

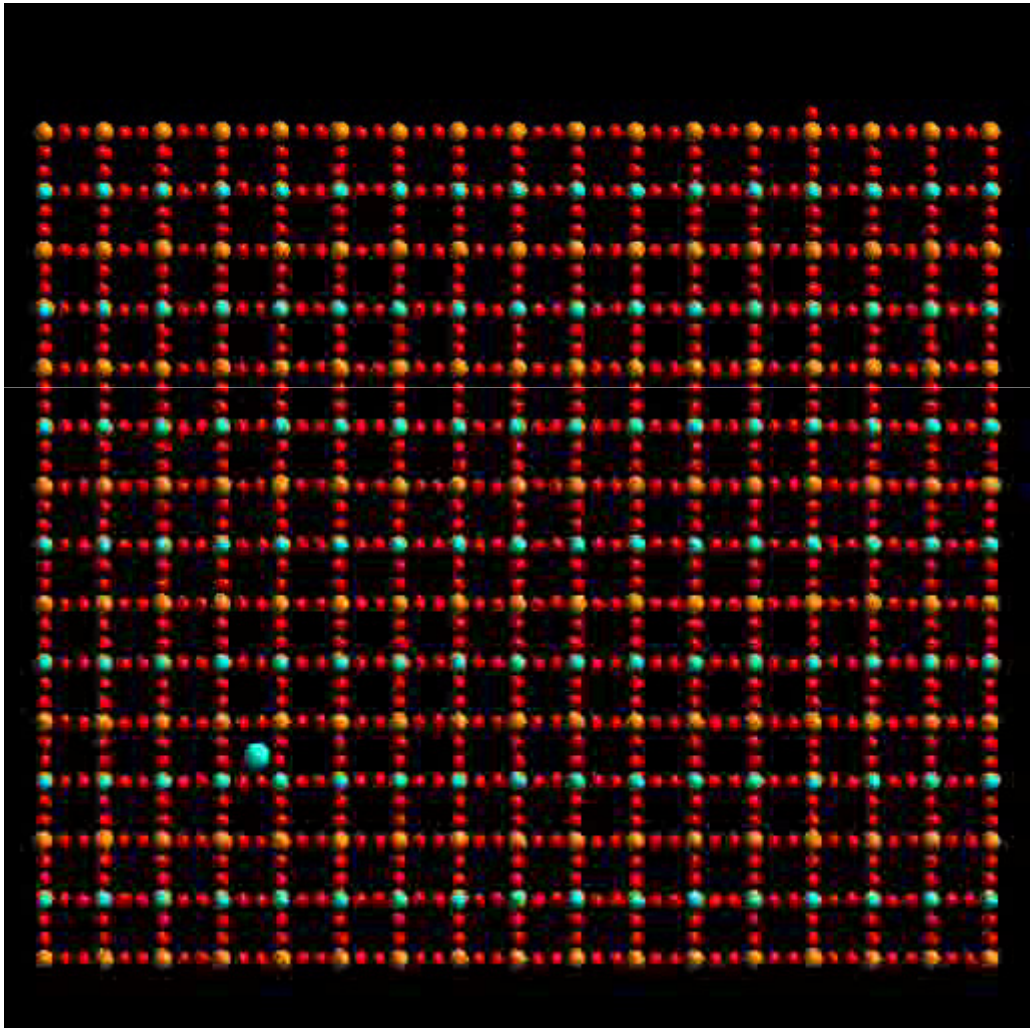


# Optimisation of the I/O for Distributed Data Molecular Dynamics Applications

**Ian Bush, *NAG Ltd. and* Ilian Todorov *and* William Smith, *STFC Daresbury Laboratory***

CUG 2010
Simulation Comes of Age


# What Is MD?




# The DL_POLY_3 MD Package

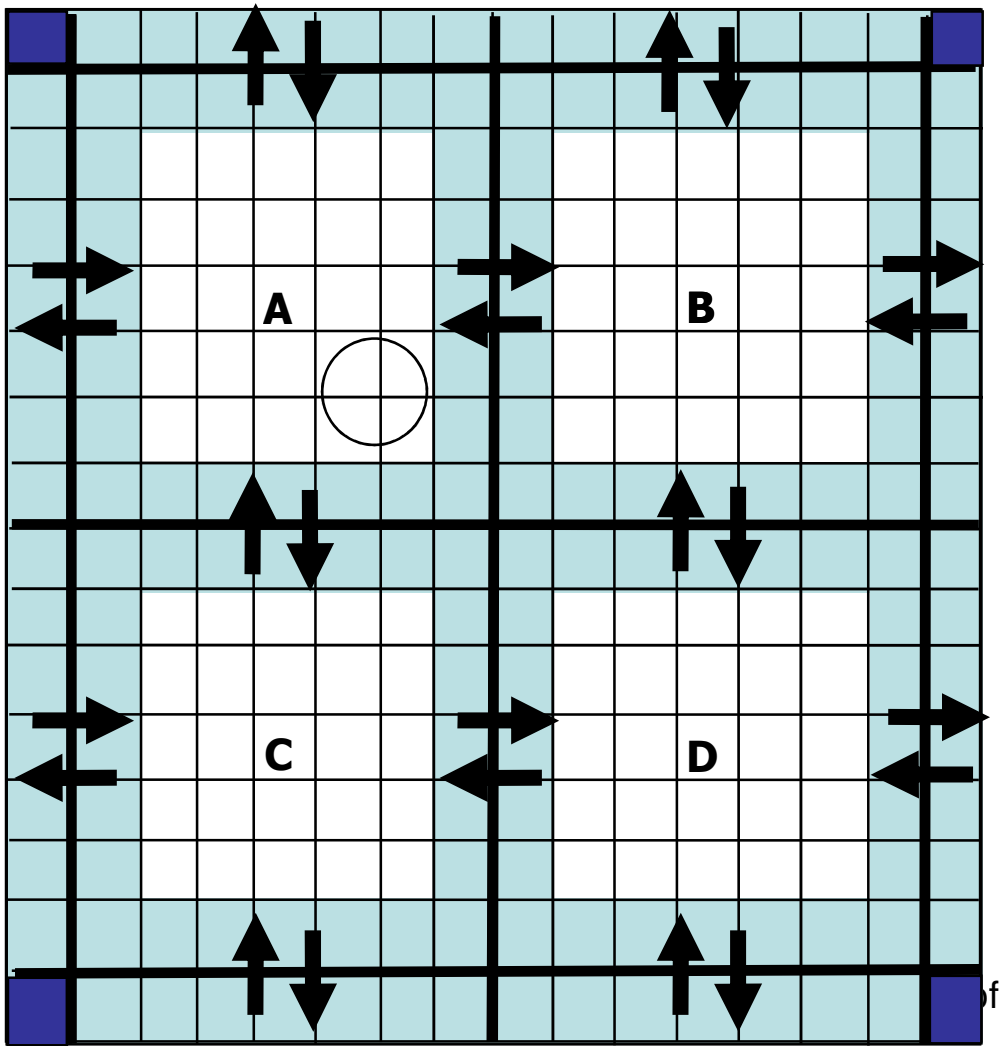
General purpose MD simulation package

