

Porting the DL_POLY molecular dynamics package to GPGPUs

C. Kartsakalis, G. Civario, R. Nestor

Irish Centre for High-End Computing (ICHEC)

www.ichec.ie



About ICHEC

- National HPC service provider for Ireland
- Established in 2005
- Funded by Science Foundation Ireland (SFI) and the Higher Education Authority (HEA)
- Activities Include:
 - User support & training
 - Research (novel architectures)

About DL_POLY 3

- Molecular dynamics package
- Developed at STFC Daresbury Laboratory
- Written in Fortran 90, no dependencies on external libraries
- Parallelisation via domain decomposition and MPI

DL_POLY 3 Runtime Profile

- ~80 – 90% of CPU time is spent in four components:

DL_POLY Component	Purpose
Two-Body Forces	Inter-atomic force computation
Link-cell Pairs	Atom neighbour list construction
Constraints Shake VV	Enforce bond constraints between atoms
Ewald SPME Forces	Coulombic energy and force computation in periodic systems

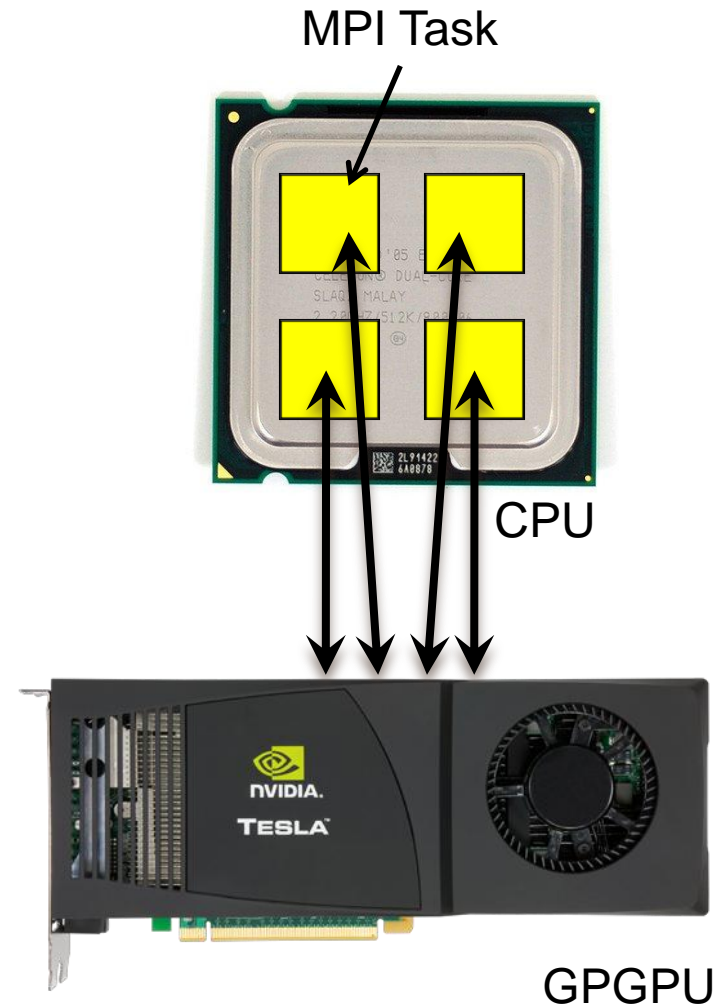
- GPGPU acceleration implemented using CUDA

Current Hardware - GPGPU Clusters

- Pure MPI code – how to associate MPI tasks with GPGPUs?
- CPU Core : GPGPU ratio is normally not 1 : 1
- Typical Configuration – 4 CPU cores per attached GPGPU

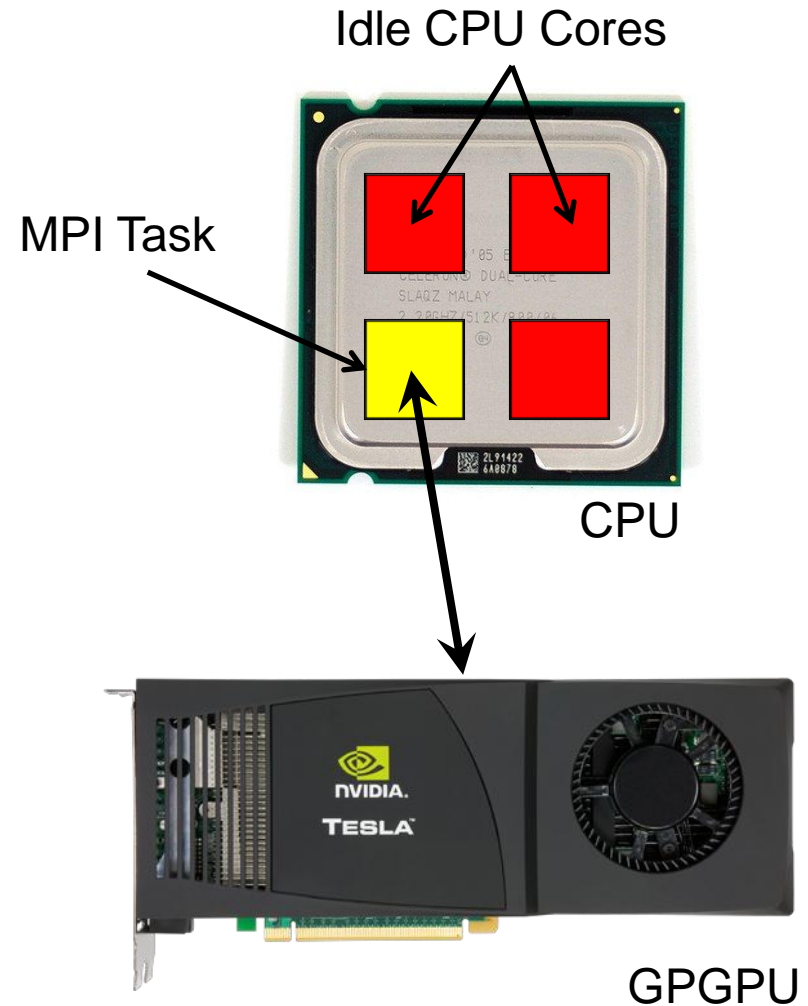
MPI task – GPGPU association

- Possibility 1:
 - 4 MPI Tasks - 1 GPGPU
- Problem: Sharing not well supported on current (Tesla) hardware
- Serialisation at GPGPU level



