Parallel Algorithms on a cluster of PCs

Ian Bush

Computational Science & Engineering Department

Daresbury Laboratory

I.J.Bush@dl.ac.uk

With Thanks to W.Smith at DL

Case Studies

- In this lecture I want to give a couple of examples of code I have worked upon in recent years

- They are
  - CRYSTAL - *ab initio* electronic structure of solids
  - DL_POLY - Classical MD

CRYSTAL

- Electronic structure and related properties of periodic solids
- All electron, local Gaussian basis set, DFT and Hartree-Fock
- Under continuous development since 1974
- Distributed to over 500 sites world wide
- Developed jointly by Daresbury and Turin

CRYSTAL - Functionality

- **Basis Set**
- LCAO - Gaussians
- All electron or pseudopotential
- **Hamiltonian**
- Hartree-Fock (UHF, RHF)
- DFT (LSDA, GGA)
- Hybrid funcionals
- **Techniques**
- Replicated data parallel
- Distributed data parallel
- Direct –SCF
- Geometry optimisation
- **Visualisation**
- Cerius^2 interface
- AVS GUI (DLV)

Properties

- Energy
- Structure
- Vibrations (phonons)
- Elastic tensor
- Ferroelectric polarisation
- Piezoelectric constants
- X-ray structure factors
- Density of States / Bands
- Charge/Spin Densities
- Magnetic Coupling
- Electrostatics (V, E, EFG classical)
- Fermi contact (NMR)
- EMD (Compton, e-2e)

CRYSTAL – Parallel Implementations

- Pcrystal
  - Replicated data
  - Good for medium to large problems on small to medium processor counts
- MPPcrystal
  - Distributed data
  - Good for large problems on large processor counts

CRYSTAL – basic algorithm

- $H^R = P^R \,.\, I^R$
  $I^R \Leftarrow$ sum of independent integrals
- $H_k \Leftarrow Q_k^T H^R Q_k$
- $H_k \psi_k = \varepsilon_k \psi_k$
  - **Solve $H_k \Rightarrow \{\varepsilon_k, \psi_k\}$**
- $P^R \Leftarrow | \psi_k |^2$
- Repeat until converged

F.T. and matrix multiply

Diagonalization - each k point independent

Gather & condense

Pcrystal - Implementation

- Standard compliant
  - Fortran 90
  - MPI for message passing
- Replicated data
  - Each processor has a complete copy of all the matrices used in the linear algebra
  - Makes implementation very simple

Pcrystal – Parallel Integrals

- Coulomb, Exchange and DFT terms all involve many independent tasks:
  - **Coulomb/Exchange have to evaluate integrals of the form <$\varphi_i\varphi_j$||$\varphi_k\varphi_l$> for all of i, j, k, l**
    - *Each integral independent*
    - *So give a subset to each of the processors*
    - *But requires more or less random access to H^R and P^R*
      - Bad for message passing - replicate
  - **DFT terms are a numerical integration over a grid**
    - *Each point of the grid independent*
    - *So give a subset of the grid to each processor*
- Almost perfectly parallel !
  - **Only global sum at end required – v. few comms**
  - **Limit on scaling is load imbalance**

Pcrystal – Linear Algebra

- Each k point (and spin) independent
- So each processor performs the linear algebra for a subset of the k points that the job requires
  - **Again very few comms so potentially good scaling, but …**
  - **Potential load imbalance**
    - *Complex v. real k points*
  - **Number of k points limits the number of processors that can be exploited**
    - *What if only a Γ point only calculation ?*
  - **Limit on size of job that can be performed does not scale with number of processors**

Pcrystal – Changes to The Input

(except think about direct)

Pcrystal - Summary

- In general scales very well provided the number of processors ≤ number of k points
  - Will gain something due to integrals
  - But large jobs in general require few k points
- The limit on the size of job is given by the memory required to store the linear algebra matrices for one k point
  - More processors do not mean larger jobs can be run

MPP Crystal - Implementation

- Uses common standards
  - Fortran 90
  - MPI for message passing
  - ScaLAPACK 1.7 (Dongarra *et al.*) for linear algebra on distributed matrices
    - *www.netlib.org/scalapack/scalapack_home.html*
  - Home grown BFG Jacobi diagonalizer
    - *www.cse.clrc.ac.uk/arc/bfg.shtml*
    - *Scales better and less memory hungry than ScaLAPACK*
- Distributed data
  - Each processor hold only a part of each of the matrices used in the linear algebra
  - More complex to implement

MPPcrystal – Parallel Integrals

- More or less as Pcrystal with some memory saving tricks
  - **Works well so why reinvent the wheel ?**
  - **However requires replicated H^R,P^R**
    - *Ultimate limit on size of job*
    - *However limit is less stringent than for Pcrystal because these are stored in sparse format*

MPPcrystal - Linear Algebra (1)

- All matrices distributed
  - **More procs means more memory so larger jobs**
- Mostly use ScaLAPACK for e.g.
  - **Choleski decomposition**
  - **Matrix matrix multiplies**
  - **Linear equation solves**
- However for diag use own BFG package
  - **Based on Jacobi which can better exploit the sparse nature of the matrix**
  - **Scales with processor number better than that provided by ScaLAPACK**
  - **Requires less memory than ScaLAPACK**
  - **But slower on first 1-2 cycles**

MPPcrystal – Linear algebra (2)

- As each processor only holds a part of the matrix comms are required to perform the linear algebra, unlike for Pcrystal
- However N^3 operations but only N^2 data to communicate
  - **Scaling gets better for larger systems**
  - **Very rough rule of thumb – if N basis functions can exploit up to around N/20 processors**
- Further the number of processors that can be exploited is NOT limited by the number of k points
  - **Great for large Γ point only calculations !**

MPPcrystal – other issues

- By default runs direct
  - **100s or processors writing to/reading from one disk not a good idea !**
- Most but not all of CRYSTAL implemented
  - **Will fail quickly and cleanly if requested feature not implemented**
  - **Perhaps the most important is symmetry adaption of the diag**
    - *For large high symmetry systems Pcrystal may be more effective*
- Too small a job on too many procs will fail
  - **In general not an issue**

MPPcrystal – Changes to Your Input

```
TEST08 - SILICON BULK: STO-3G
CRYSTAL
0 0 0
227
5.42
1
14  .125  .125   .125
END
14 3
1 0 3  2.  0.
1 1 3  8.  0.
1 1 3  4.  0.
99 0
END
END
8 4 8
MPP
END
```

MPPcrystal - summary

- For large systems can scale well, but not so good for small to medium size ones.
- Size of linear algebra matrices is, at present, not an issue given enough processors.
- Memory limitation is from the replicated H^R,P^R NOT the linear algebra matrices. As the former are stored in a sparse format they tend to be much smaller than the latter.

Pcrystal and MPPcrystal

- Pcrystal
  - Few comms means scales very well
  - However scaling limited by number of k points
  - Memory usage in linear algebra limits size of system that may be studied
  - Load imbalance in linear algebra may be an issue
- MPPcrystal
  - More comms but scales well for large system
  - Scaling not limited by number of k points
  - Distributing the matrices allows larger systems to be studied, especially on large number of processors

MPPcrystal – an example

- I will illustrate the behaviour of MPPcrystal with some calculations on a small protein, Crambin.