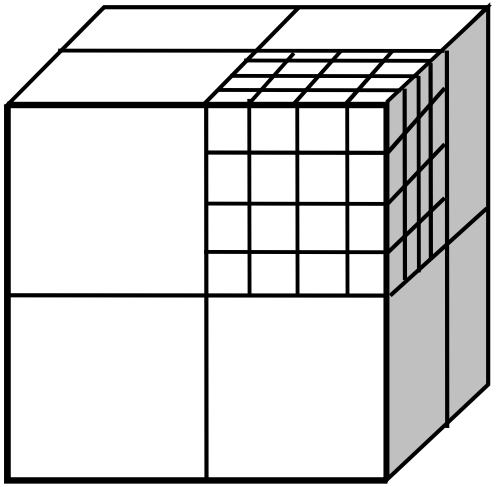
Written by Ilian Todorov and Bill Smith at STFC Daresbury Laboratory

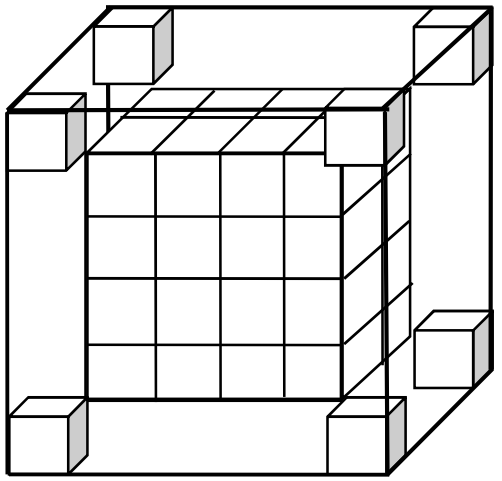
Written in modularised free formatted Fortran 95 - FORCHECK and NAGWare verified

Generic parallelisation (for short-ranged interactions) based on spatial domain decomposition (DD) and linked cells (LC)