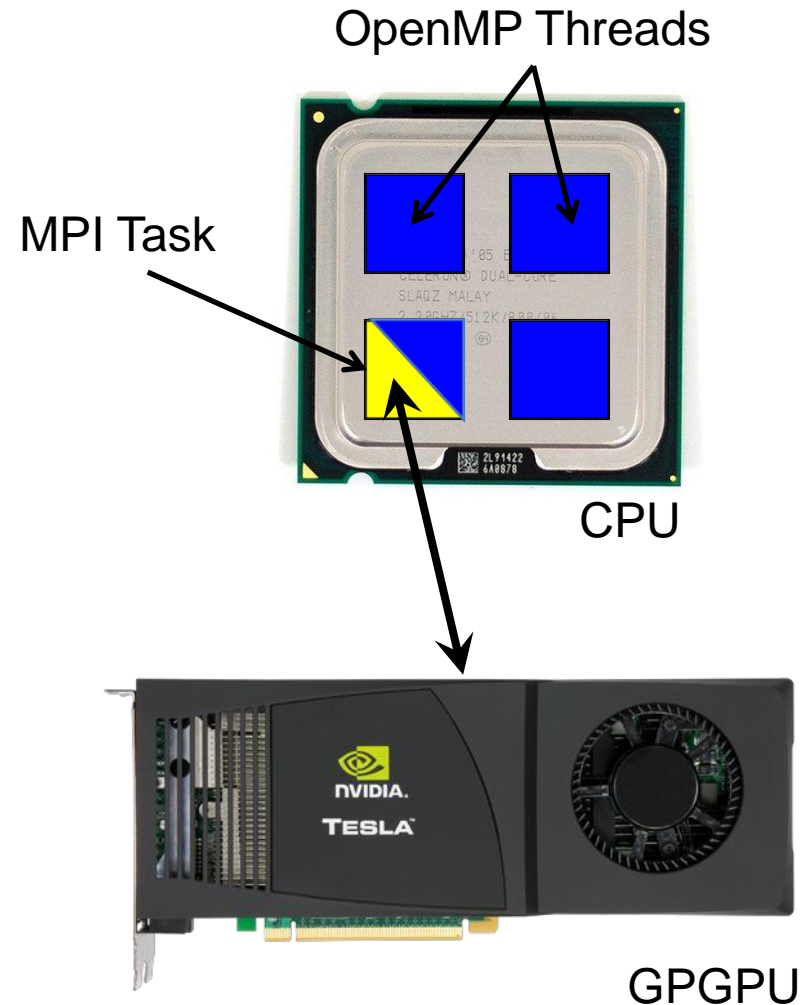
MPI task – GPGPU association

- Possibility 2:
 - 1 MPI Task - 1 GPGPU
- Problem: Idle CPU cores



MPI task – GPGPU association

- Solution:
 - 1 MPI Task - 1 GPGPU
 - 4 OpenMP threads per MPI Task
- All available hardware is used



MPI+GPGPU+OpenMP – problems

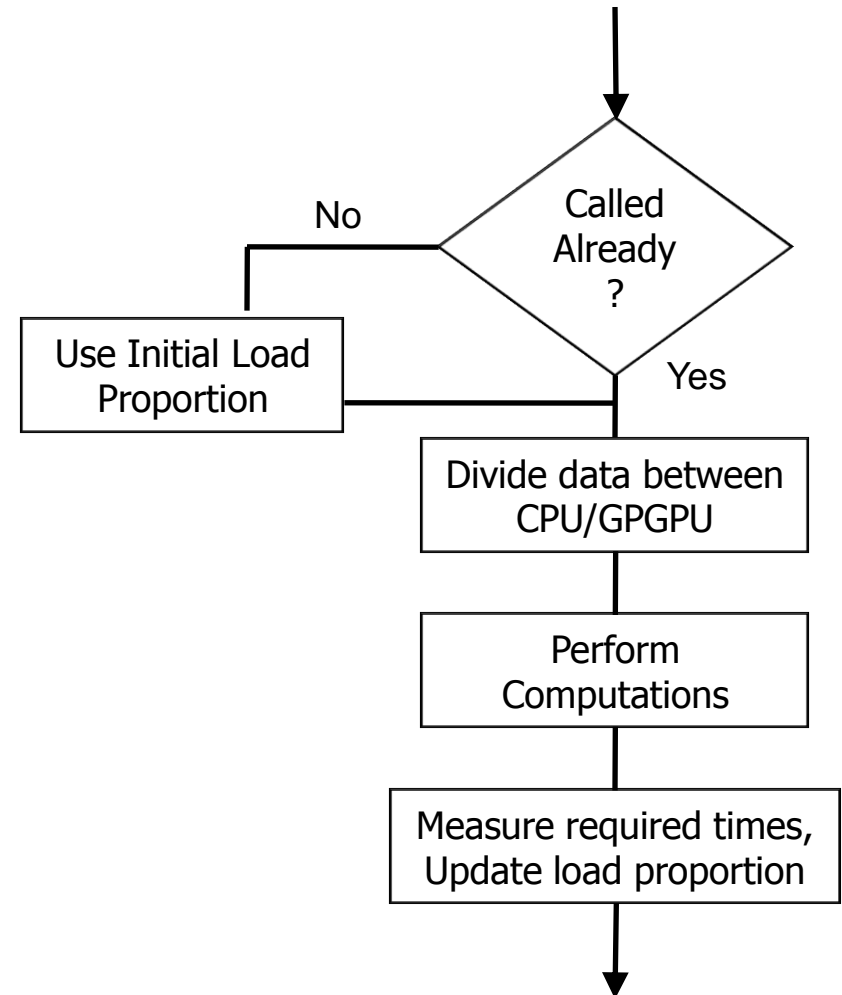
- Non-accelerated portions of the code => idle cores
- Load Balancing

