The DL_POLY Tutorial

W. Smith Computational Science and Engineering Department
CCLRC Daresbury Laboratory
Warrington WA4 4AD

Programme

A.M.

- Overview of DL_POLY packages
- MD on distributed parallel computers
- DL_POLY under the hood
- DL_POLY on HPCx
- The DL_POLY GUI
- Overview of DL_POLY Hands-on session

PM

- DL_POLY Hands-on session

Part 1

Overview of the DL_POLY Packages

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 30,000 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

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

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

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

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

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
  - Ewald* & SPME (3D), HK Ewald* (2D), Adiabatic shell model, Reaction field, Neutral groups*, Bare Coulombic, Shifted Coulombic
- Externally applied field
  - Walled cells,electric field,shear field, etc

* **Not in DL_POLY_3**

Boundary Conditions

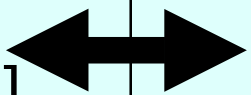
- None (e.g. isolated macromolecules)
- Cubic periodic boundaries
- Orthorhombic periodic boundaries
- Parallelepiped periodic boundaries
- Truncated octahedral periodic boundaries
- Rhombic dodecahedral periodic boundaries
- Slabs (i.e. x,y periodic, z nonperiodic)

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

DL_POLY_2&3 Differences

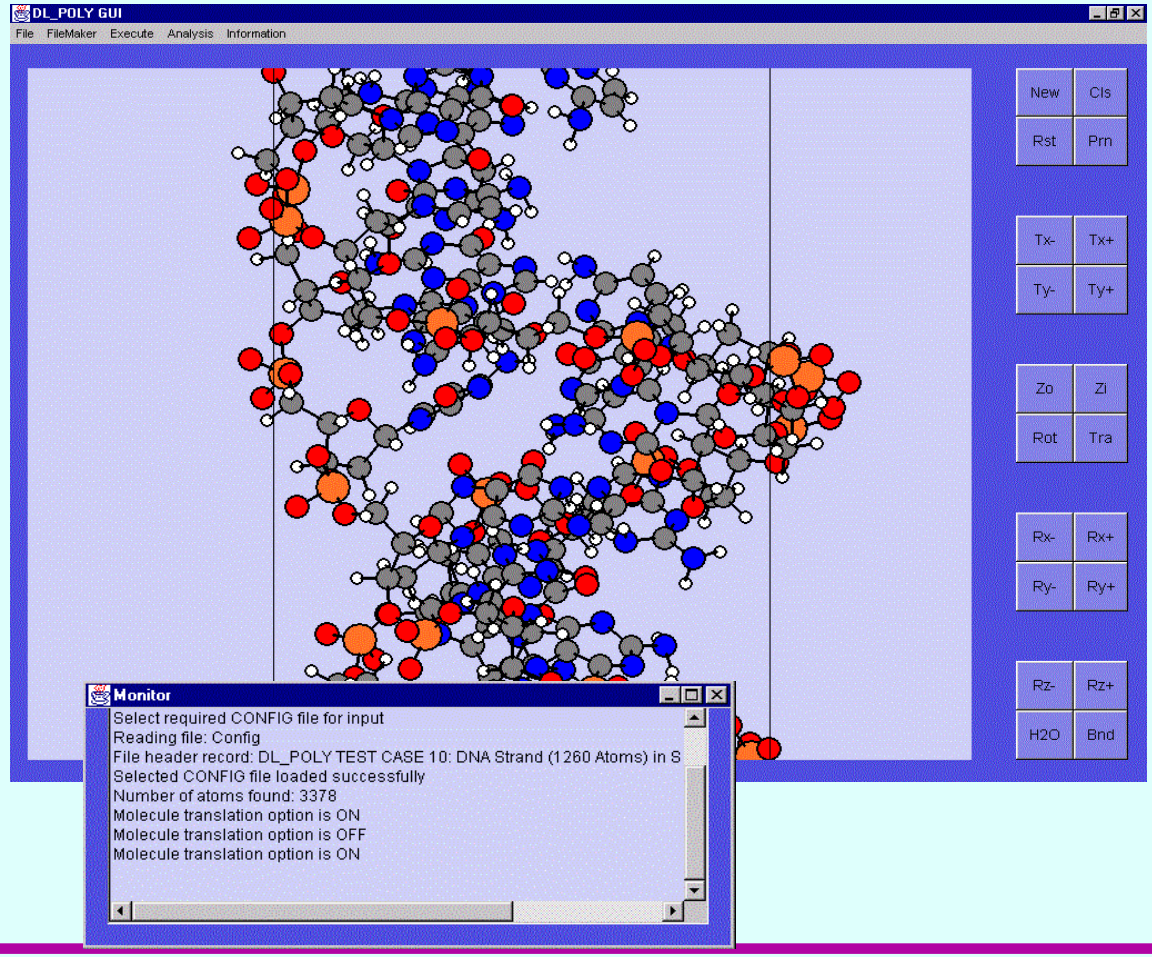
- Rigid bodies not in _3
- MSD not in _3
- Tethered atoms not in _3
- Standard Ewald not in _3
- HK_Ewald not in _3
- DL_POLY_2 I/O files work in _3 but NOT vice versa
- No multiple timestep in _3
- No potential of mean force in _3