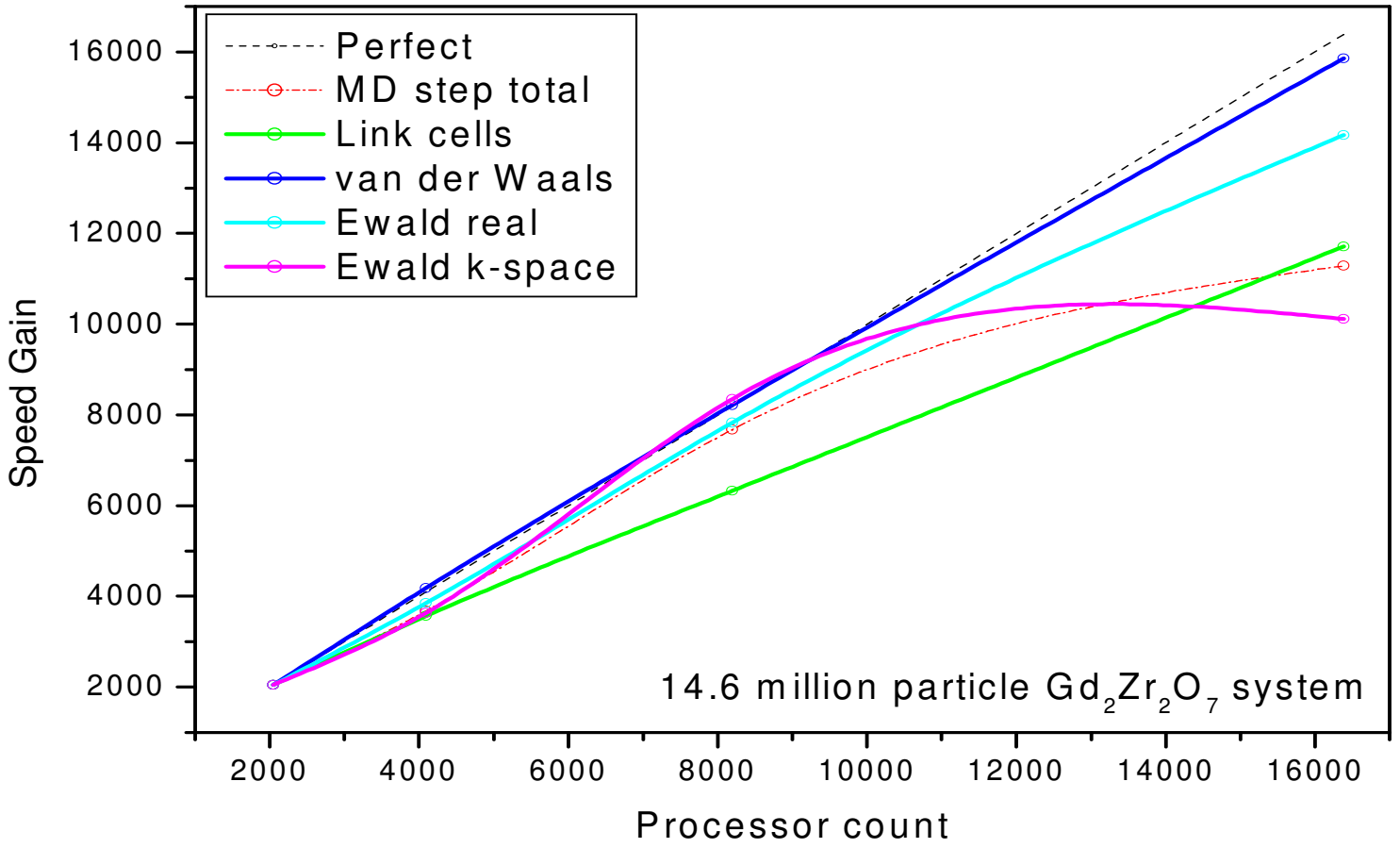
# Domain Decomposition Parallelisation






# How Well Does The Compute Scale? (BG/L)




# So What's The Problem?

For the 14,600,000 particle system on 16,384 processors of the the Jülich BG/L system it takes ~0.5s for a MD timestep

- Fast enough to do science !

~1800s to write the coordinates

- Not fast enough to do science !