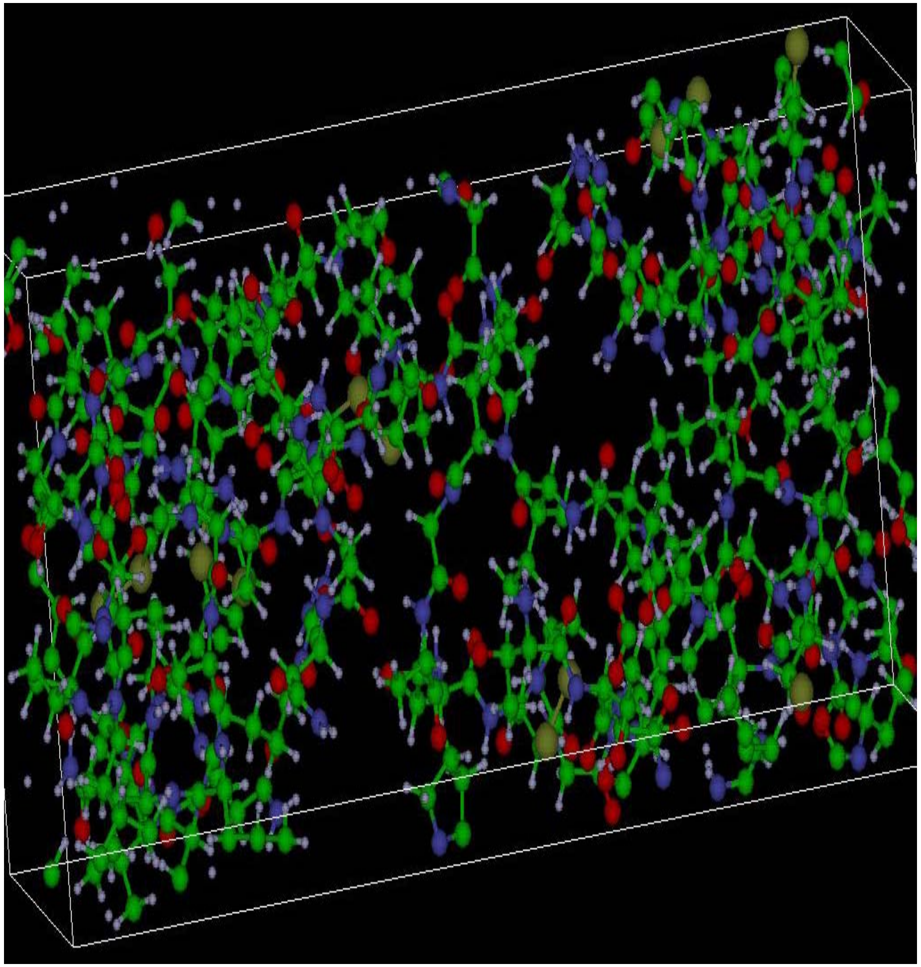
- I will also give an indication of what we are trying to do with MPPcrystal

Why Crambin (the official version)

- Small protein (46 residues)
- Crystal structure characterized to very high precision by XRD studies (0.52 Å)
- PDB entry (1EJG) includes hydrogens (this is unusual)

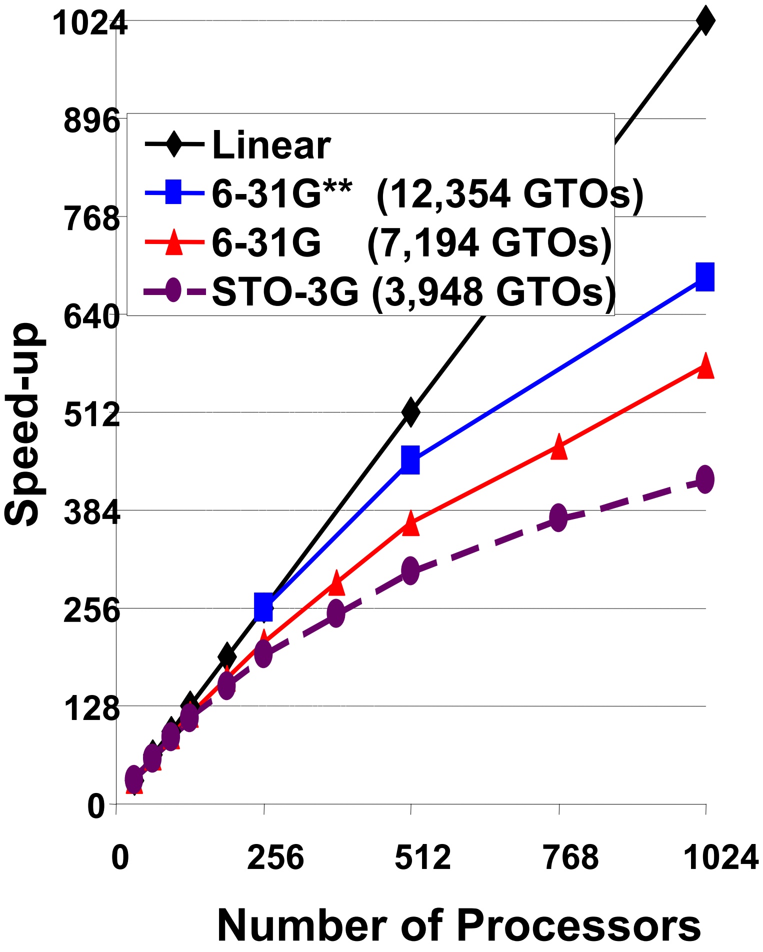
Crambin - The Crystal

- 2 Chains in unit Cell
- 1284 Atoms
- Initial studies using STO3G (3948 basis functions)
- Upped to 6-31G * * (12354 functions)
- All calculations Hartree-Fock



Parallel Performance

- Fit measured data to Amdahl's law to obtain estimate of speed up
- Increasing the basis set size increases the scalability
- About 700 speed up on 1024 processors for 6-31G * *
- Takes about 3 hours instead of about 3 months
- 99.95% parallel