GPGPU/OpenMP Load Balancing

- How to divide work between GPGPU/OpenMP threads?
- Depends on:
 - Relative GPGPU/CPU performance – machine dependent
 - Computation at hand
- Can't be predicted in advance

GPGPU/OpenMP Load Balancing

- Solution:
 - Dynamically adjust workload between CPU and GPGPU
- Load proportions stabilise after ~ 3 iterations
- Tuning not required

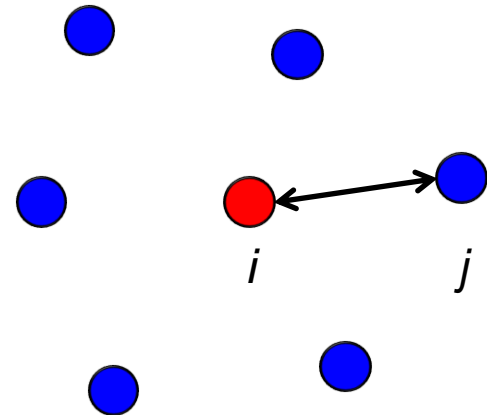


DL_POLY 3 Accelerated Components

DL_POLY Component	Purpose
Two-Body Forces Link-cell Pairs	Inter-atomic force computation Atom neighbour list construction
Constraints Shake VV Ewald SPME Forces	Enforce bond constraints between atoms Coulombic energy and force computation in periodic systems

Two-Body Forces

- Compute forces acting on each atom i
- Loop over neighbours of i and accumulate contributions



Two-Body Forces

- Original Looping Structure:

```
for i = all atoms
  ...

  for j = neighbour atoms of i
    force(j) += ...

  end loop
end loop
```

Two-Body Forces

- Thread Organisation

```
for i = all atoms
```

```
...
```

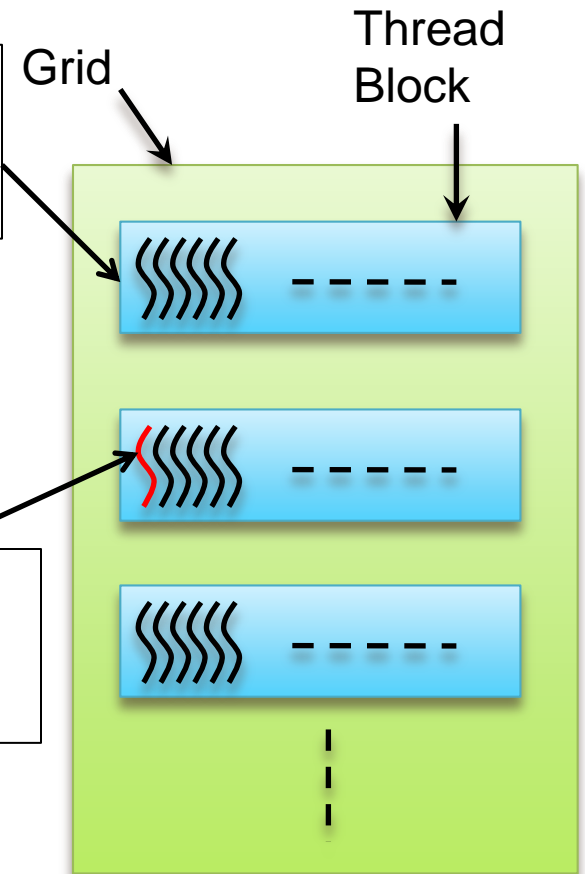
```
for j = neighbour atoms of i  
  force(j) += ...
```

```
end loop
```

```
end loop
```