Want to write the coordinates every ~100-1000 timesteps

**So while the compute is fast enough the I/O prohibits any useful science being done**


# It's Not Just Blue Gene

14.6 million system on 2048 processors of HECToR Phase 1

- MD time per timestep ~0.7 seconds on Cray XT4
- Configuration read ~100 seconds (once during the simulation)
- Configuration write ~600 seconds


# So What Do We Have To Write?

```
pyrochlore
         2         3   3773000        50     0.00003125     0.00156250
      378.0382791976         0.0000000000         0.0000000000
        0.0000000000       378.0382791976         0.0000000000
        0.0000000000         0.0000000000       378.0382791976
GD                3
    -186.2697242          -188.9656799         -186.3793036
    0.2315100734          -1.673201463         0.9363383539
     13210.65286          -235052.7542          44828.56133
GD                4
    -188.9764926          -186.3753017         -186.3328710
   -0.2949178501          0.9443083034          2.428692460
    -254542.5135           49396.61430          67986.12075
GD                5
    -189.0096634          -183.5772665         -183.4873639
     1.344516913          0.3640837776E-01     -1.250456823
    -21153.56476           1492.614280          949.9063469
GD                6
    -186.2854413          -180.8116309         -183.7179432
   -0.3272091542         -0.3909127980         -2.407327182
    -5003.623307          -288.9791458          5327.259472
```


# And What's the Problem?

**The atoms move!**

An atom can migrate from one processor to another, so the original ordering of atoms is not preserved.

But users' analysis programs (e.g. for visualization) often assume that the ordering is preserved.

So have to rearrange data so that it can be written out in the form the users require.

Also files need to be **portable**


# First Tries

The first writing methods used Fortran Direct Access Files

- If you know the index of the atom you know which record to write to
- So just write to that record


# SWRITE AND PWRITE

Two Methods tried

- SWRITE
  - In turn gather each processors data to core 0
  - And the core 0 does the writing
  - Serial and poor performance
- PWRITE
  - Each core just writes each atom to its correct place
  - Better but still not good enough performance
  - NOT PORTABLE
    - Behaviour not defined by Fortran


# MWRITE

However can easily use MPI-I/O to “simulate” Fortran direct access file

- Create a MPI derived type the length of the record
- Use that as the etype for the fileview
- Now all offsets are almost the same as for Fortran direct access
  - Except indexed from zero
- Thanks to David Tanqueray for this idea
- Leads to ***MWRITE*** – released in DL_POLY 3.09


# MWRITE – The Innards

```
Integer, Parameter       :: recsz    =   73
Character( Len = recsz ) :: record
...
Call MPI_TYPE_CONTIGUOUS( recsz, MPI_CHARACTER, rec_type, ierr )
Call MPI_TYPE_COMMIT( rec_type, ierr )
Call MPI_FILE_OPEN( comm, file_name, flags, MPI_INFO_NULL, file_handle, ierr )
Call MPI_FILE_SET_VIEW( file_handle, 0_MPI_OFFSET_KIND, rec_type, rec_type, &
                datarep, MPI_INFO_NULL, ierr )
...
Write(record, Fmt='(3g20.10,a12,a1)') xxx(i),yyy(i),zzz(i),Repeat(' ',12),lf
rec_mpi_io=6_MPI_OFFSET_KIND+Int(index(i),MPI_OFFSET_KIND)*4_MPI_OFFSET_KIND
Call MPI_FILE_WRITE_AT( file_handle, rec_mpi_io, record, 1, rec_type, status, ierr )
```


# Measuring Performance

Throughout the rest of the talk I shall use two different physical systems to measure the performance of the I/O methods:

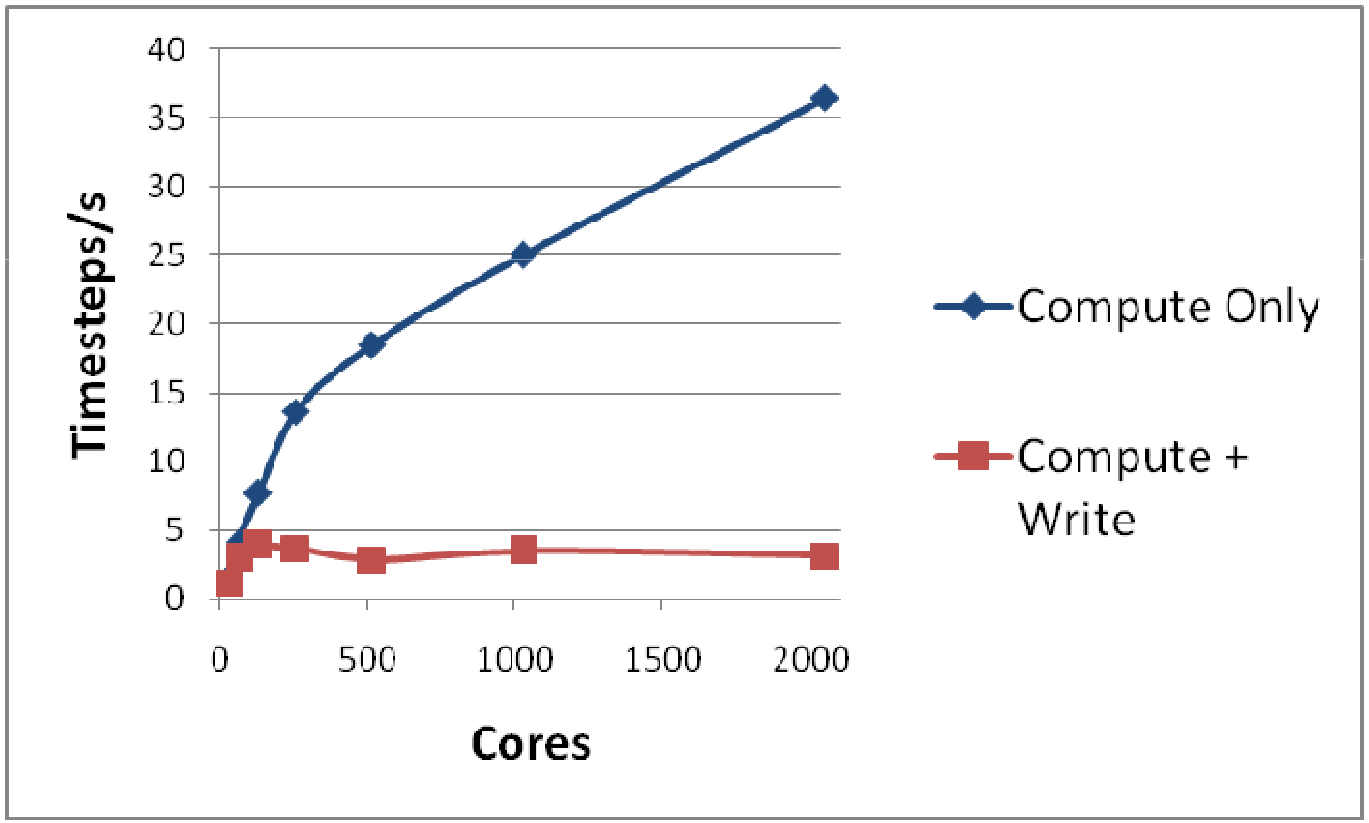
- 216,000 ions of Sodium Chloride. Run for 1000 timesteps and then write the configuration
- As before but 1728000 ions of NaCl

I shall use one computational system

- HECToR Phase2a – Cray XT4 + Lustre

**All default settings used throughout**


# MWRITE – The Performance for 216000 Ions of NaCl




# What's The Problem?

All the processors are writing