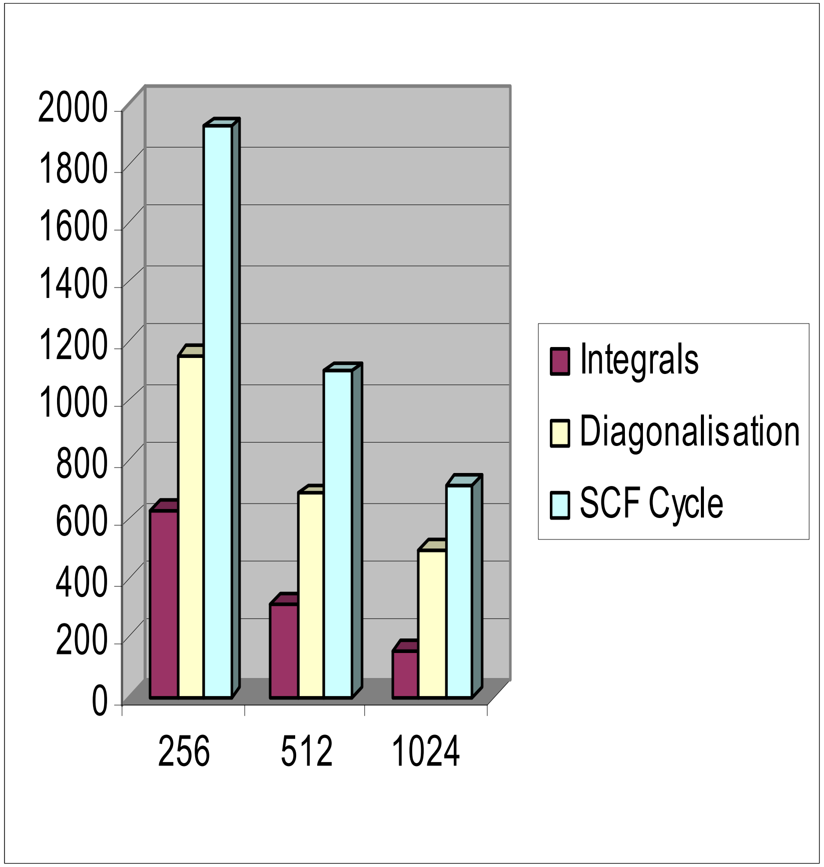


6-31G * * Parallel Performance

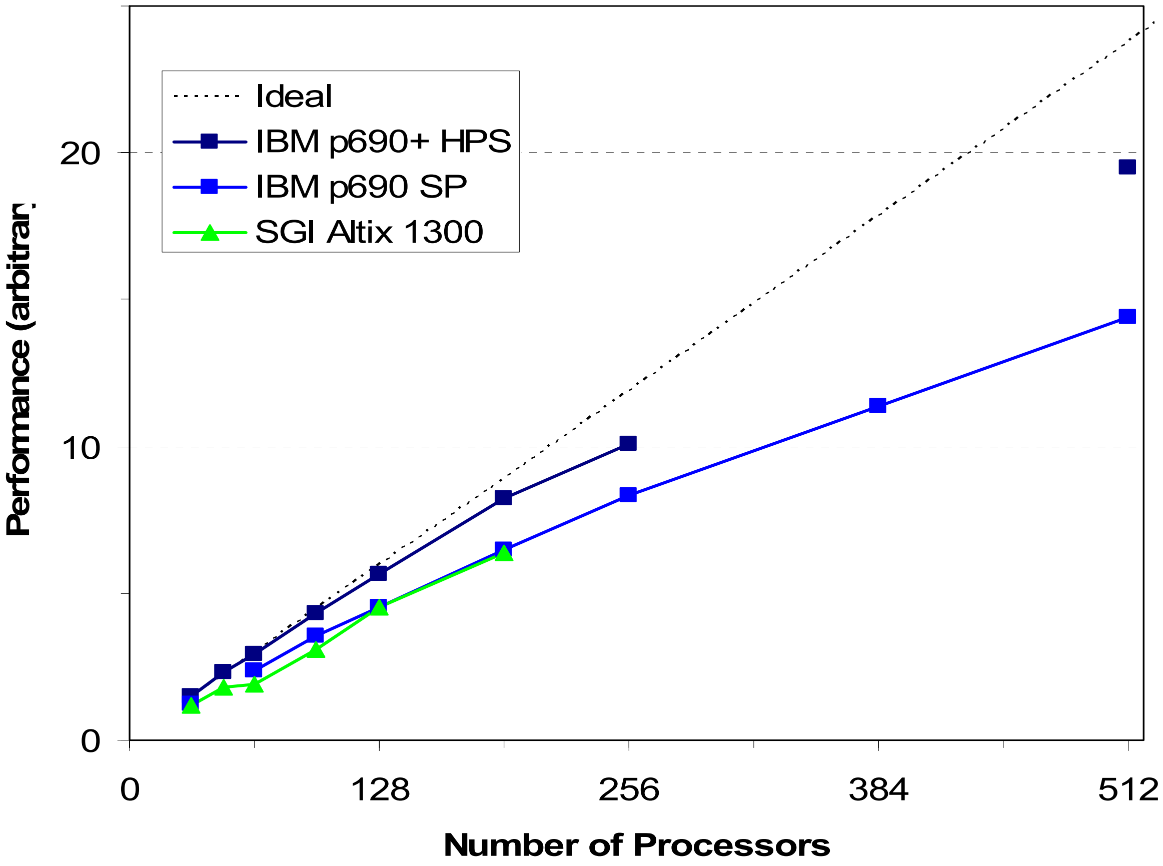
- The integrals scale almost perfectly

- The diag is 3.1 times quicker on 1024 compared to 256 for the whole run

- Overall good scaling exhibited



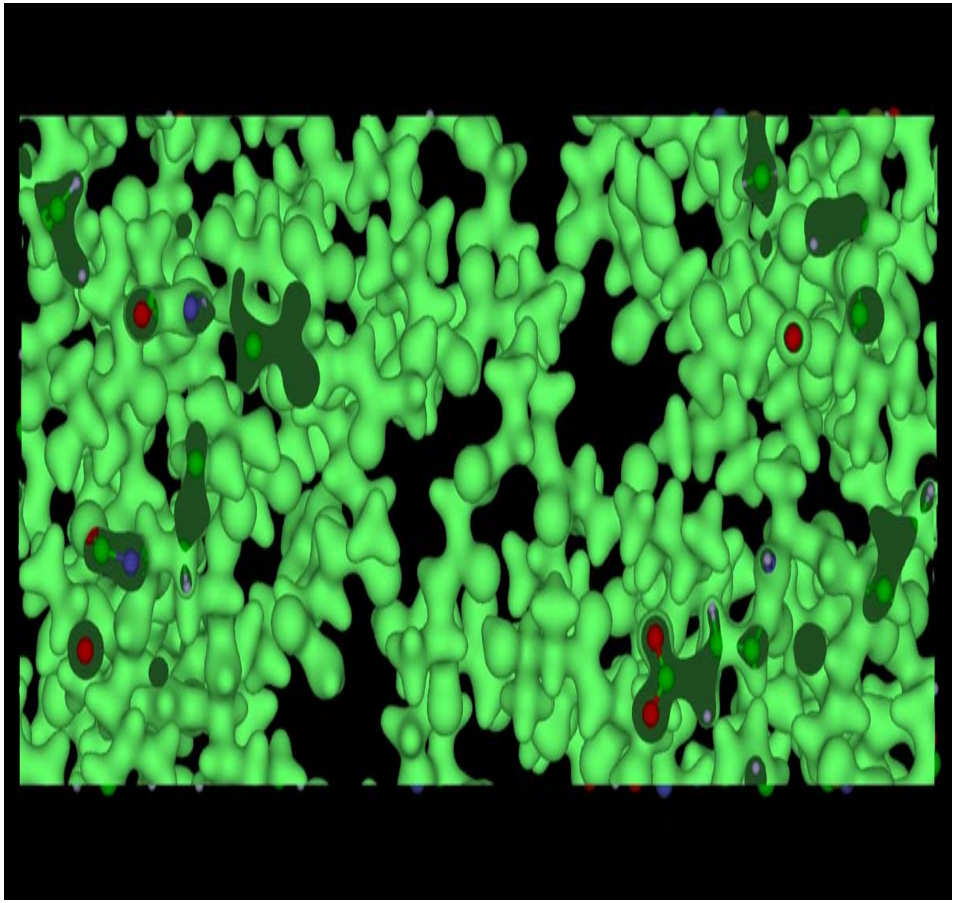
6-31G Latest data



Results - Charge Density

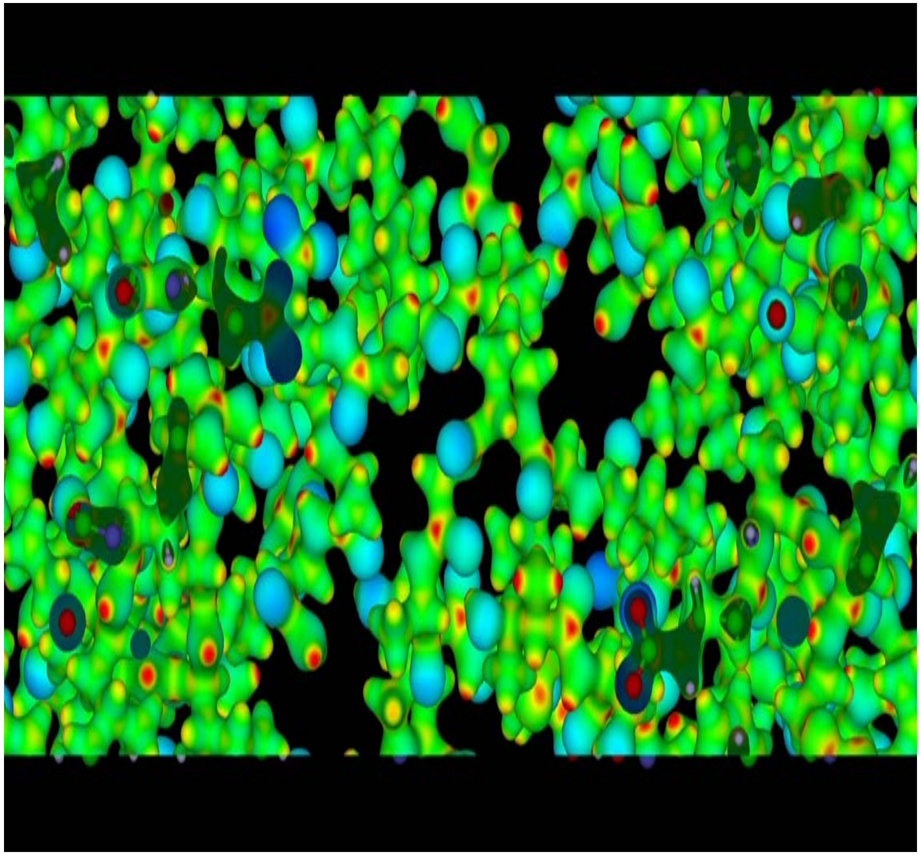
- Isosurface of the charge density at 0.1Å resolution