DL_POLY_2 Spin-Offs

- DL_MULTI - Distributed multipoles
- DL_PIMD - Path integral (ionics)
- DL_HYPE - Rare event simulation*
- DL_POLY - Symplectic version*

* Under development

The DL_POLY Java GUI



The DL_POLY Java GUI

- Works on any platform (Java 1.3)
- Allows visualisation/analysis of DL_POLY input and output files
- Input file generation features
- Force field builders (ongoing)
- Can be used for job submission
- Extendable by user

DL_POLY People

- Bill Smith DL_POLY_2 & _3 & GUI
  - w.smith@dl.ac.uk
- Ilian Todorov DL_POLY_3
  - i.t.todorov@dl.ac.uk
- Ian Bush DL_POLY optimisation
  - i.j.bush@dl.ac.uk
- Maurice Leslie DL_MULTI
  - m.leslie@dl.ac.uk

Information

http://www.cse.clrc.ac.uk/msi/software/DL_POLY/index.shtml

W. Smith and T.R. Forester,
J. Molec. Graphics, (1996), **14**, 136

W. Smith, C.W. Yong, P.M. Rodger,
Molecular Simulation (2002), **28**, 385

Part 2

Molecular Dynamics on Parallel Computers

Criteria for Assessing Parallel Strategies

- Load Balancing:
  - Sharing work equally between processors
  - Sharing memory requirement equally
  - Maximum concurrent use of each processor
- Communication:
  - Maximum size of messages passed
  - Minimum number of messages passed
  - Local versus global communications
  - Asynchronous communication

Scaling in Parallel Computing

Type 1 Scaling

- Performance scaling with problem size

Type 2 Scaling

- Performance scaling with number of processors

Performance Parameters

Time required per step:

$$T_s = T_p + T_c$$

where

- T_s is the time per step
- T_p is the processing (computation) time/step
- T_c is the communication time/step

Performance Parameters

Can also write:

$$T_s = T_p(1 + R_{cp})$$

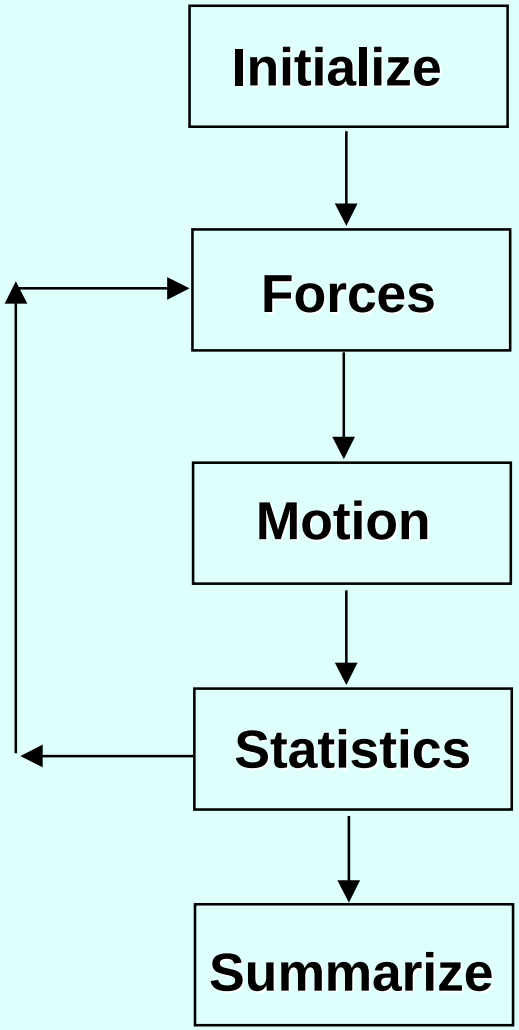
where:

$$R_{cp} = T_c / T_p$$

R_{cp} is the *Fundamental Ratio**

(*NB Assume synchronous communications)

Key Stages in MD Simulation

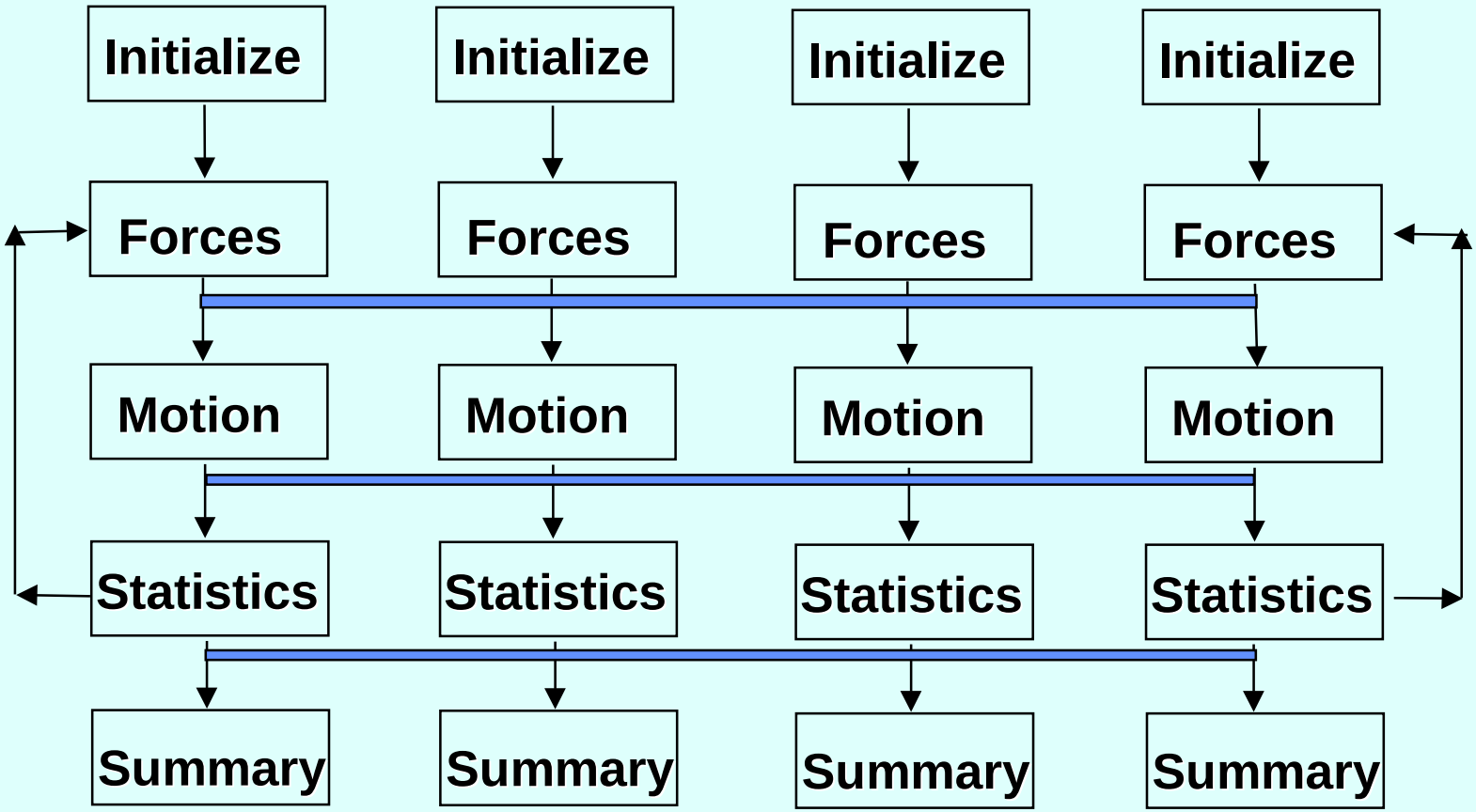


- *Set up initial system*
- *Calculate atomic forces*
- *Calculate atomic motion*
- *Calculate physical properties*
- *Repeat !*
- *Produce final summary*

Basic MD Parallelization Strategies

- Computing Ensemble (Cloning)
- Hierarchical Control (Master-Slave)
- Replicated Data*
- Systolic Loops
- Domain Decomposition*