- So possible contention at the disk

Only 1 atom's data is being written at one time

- Very short I/O transactions (292 Bytes)


# A Solution?

Gather the data onto a subset of the processors

- The *I/O Processors*
- Do in batches so as to avoid memory overhead

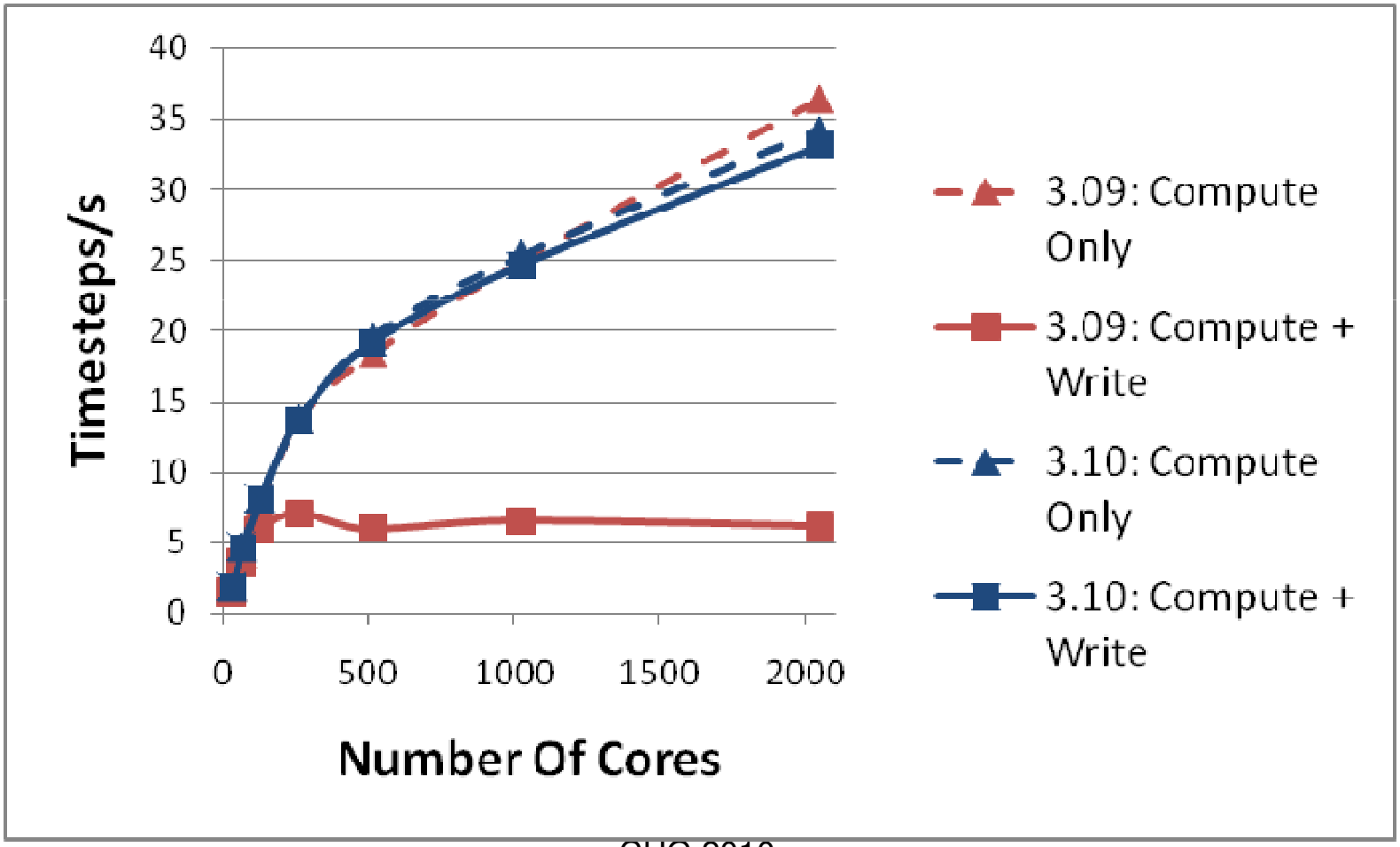
Then sort in parallel across the I/O processors