- Can be compared with SR results



Results – Electrostatic Potential

- Charge density isosurface coloured according to potential
- Useful to determine possible chemically active groups

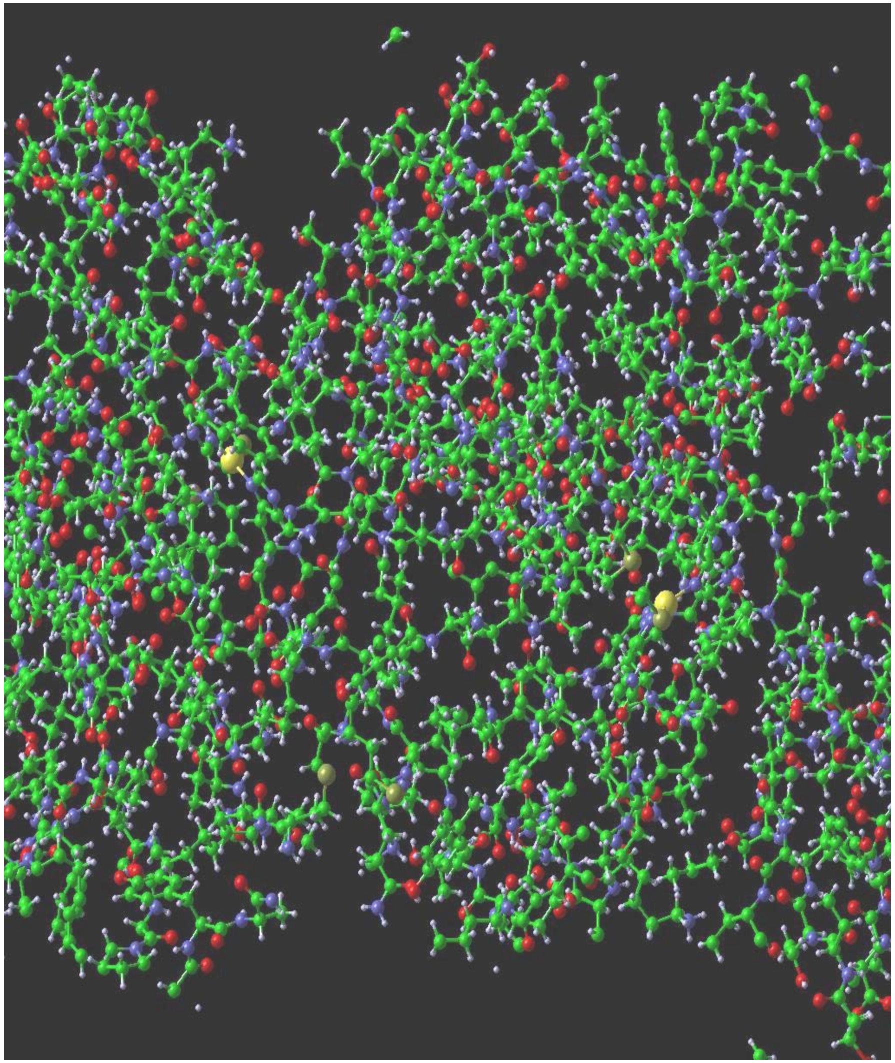


Rusticyanin

- Rusticyanin, a blue copper protein, has 6284 atoms and is involved in redox processes

- We have started calculations using over 30000 basis functions

- In collaboration with S.Hasnain (DL) we want to calculate redox potentials for rusticyanin and associated mutants. Rusti has a large potential, 680mV



CRYSTAL Summary

- Crystal can use to parallelization strategies
  - **Pcrystal uses replicated data**
    - *Good for medium to large problems*
    - *Memory limits size of problem that may be addressed*
    - *Scales well up to number of k points*
    - *The one you'll use most often – it's the DL day to day workhorse*
  - **MPPcrystal uses distributed data**
    - *Needs large problems to perform well*
    - *Memory limitations much less stringent than Pcrystal*
    - *For a big enough problem can scale very well*

DL_POLY Background

- General purpose parallel MD code
- Developed at Daresbury Laboratory for CCP5 1994-today
- Available free of charge (under licence) to University researchers world-wide

DL_POLY Versions

- DL_POLY_2
  - Replicated Data, up to 50,000 atoms
  - Full force field and molecular description

- DL_POLY_3
  - Domain Decomposition, up to 1,000,000 atoms+
  - Full force field but no rigid body description.

The DL_POLY Force Field

$$V(\vec{r}_1,\vec{r}_2,\ldots\ldots\vec{r}_N)=\sum_{i,j}^{N'}U_{\mathrm{pair}}\left(|\vec{r}_i-\vec{r}_j|\right)+\frac{1}{4\pi\varepsilon}\sum_{i,j}^{N'}\frac{q_iq_j}{|\vec{r}_i-\vec{r}_j|}+$$

$$\sum_{i,j,k}^{N'}U_{\mathrm{3-body}}\left(\vec{r}_i,\vec{r}_j,\vec{r}_k\right)+\sum_{i,j,k,n}^{N'}U_{\mathrm{4-body}}\left(\vec{r}_i,\vec{r}_j,\vec{r}_k,\vec{r}_n\right)+\varepsilon_{\mathrm{metal}}\left(\sum_{i,j}^{N'}\left(\frac{\alpha}{r_{ij}}\right)^{n}-C\sum_{i=1}^{N}\rho_i^{1/2}\right)+$$

$$\sum_{i_{\mathrm{bond}}}^{N_{\mathrm{bond}}}U_{\mathrm{bond}}\left(i_{\mathrm{bond}},\vec{r}_a,\vec{r}_b\right)+\sum_{i_{\mathrm{angle}}}^{N_{\mathrm{angle}}}U_{\mathrm{angle}}\left(i_{\mathrm{angle}},\vec{r}_a,\vec{r}_b,\vec{r}_c\right)+\sum_{i_{\mathrm{dihed}}}^{N_{\mathrm{dihed}}}U_{\mathrm{dihed}}\left(i_{\mathrm{dihed}},\vec{r}_a,\vec{r}_b,\vec{r}_c,\vec{r}_d\right)+$$

$$\sum_{i_{\mathrm{invers}}}^{N_{\mathrm{invers}}}U_{\mathrm{invers}}\left(i_{\mathrm{invers}},\vec{r}_a,\vec{r}_b,\vec{r}_c,\vec{r}_d\right)+\sum_{i=1}^{N}\Phi_{\mathrm{external}}\left(\vec{r}_i\right)$$

DL_POLY Force Field

- Intermolecular forces
  - All common van de Waals potentials
  - Sutton Chen many-body potential
  - 3-body angle forces (SiO_2)
  - 4-body inversion forces (BO_3)
- Intramolecular forces
  - Bonds, angle, dihedrals, inversions

DL_POLY Force Field

- Coulombic forces
  - SPME (3D), Adiabatic shell model, Reaction field, Bare Coulombic, Shifted Coulombic
- Externally applied field
  - Walled cells,electric field,shear field, etc

Boundary Conditions

- None (e.g. isolated macromolecules)
- Cubic periodic boundaries
- Orthorhombic periodic boundaries
- Parallelepiped periodic boundaries
- Truncated octahedral periodic boundaries
- Rhombic dodecahedral periodic boundaries
- Slabs (i.e. x,y periodic, z nonperiodic)