Each **Thread Block**:
Responsible for a single
'primary' atom, i

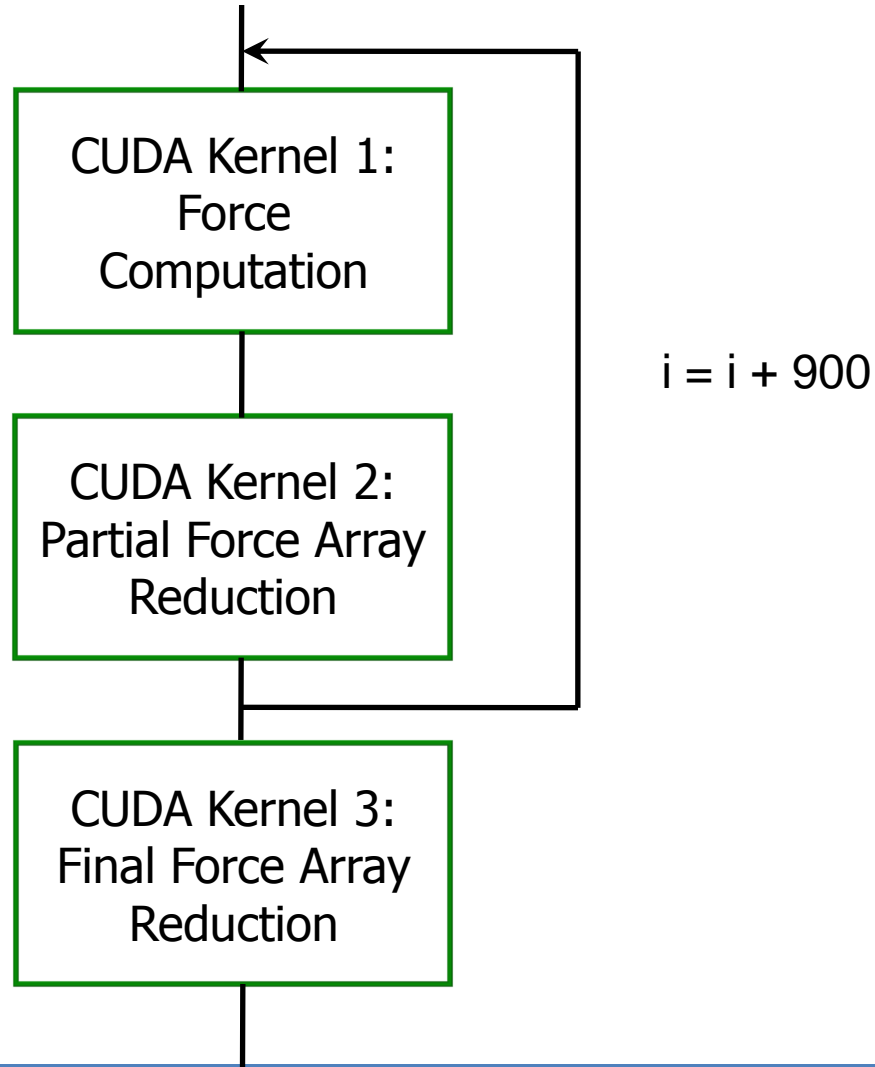
Each **Thread**:
Responsible for a single
neighbour atom, j



Two-Body Forces

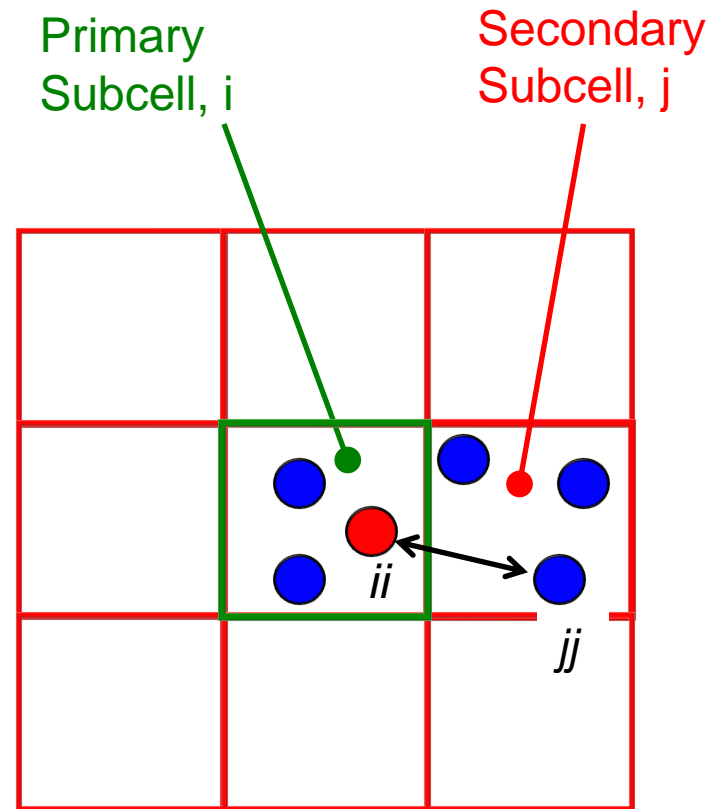
- Each thread block: force contributions of **one** primary particle
- Forces are stored in a temporary block-local array
- => Must also perform a reduction on the force arrays associated with each block
- Reduction is performed in a separate kernel

Two-body Forces



Link-cell pairs

- Builds a neighbour list for each particle
- Loop over all grid cells
- Check distance from atoms in neighbouring cells
- Add to list if less than cutoff radius



Link-Cell Pairs

- Original looping structure

```
for i = all grid cells
```

Each **Thread Block**:
Responsible for a single
grid cell, i

```
for ii = all atoms in cell i
```

Thread block
Z-Dimension

```
for j = neighbour cells of i
```

Thread block
Y-Dimension

```
for jj = all atoms in cell j
```

Thread block
X-
Dimension

```
if (inter-atomic distance < cutoff)  
  //Add to neighbour list of atom i  
  list(neighb,i) = jj  
  neighb++  
endif
```