Finally use MWRITE but can now write many atoms at once

Call this ***MWRITE_SORTED***

- Released in version 3.10 of code

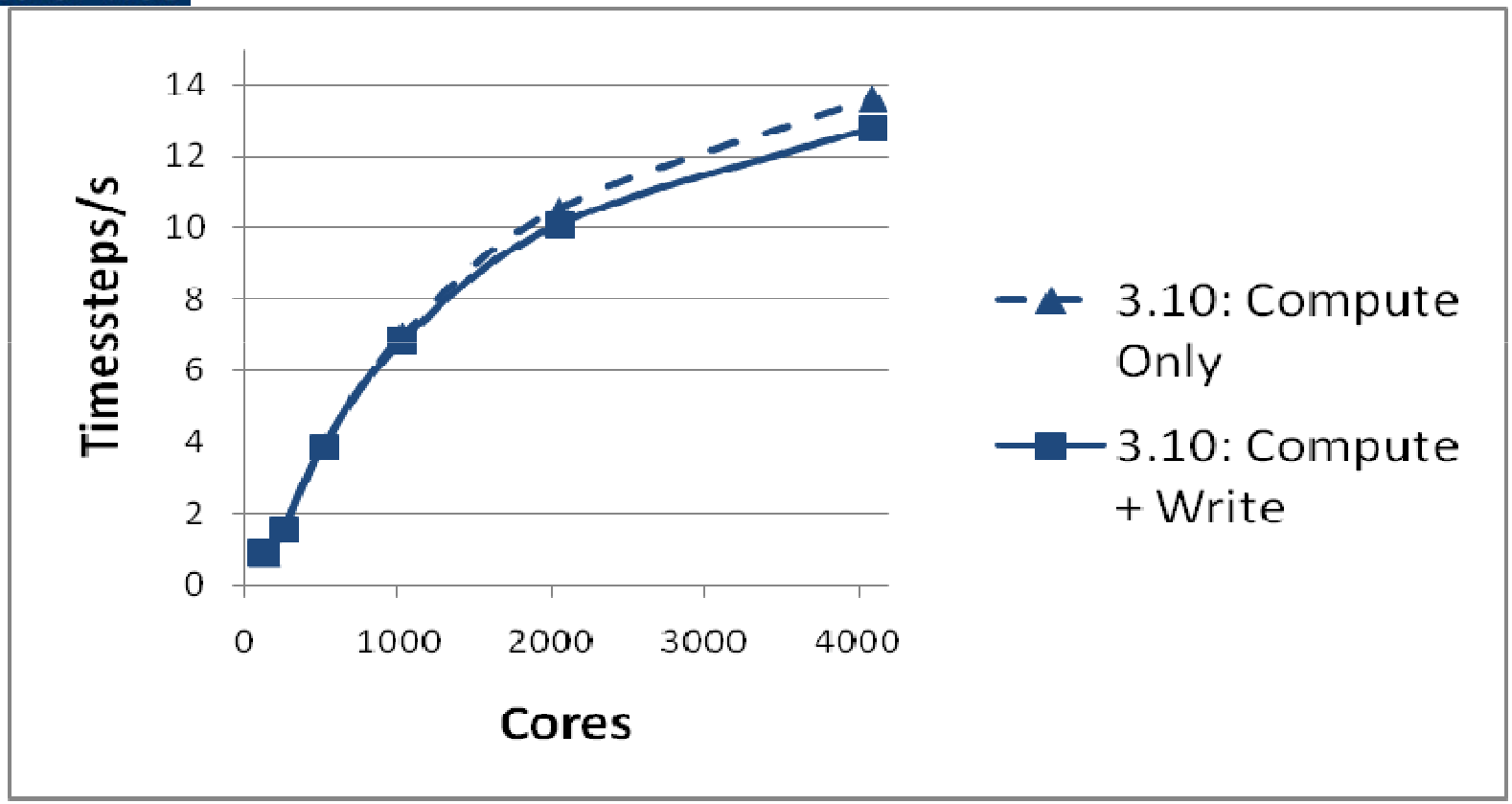

# Performance for 216000 Ions of NaCl




# Performance for 216000 Ions of NaCl

| | | 3.09 | 3.10 | 3.09 | 3.10 |
|---|---|---|---|---|---|
| Cores | I/O Procs | Time/s | Time/s | Mbyte/s | Mbyte/s |
| 32 | 32 | 143.30 | 1.27 | 0.44 | 49.78 |
| 64 | 64 | 48.99 | 0.49 | 1.29 | 128.46 |
| 128 | 128 | 39.59 | 0.53 | 1.59 | 118.11 |
| 256 | 128 | 68.08 | 0.43 | 0.93 | 147.71 |
| 512 | 256 | 113.97 | 1.33 | 0.55 | 47.60 |
| 1024 | 256 | 112.79 | 1.20 | 0.56 | 52.47 |
| 2048 | 512 | 135.97 | 0.95 | 0.46 | 66.39 |


# Performance For 1728000 Ions of NaCl




Maximum performance is 810 Mbyte/s


# Parallel Reading

Though not nearly as important as writing, reading can be an issue for large systems

In next release will be a parallel reading method

- Currently serial

Parallel method is

- A subset of the processors read in a batch
- Each scatters the atoms to the correct processors
- Repeat


# Parallel Reading For 216000 Ions of NaCl

| | | 3.10 | New | 3.10 | New |
|---|---|---|---|---|---|
| Cores | I/O Procs | Time/s | Time/s | Mbyte/s | Mbyte/s |
| 32 | 16 | 3.71 | 0.29 | 17.01 | 219.76 |
| 64 | 16 | 3.65 | 0.30 | 17.28 | 211.65 |
| 128 | 32 | 3.56 | 0.22 | 17.74 | 290.65 |
| 256 | 32 | 3.71 | 0.30 | 16.98 | 213.08 |
| 512 | 64 | 3.60 | 0.48 | 17.53 | 130.31 |
| 1024 | 64 | 3.64 | 0.71 | 17.32 | 88.96 |
| 2048 | 128 | 3.75 | 1.28 | 16.84 | 49.31 |


# NetCDF

Also there is a initial NetCDF implementation

- Files can get very big – 100s Gbytes
- "Binary" but portable
- NetCDF files roughly 1/3 size of the formatted files
- Current performance very poor
  - Needs more investigation
    - Suggestions welcome!