Algorithms and Ensembles

Algorithms

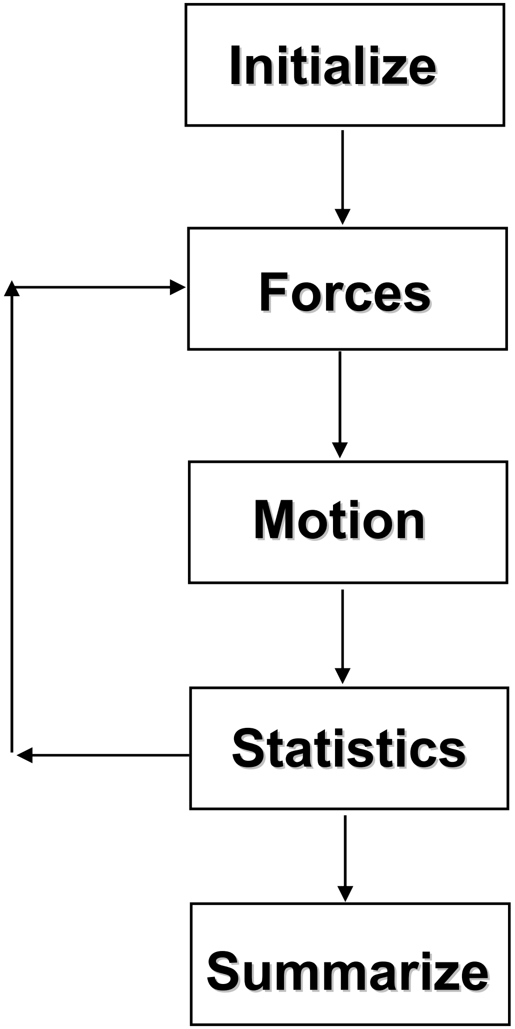
- Verlet leapfrog
- RD-SHAKE
- Euler-Quaternion*
- QSHAKE*
- [All combinations]

* **Not in DL_POLY_3**

Ensembles

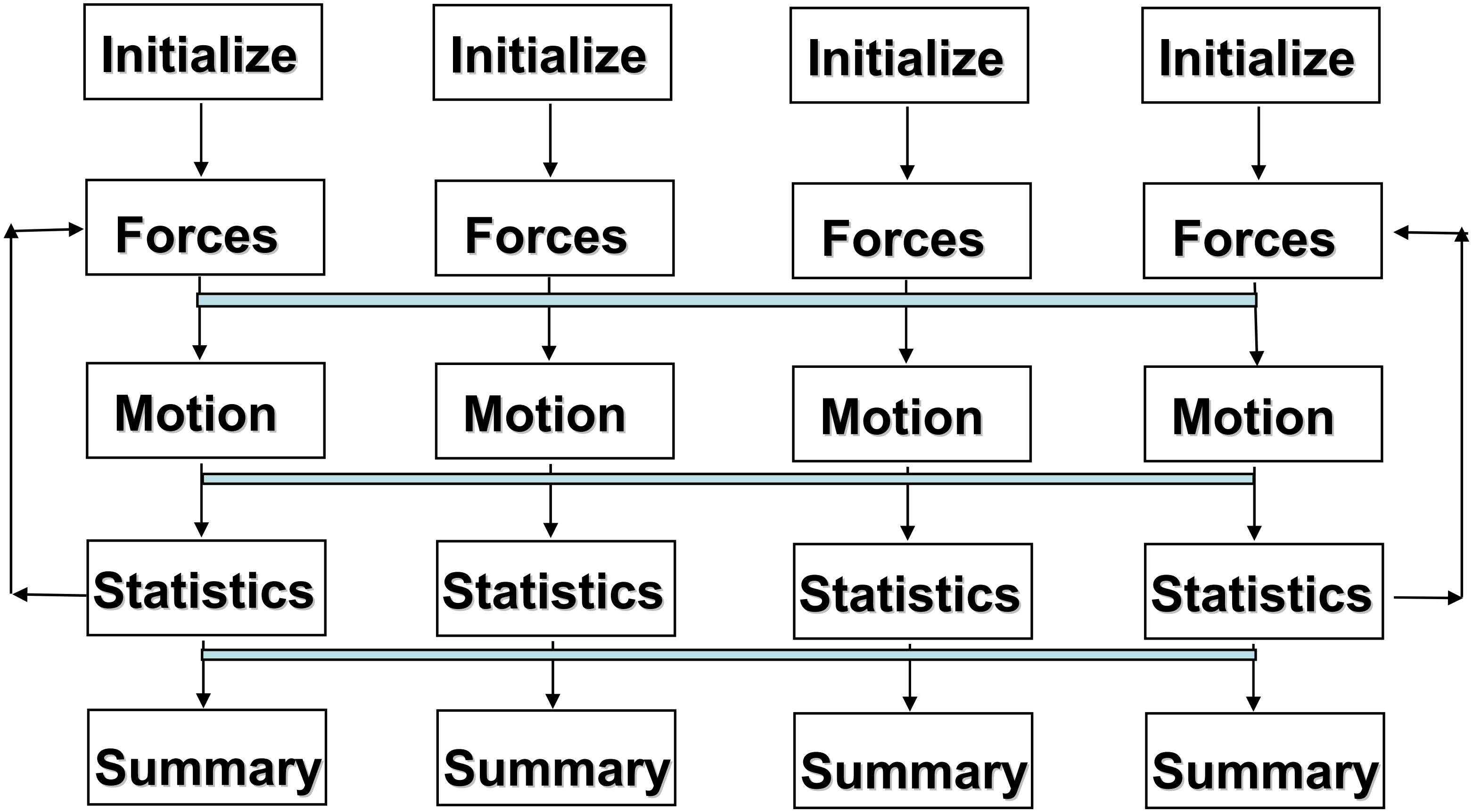
- NVE
- Berendsen NVT
- Hoover NVT
- Evans NVT
- Berendsen NPT
- Hoover NPT
- Berendsen NσT
- Hoover NσT

Key Stages in MD Simulation



- *Set up initial system*
- *Calculate atomic forces*
- *Calculate atomic motion*
- *Calculate physical properties*
- *Repeat !*
- *Produce final summary*

Replicated Data



Replicated Data MD Algorithm

Features:

- **Each node has copy of all atomic coordinates (Ri,Vi,Fi)**
- **Force calculations shared equally between nodes (i.e. up to N(N-1)/2P pair forces per node).**
  - *Use neighbour list*
- **Atomic forces summed globally over all nodes**
- **Motion integrated for all or some atoms on each node**
- **Updated atom positions circulated to all nodes**

Replicated Data MD Algorithm

Advantages:

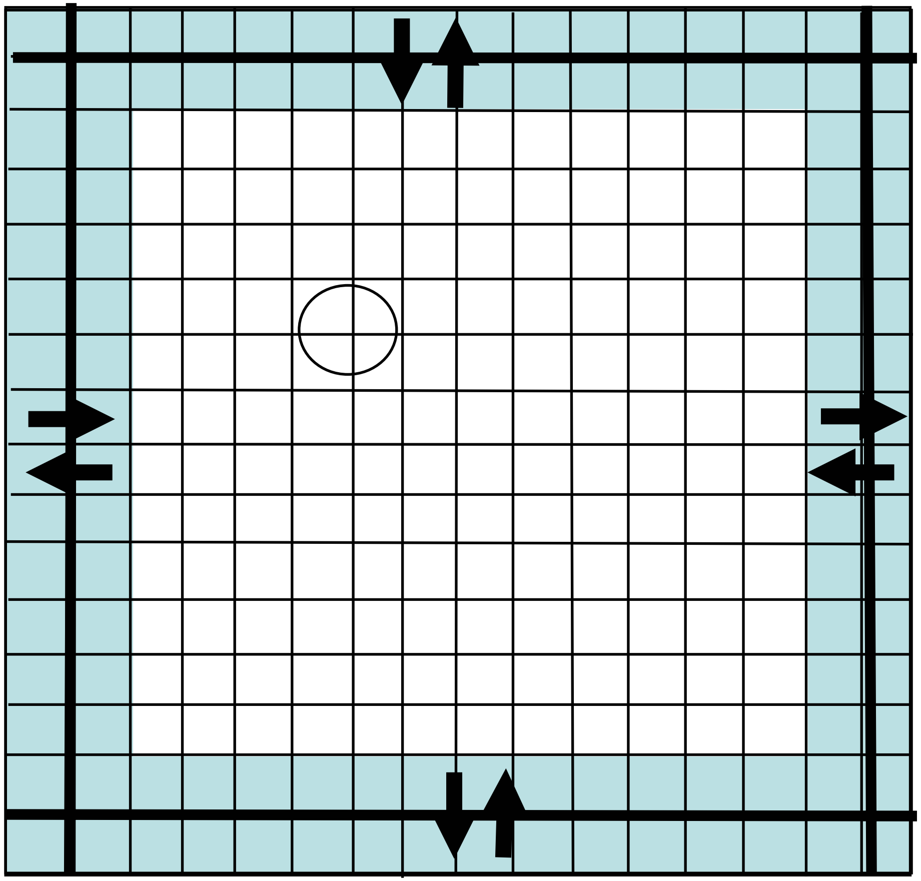
- Simple to implement
- Good load balancing
- Suitable for complex force fields
- Dynamic load balancing possible