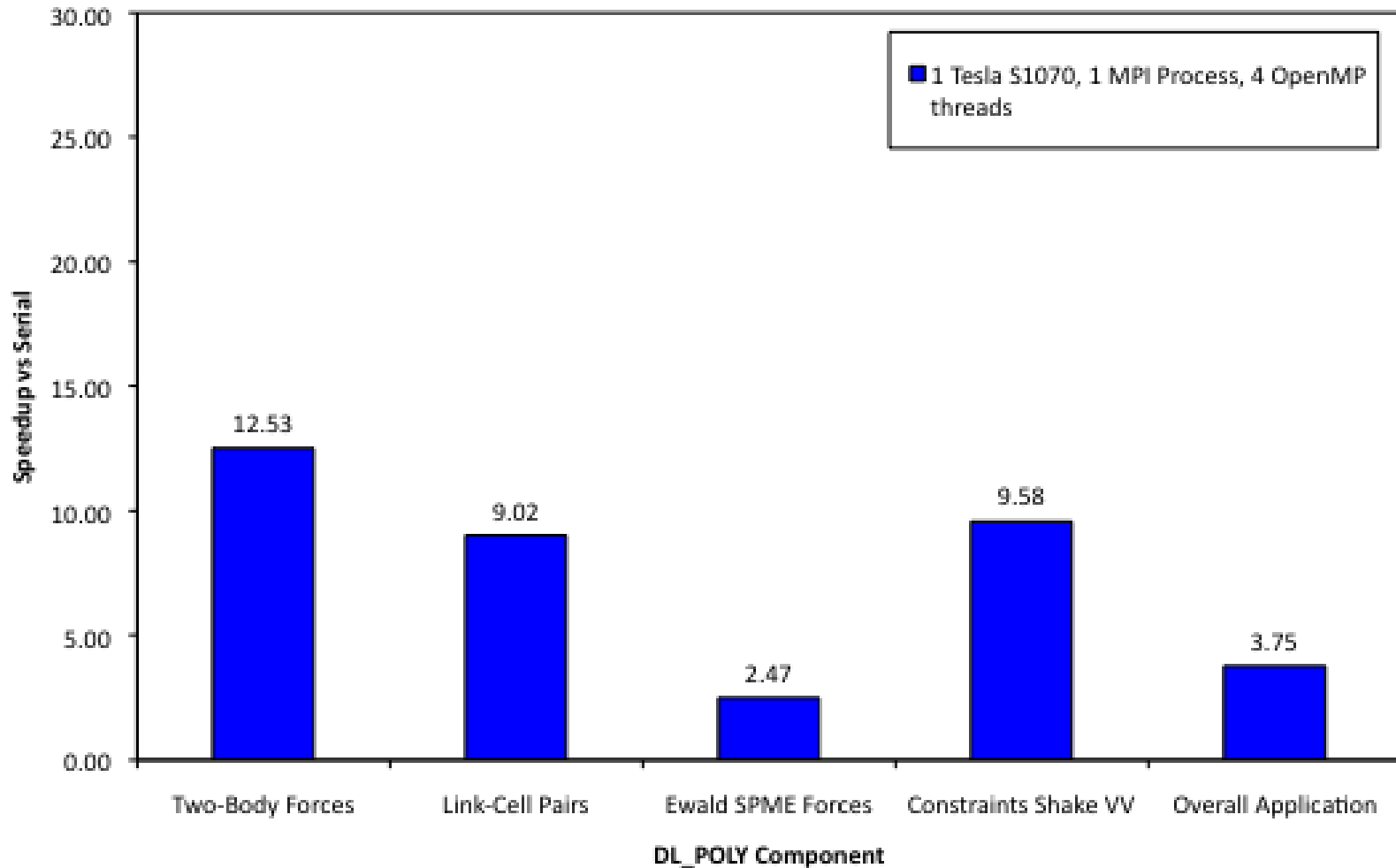
Link-Cell Pairs

- No additional reductions are performed
- Neighbour list is written directly to global memory

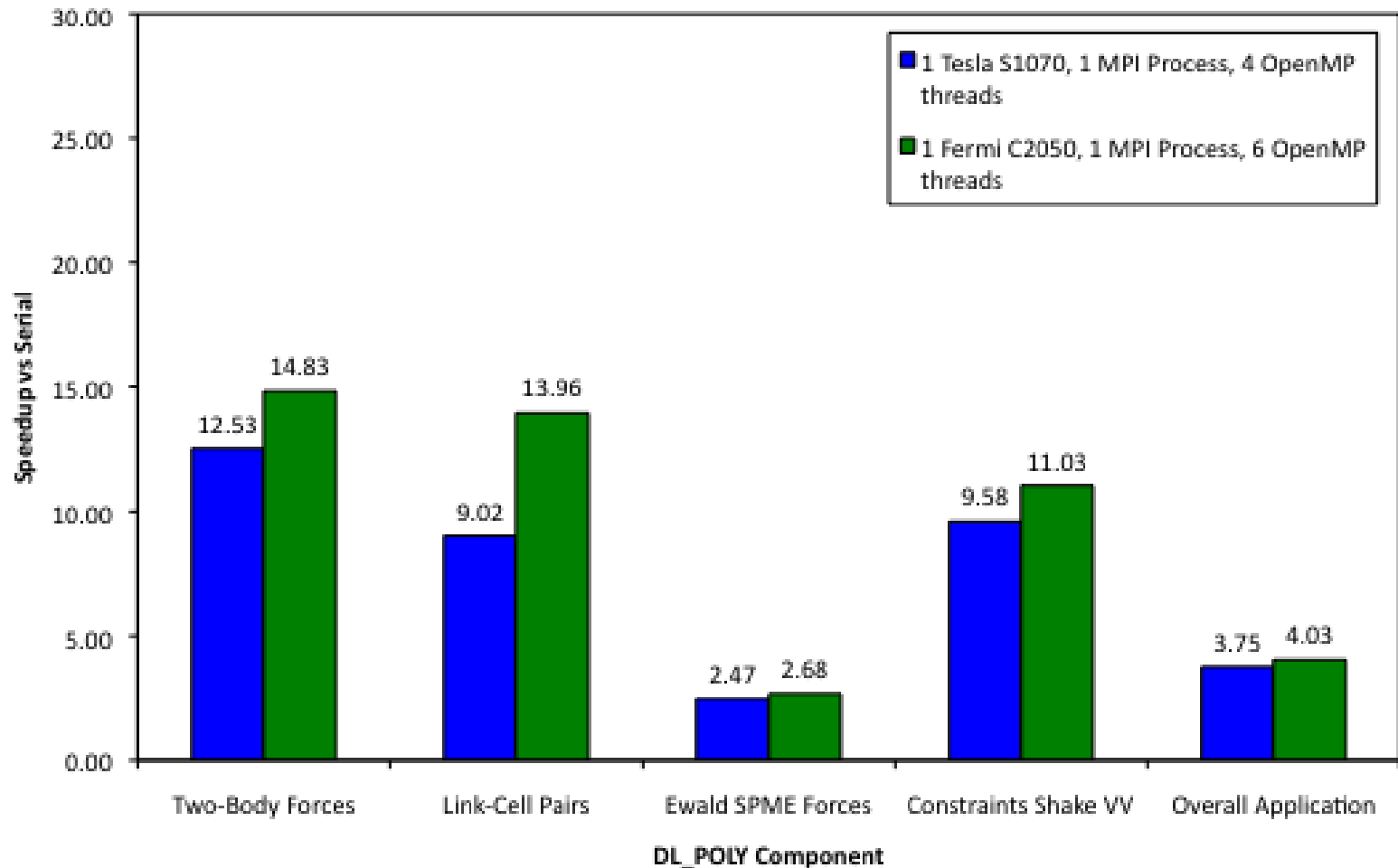
Results

- 2 Hardware Configurations
 - Tesla C1060, Intel Xeon 5560, 4 OpenMP threads
 - Fermi C2050, Intel Xeon 5650, 6 OpenMP threads
- 2 benchmarks
 - Sodium Chloride, 216000 Ions (TEST 2)
 - DMPC in Water, 413896 atoms (TEST 4)
- Speedup vs Serial, **Double Precision**
- Original CPU results are reproduced

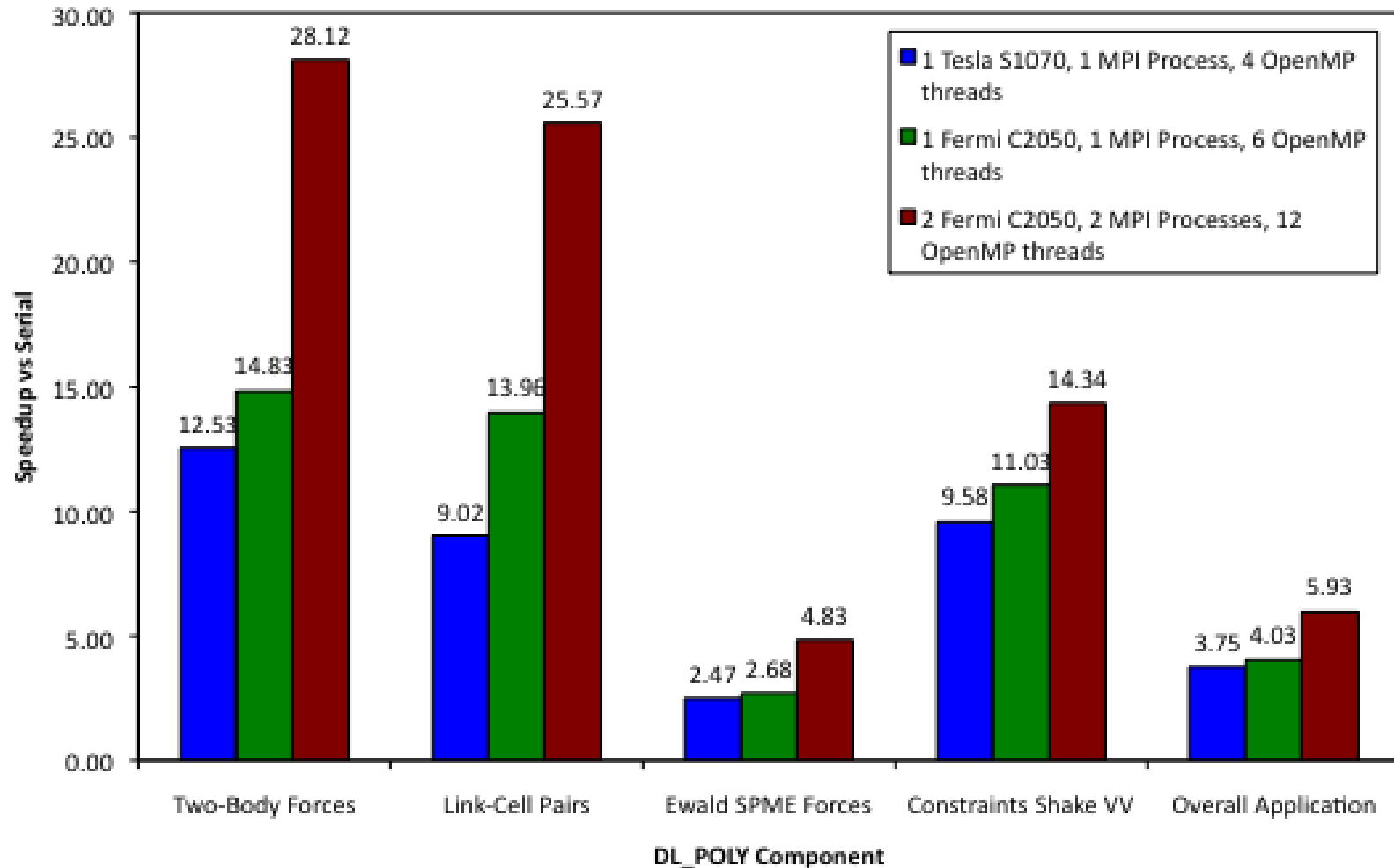
Results: Speedup for TEST 4



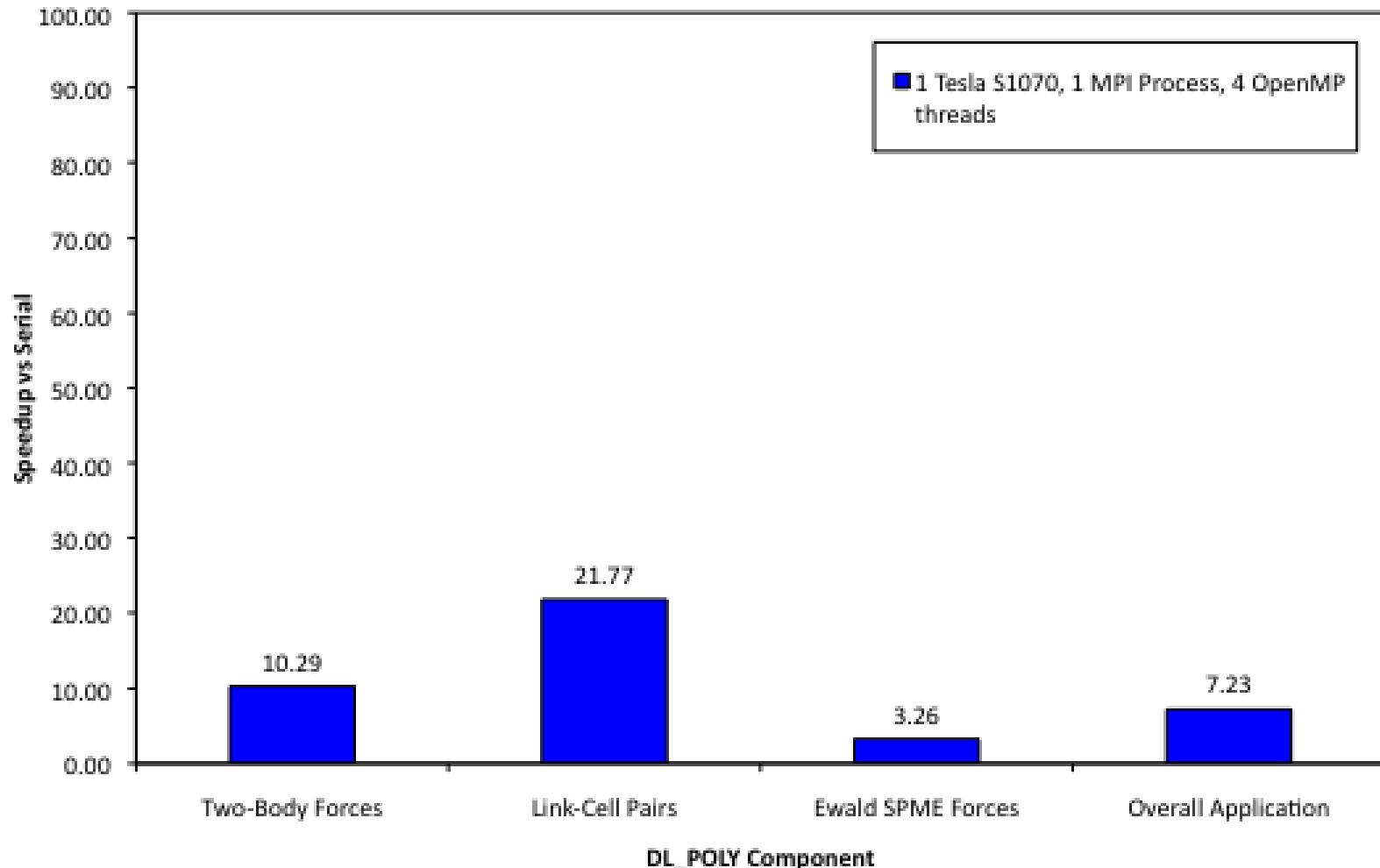
Results: Speedup for TEST 4



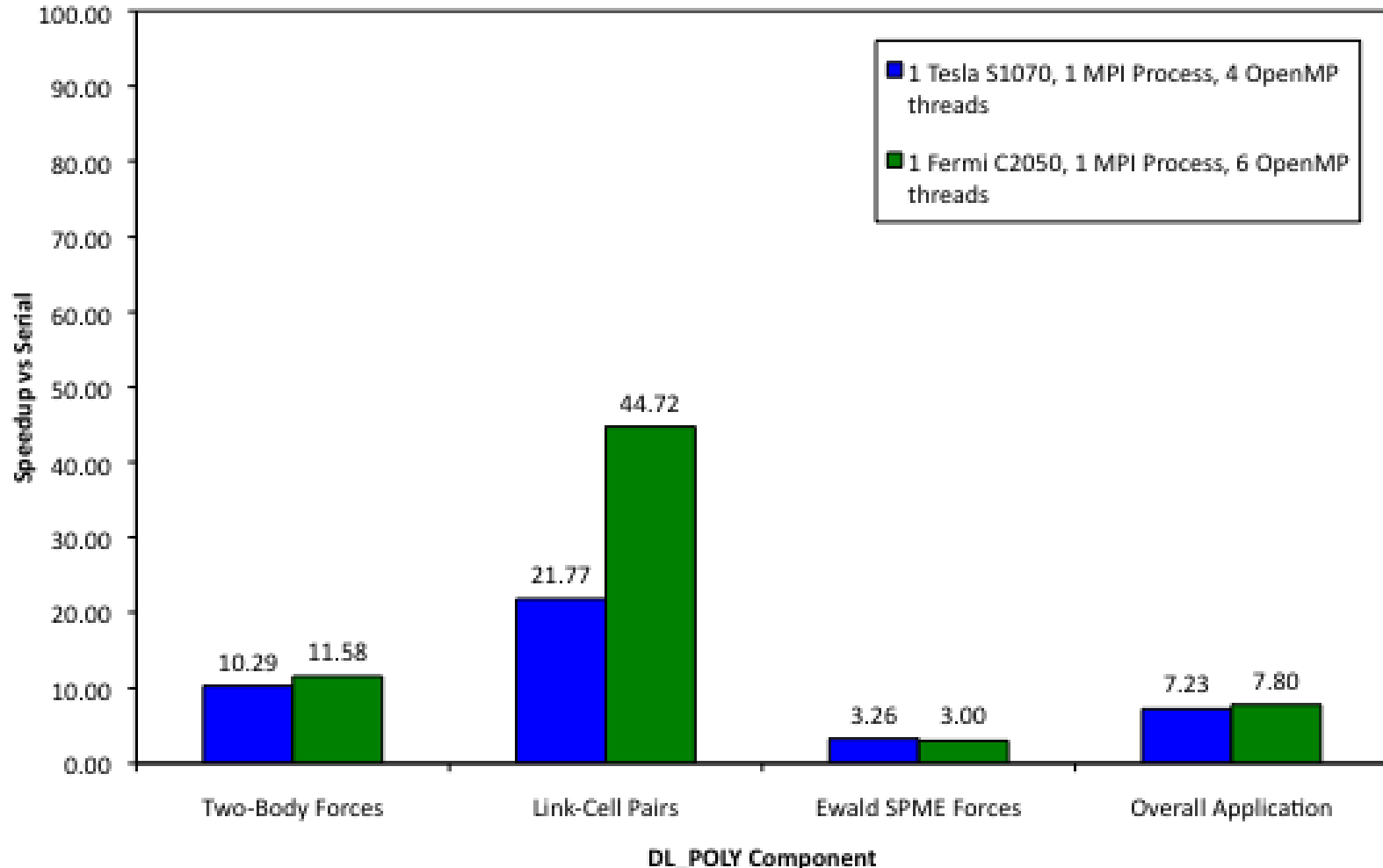
Results: Speedup for TEST 4



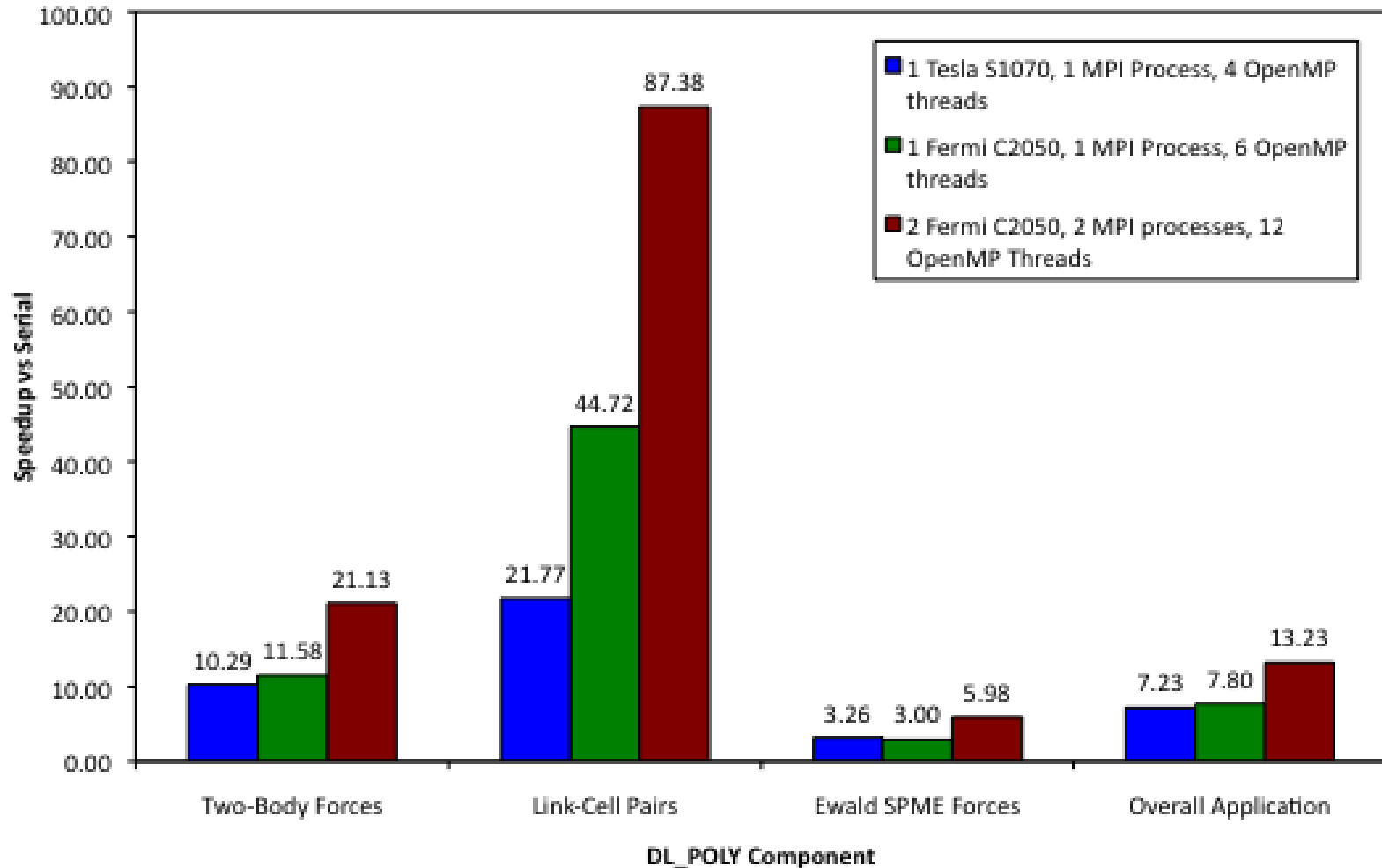
Results: Speedup for TEST 2



Results: Speedup for TEST 2



Results: Speedup for TEST 2



GPGPU Overloading

- More MPI processes than GPGPUs
- Inefficient on Tesla – kernels get serialised
- Improved on Fermi
 - Further 1.45x application speedup
 - Better usage of CPU cores – non-accelerated functions are parallelised across cores via MPI

Conclusions

- GPGPU + OpenMP strategy – all available hardware is used
- Dynamic load balancing – obviates the need for tuning on a per-machine basis
- Application speedup of up to $\sim 8x$ with 1 MPI task and 1 GPGPU in double precision

Acknowledgements

- STFC/Daresbury Laboratory: Support and advice
- Supported by Science Foundation Ireland under grant 08/HEC/I1450 and by HEA's PRTL1-C4.