# NetCDF Performance – Writing 21600 Ions

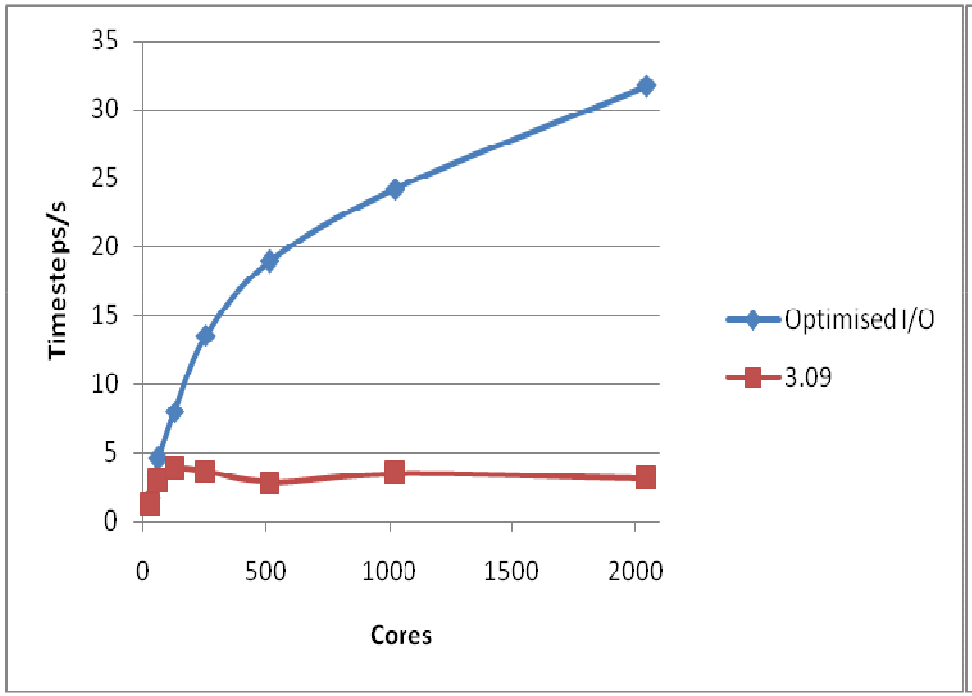
| | | 3.10 | 3.10 | NetCDF | NetCDF |
|---|---|---|---|---|---|
| Cores | I/O Procs | Time/s | Mbyte/s | Time/s | Mbyte/s |
| 32 | 32 | 1.27 | 49.78 | 4.77 | 13.22 |
| 64 | 64 | 0.49 | 128.46 | 8.63 | 7.31 |
| 128 | 128 | 0.53 | 118.11 | 13.81 | 4.57 |
| 256 | 128 | 0.43 | 147.71 | 27.24 | 2.32 |
| 512 | 256 | 1.33 | 47.60 | 40.57 | 1.55 |
| 1024 | 256 | 1.20 | 52.47 | 67.55 | 0.93 |
| 2048 | 512 | 0.95 | 66.39 | 147.47 | 0.43 |

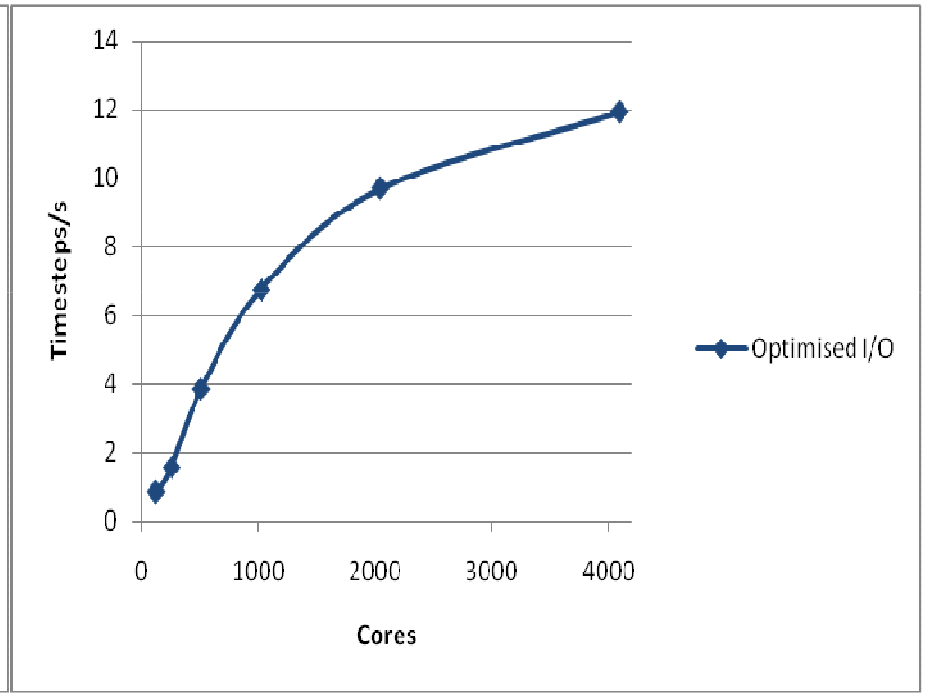

# Overall Performance

The most important measure of the performance of the whole code is:

- Is it fast enough for the scientist to do science?


# Is It Fast Enough




**YES!**


# Conclusion

Extensive reorganization of the data may be required to get the best out of the I/O subsystem

This may well be beneficial because I/O is so slow compared to compute or communication

But most importantly: Optimisation of the I/O now allows the scientist to perform real science more quickly on many more processors