Replicated Data MD Algorithm

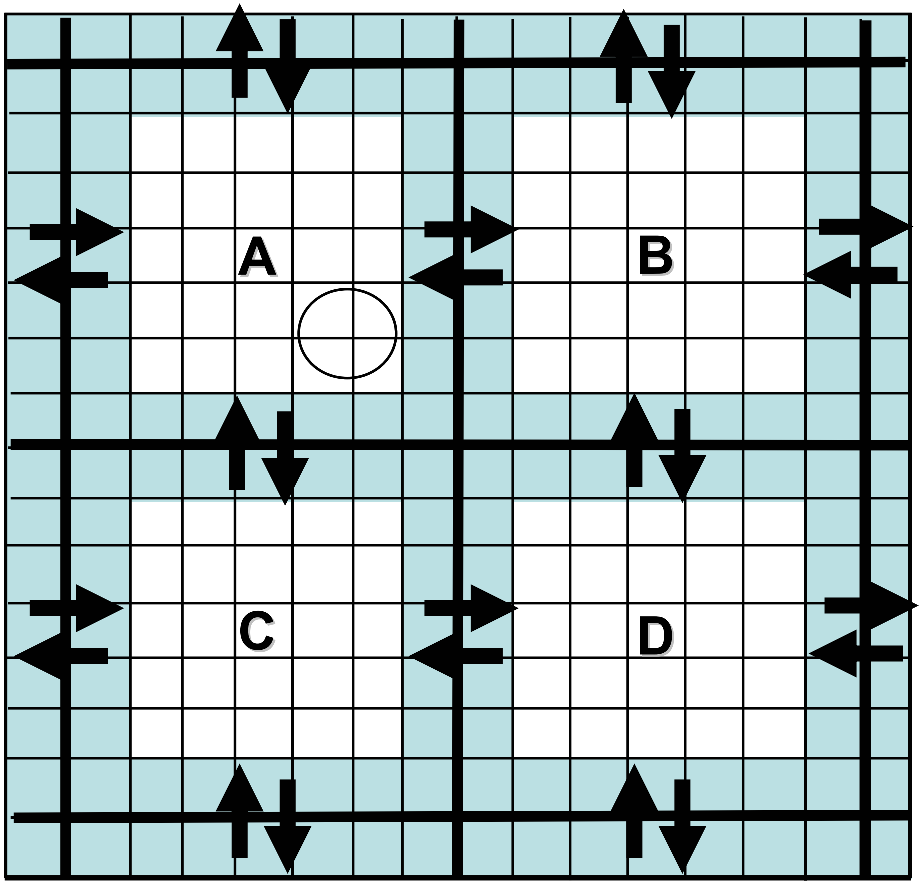
Disadvantages:

- High communication overhead
- Sub-optimal type 2 scaling
- Large memory requirement
- Unsuitable for massive parallelism

DL_POLY 3 - Link Cell Algorithm



DL_POLY 3 Domain Decomposition MD



DL_POLY 3 - Domain Decomposition MD

Features:

- Short range potential cut off ($r_{cut} << L_{cell}$)
- Spatial decomposition of atoms into domains
- Map domains onto processors
- Use link cells in each domain
- Pass border link cells to adjacent processors
- Calculate forces, solve equations of motion
- Re-allocate atoms leaving domains

DL_POLY 3 - Domain Decomposition MD

Advantages:

- Good load balancing
- Ideal for huge systems
- Simple communication structure
- Fully distributed memory requirement
- Dynamic load balancing possible
- Good but not perfect scaling
    - Latency effects

Domain Decomposition MD

Disadvantages

- Requires short potential cut off
- Complex force fields tricky
- Not Suitable for small systems
  - **~< 50,000 atoms**

Parallel Force Calculation

Short Range Non-Bonded Forces:

- DL_POLY 2
  - Have complete list of all atoms
  - User Verlet neighbour list
  - So know how many forces we need to calculate
  - So Simply spilt them up amongst the processors

- DL_POLY 3
  - Once data exchanged with neighbour procs have all the data we need to calculate the forces on my atoms
  - So calculate them !

Also bond forces and constraint forces – will not cover here

The Ewald Summation

$$U_c = \frac{1}{2V\varepsilon_o} \sum_{\vec{k} \neq \vec{0}}^{\infty} \frac{\exp(-k^2 / 4\alpha^2)}{k^2} \left| \sum_{j=1}^{N} q_j \exp\left(-i\vec{k} \cdot r_j\right) \right|^2 +$$

$$\frac{1}{4\pi\varepsilon_o} \sum_{\vec{R}_\ell = \vec{0}}^{\infty} \sum_{n<j}^{N} \frac{q_n q_j}{\left|\vec{R}_\ell + \vec{r}_{jn}\right|} erfc\left(\alpha\left|\vec{R}_\ell + \vec{r}_{jn}\right|\right) -$$

$$\frac{\alpha}{4\pi^{3/2}\varepsilon_o} \sum_{j=1}^{N} q_j^2$$

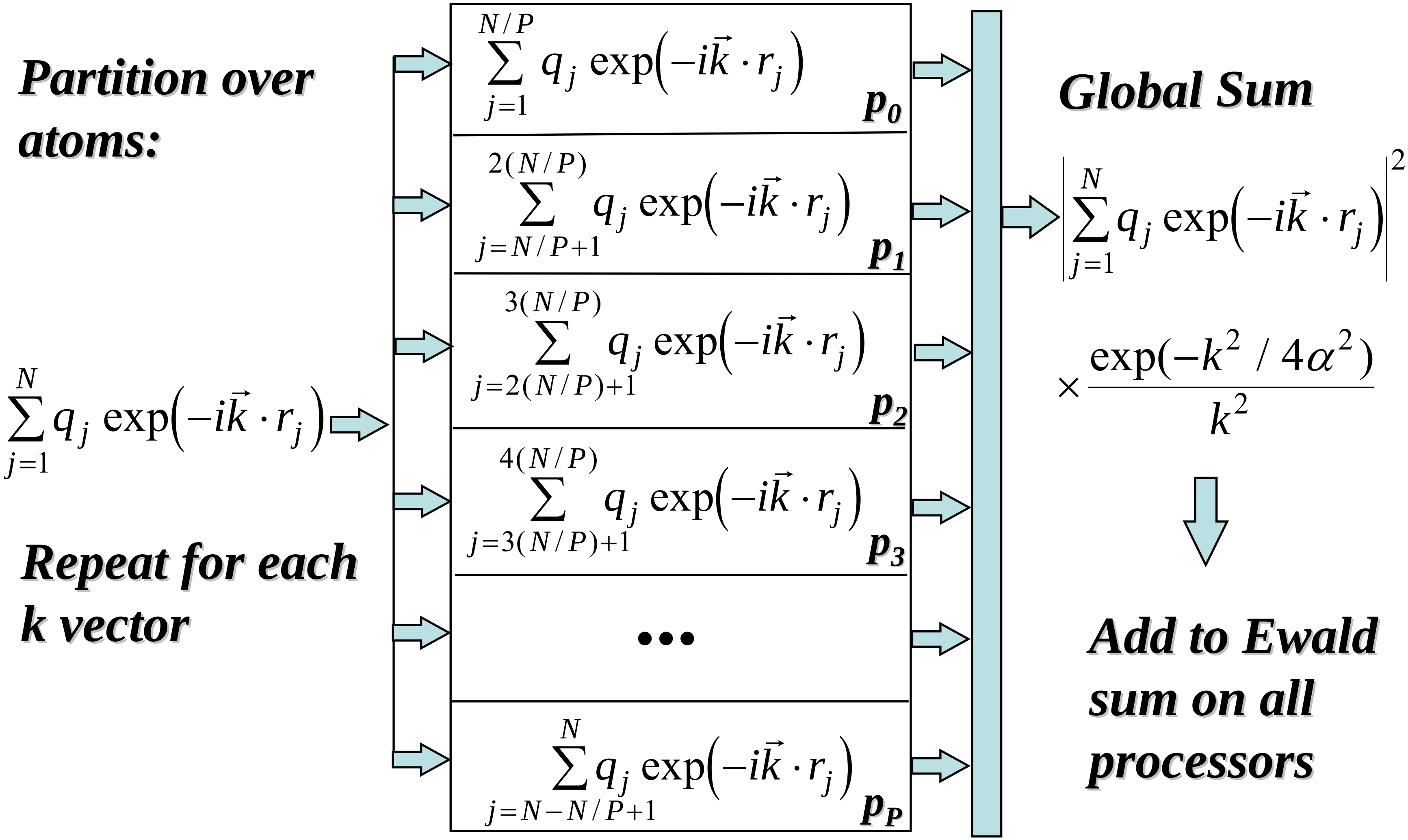
with: $\vec{k} = \frac{2\pi}{V^{1/3}} (\ell, m, n)^{\perp}$

DL_POLY 2 - Parallel Ewald Summation

- Self interaction correction - as is.

- Real Space terms:
  - Handle using parallel Verlet neighbour list
  - A short range force so handle as appropriate for DL_POLY 2 or 3

- Reciprocal Space Terms:
  - Distribute over atoms



Partition over atoms:

$$\sum_{j=1}^{N} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

Repeat for each k vector

$$\sum_{j=1}^{N/P} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

p_0

$$\sum_{j=N/P+1}^{2(N/P)} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

p_1

$$\sum_{j=2(N/P)+1}^{3(N/P)} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

p_2

$$\sum_{j=3(N/P)+1}^{4(N/P)} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

p_3

•••

$$\sum_{j=N-N/P+1}^{N} q_j \exp\left(-i\vec{k}\cdot r_j\right)$$

p_P

Global Sum

$$\left|\sum_{j=1}^{N} q_j \exp\left(-i\vec{k}\cdot r_j\right)\right|^2$$

$$\times \frac{\exp(-k^2/4\alpha^2)}{k^2}$$

Add to Ewald sum on all processors

Smoothed Particle-Mesh Ewald

Ref: Essmann *et al*., J. Chem. Phys. (1995) **103** 8577

The crucial part of the SPME method is the conversion of the Reciprocal Space component of the Ewald sum into a form suitable for Fast Fourier Transforms (FFT). Thus:

$$U_{recip} = \frac{1}{2V\varepsilon_o} \sum_{\vec{k}\neq\vec{0}}^{\vec{\infty}} \frac{\exp(-k^2/4\alpha^2)}{k^2} \left| \sum_{j=1}^{N} q_j \exp\left(-i\vec{k}\cdot r_j\right) \right|^2$$

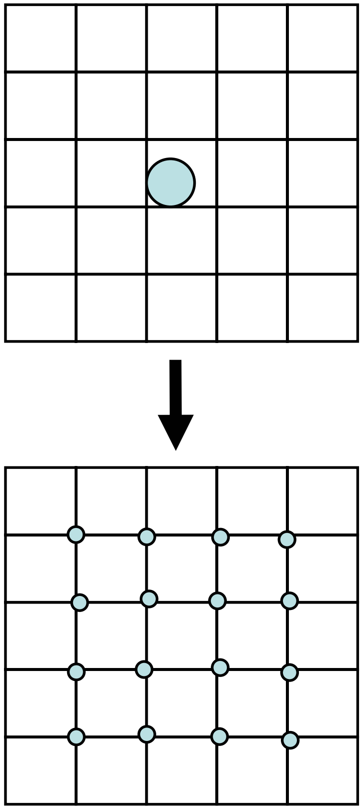
becomes:

$$U_{recip} = \frac{1}{2V\varepsilon_o} \sum_{k_1,k_2,k_3} G^T(k_1,k_2,k_3) Q(k_1,k_2,k_3)$$

where G and Q are 3D grid arrays (see later)

SPME: Spline Scheme

Central idea - share discrete charges on 3D grid:



Cardinal B-Splines $M_n(u)$ - in 1D:

$$\exp\left(2\pi i u_j k / L\right) \approx b(k) \sum_{\ell=-\infty}^{\infty} M_n(u_j - \ell) \exp\left(2\pi i k\ell / K\right)$$

$$b(k) = \exp(2\pi i(n-1)k / K)\left[\sum_{\ell=0}^{n-2} M_n(\ell+1) \exp\left(2\pi i k\ell / K\right)\right]^{-1}$$

$$M_n(u) = \frac{1}{(n-1)!}\sum_{k=0}^{n}(-1)^k \frac{n!}{k!(n-k)!}\max(u-k,0)^{n-1}$$

$$M_n(u) = \frac{u}{n-1}M_{n-1}(u) + \frac{n-u}{n-1}M_{n-1}(u-1)$$

Recursion relation

SPME: Building the Arrays

$$Q(\ell_1,\ell_2,\ell_3) =$$
$$\sum_{j=1}^{N} q_j \sum_{n_1,n_2,n_3} M_n(u_{1j}-\ell_1-n_1K_1)M_n(u_{2j}-\ell_2-n_2K_2)M_n(u_{3j}-\ell_3-n_3K_3)$$

Is the charge array and $Q^T(k_1,k_2,k_3)$ its discrete Fourier transform.

$G^T(k_1,k_2,k_3)$ is the discrete Fourier Transform of the function:

$$G(k_1,k_2,k_3) = \frac{\exp(-k^2/4\alpha^2)}{k^2} B(k_1,k_2,k_3)(Q^T(k_1,k_2,k_3))^*$$

with $$B(k_1,k_2,k_3) = |b_1(k_1)|^2 |b_2(k_2)|^2 |b_3(k_3)|^2$$

SPME: Comments

- SPME is generally faster then conventional Ewald sum in most applications. Algorithm scales as O(N*log*N)
- In DL_POLY_2 the FFT array is built in pieces on each processor and made whole by a global sum for the FFT operation
- In DL_POLY_3 the FFT array is built in pieces on each processor and kept that way for the distributed FFT operation (DAFT)
- The DAFT FFT `hides' all the implicit communications

Parallel FFTs - The Basics

FFTs are

- Fast (!) - $O(V\log V)$ operations where V is the number of points in the grid
- Global operations - to perform a FFT you need all the points

This makes it difficult to write an efficient, good scaling FFT.

Traditional Parallel FFTs (1)

- Distribute the data by planes
- Each processor has a complete set of points in the x and y directions so can do those Fourier transforms
- Redistribute data so that a processor holds all the points in z
- Do the z transforms

Traditional Parallel FFTs (2)

- Allows efficient implementation of the serial FFTs (use a library routine)
- In practice for large enough 3D FFTs can scale reasonably
- **However** the distribution does not map onto DL_POLY 3's distribution - large amounts of data redistribution

DAFT(1)

- Takes data distributed as DLPOLY
- So do a distributed data FFT in the x direction
- Then the y
- And finally the z

DAFT(2)

- Disadvantage is that can not use the library routine for the 1D FFT (not quite true …)
- Scales quite well - e.g. on 512 procs, an 8x8x8 proc grid, a 1D FFT need only scale to 8 procs
- Totally avoids data redistribution

Traditional v. DAFT

- Traditional has faster serial speed as can use library routines
- DAFT avoids a lot of communication because it maps directly onto DLPOLYs distribution

In practice DAFT wins (on the few machines we have compared), and also the coding is simpler !