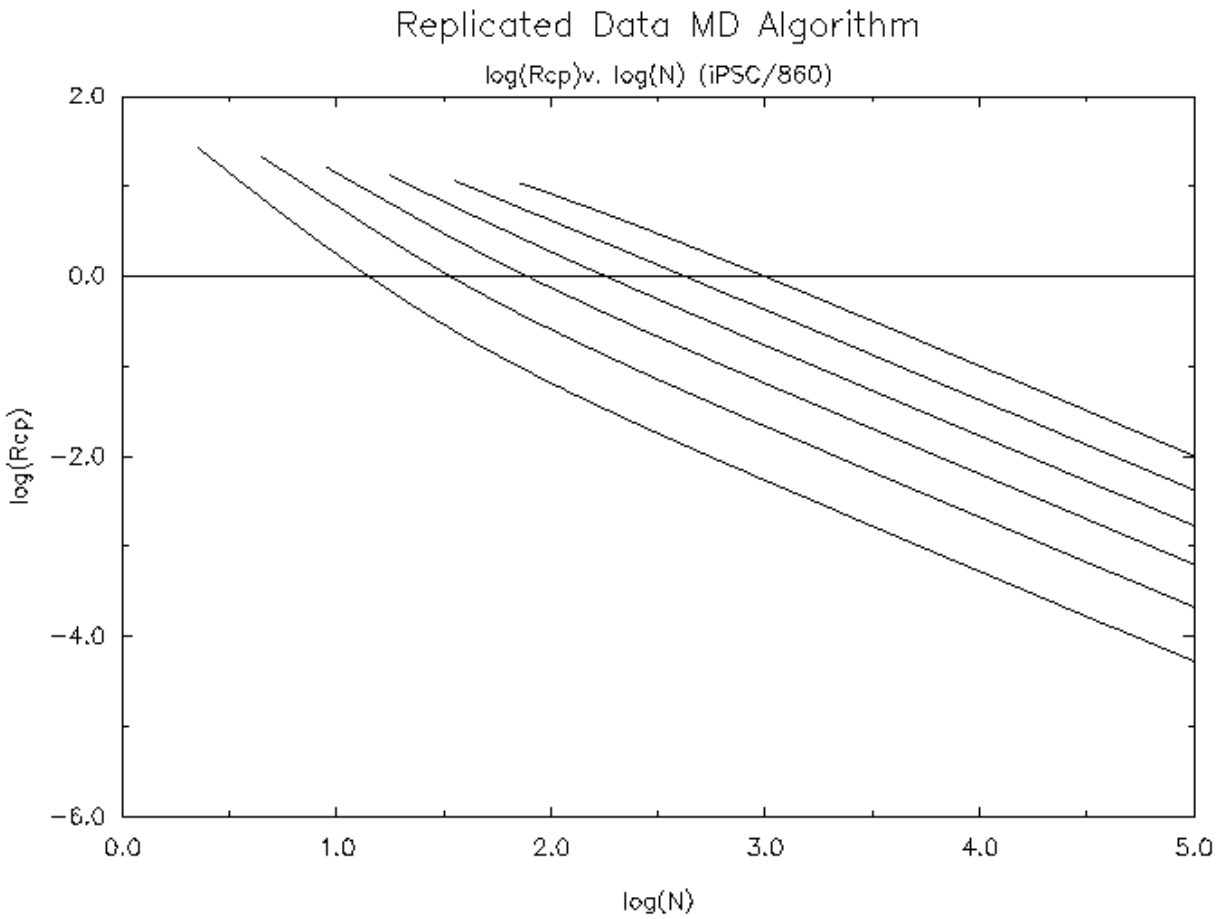
Replicated Data

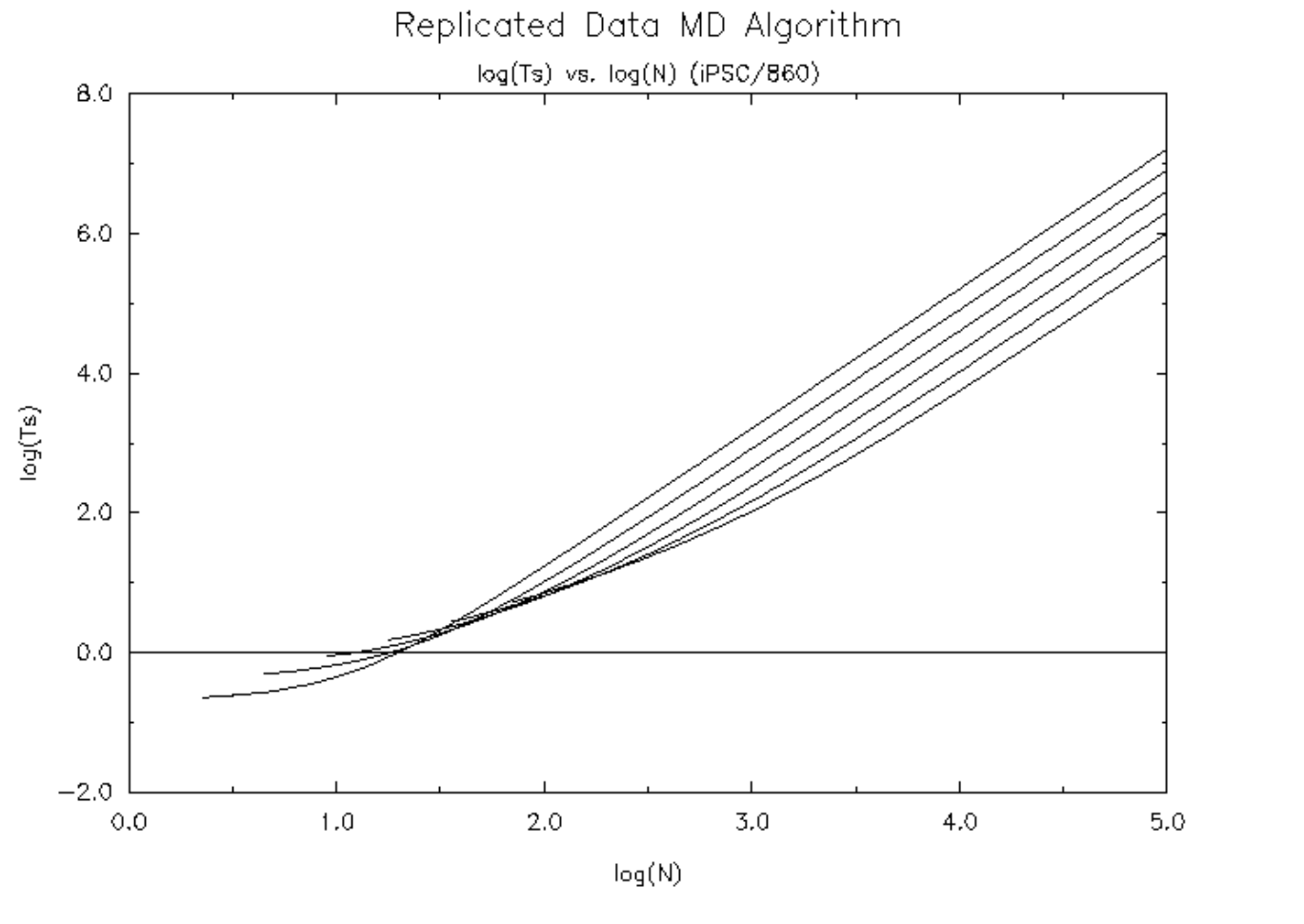


Replicated Data MD Algorithm

Features:

- Each node has copy of all atomic coordinates (Ri,Vi,Fi)
- Force calculations shared equally between nodes (i.e. N(N-1)/2P pair forces per node).
- 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
log(Rcp)v. log(N) (iPSC/860)
2.0
0.0
−2.0
−4.0
−6.0
log(Rcp)
0.0
1.0
2.0
3.0
4.0
5.0
log(N)


Replicated Data MD Algorithm
log(Ts) vs. log(N) (iPSC/860)
8.0
6.0
4.0
2.0
0.0
-2.0
log(Ts)
0.0
1.0
2.0
3.0
4.0
5.0
log(N)

Replicated Data MD Algorithm

Advantages:

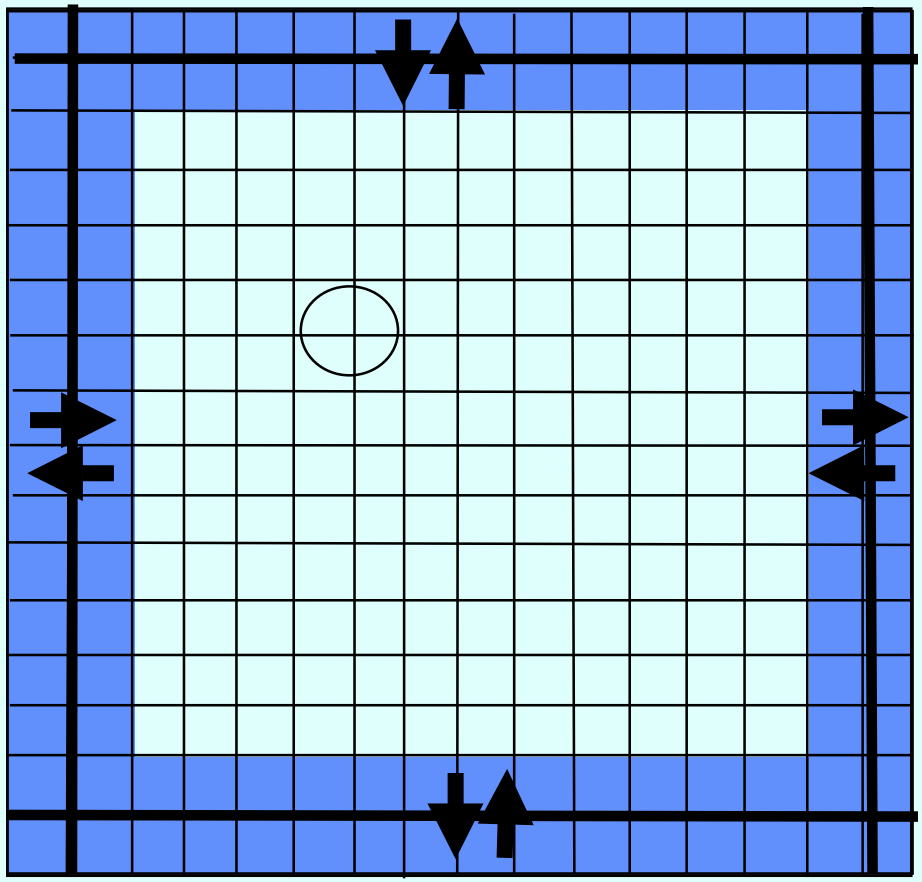
- Simple to implement
- Good load balancing
- Highly portable programs
- Suitable for complex force fields
- