NUMA architecture of the Origin 2000
      - data placement policy changes have no effect
    - MPI calls within thread sequential regions
      - only occur during the first iteration
    - ORDERED statement
    - hierarchical model - added synchronisation
-

- ▶ Clearly not the most effective mechanism for all codes
  - ▶ Significant benefit may be obtained in certain situations:
    - poor scaling with MPI processes
    - replicated data codes
    - restricted MPI process codes
    - limited scaling MPI implementations
  - ▶ Unlikely to benefit well optimised existing MPI codes.
  - ▶ Portability and development / maintenance considerations.
-