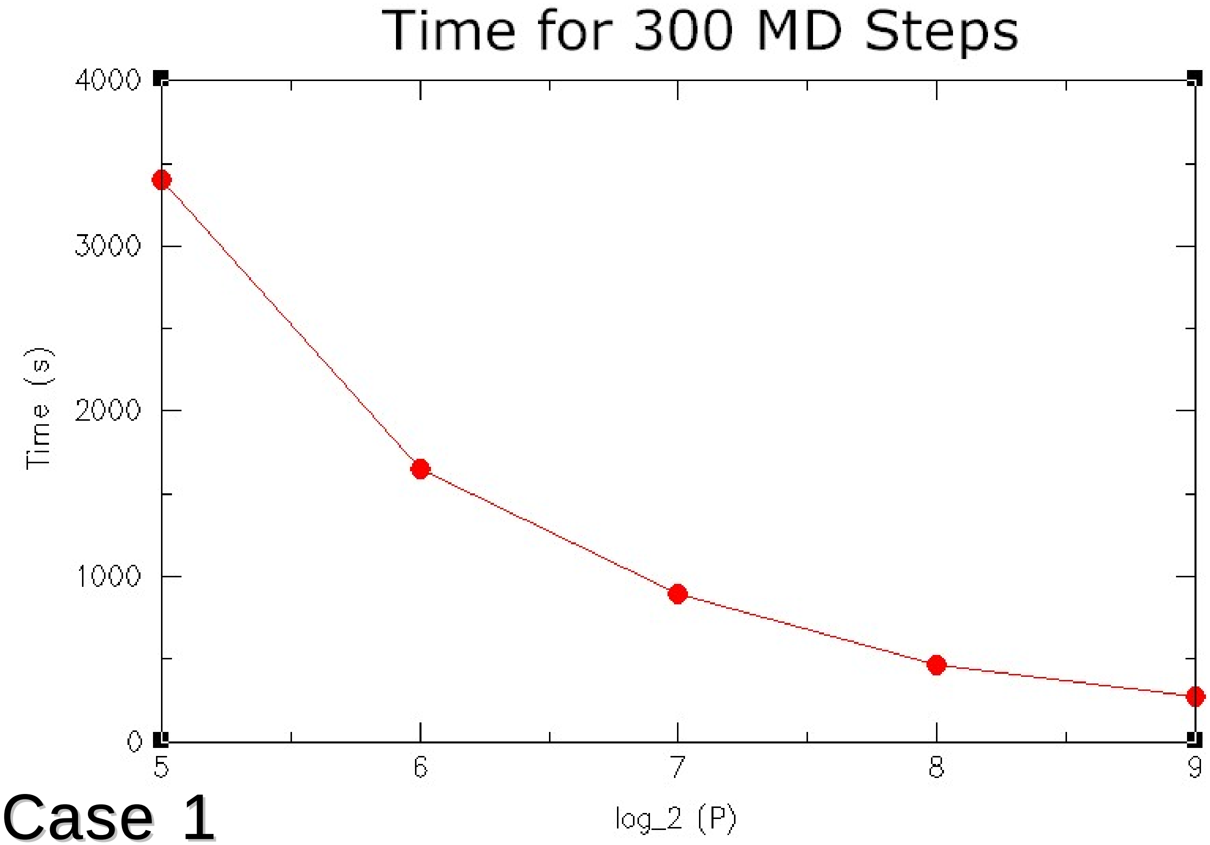
DAFT, DAFT prices !

- DAFT is a standard Fortran 90 module and is extremely portable
- If anybody wants a copy for their other codes please ask me !

DL_POLY_3 on HPCx

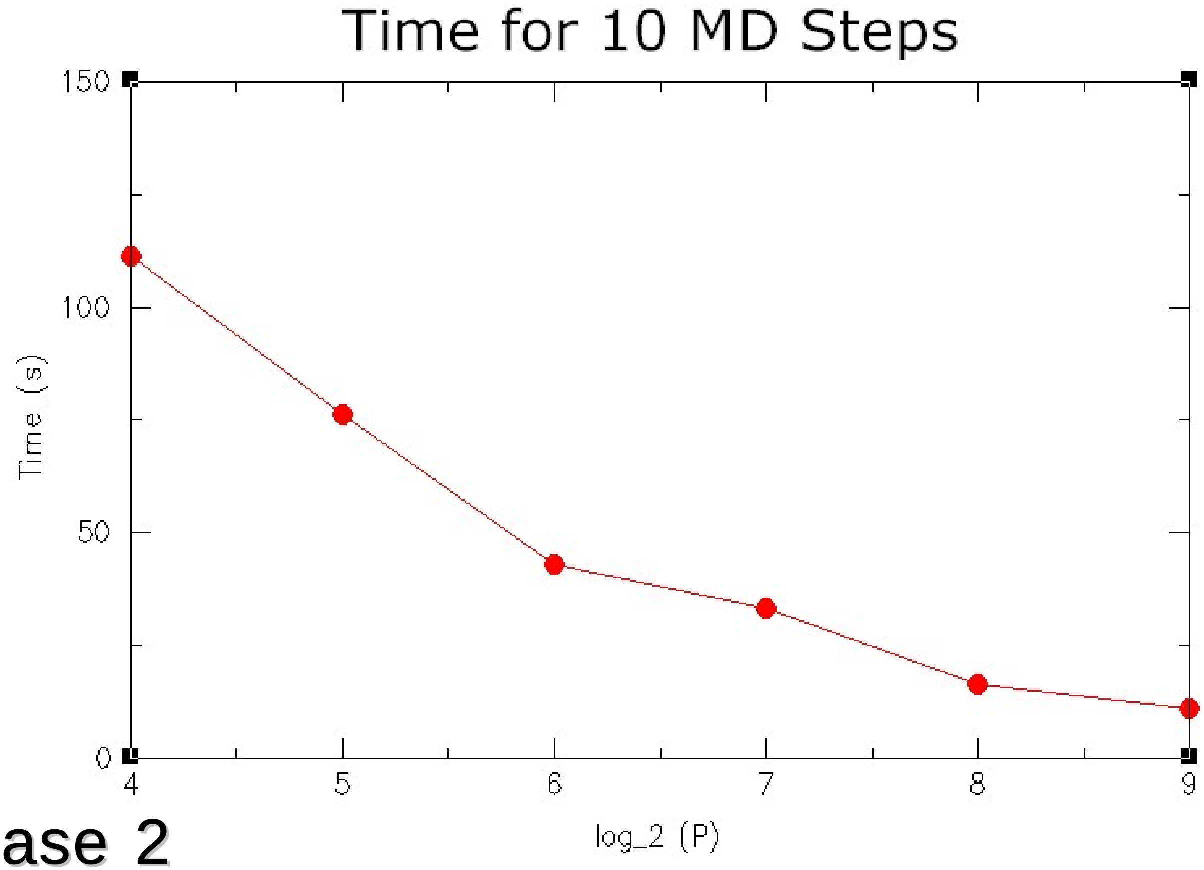
- Test case 1 (552960 atoms, 300Δt)
  - **$NaKSi_2O_5$ - disilicate glass**
  - **SPME (128^3grid)+3 body terms, 15625 LC)**
  - **32-512 processors (4-64 nodes)**
- Test case 2 (792960 atoms, 10Δt)
  - **64xGramicidin(354)+256768 H_2O**
  - **SHAKE+SPME(256^3 grid),14812 LC**
  - **16-256 processors (2-32 nodes)**

DL_POLY_3 on HPCx



Case 1

DL_POLY_3 on HPCx



Case 2

Course Summary

- So I hope I Have
  - **Introduced you to what parallel computers are capable of**
  - **Shown why it is very rarely possible to get perfect scaling**
  - **Introduced you to the *de facto* parallel programming standards**
    - *MPI*
    - *OpenMP*
  - **Introduced you to one or two common parallel programming methods**
  - **Given you a feeling about how to think about how to get effective use out of parallel codes**
  - **Introduced you to how a couple of real, large scale codes actually work**
- I also hope that you have enjoyed it !

Acknowledgements

- All the people in the CSE department at DL, especially
  - **Mike Ashworth**
  - **Bill Smith**
  - **Martin Plummer**
  - **Andy Sunderland**
  - **Martyn Guest**
- At EPCC
  - **Mark Bull**
  - **Lorna Smith**
- And to the organizers for inviting me.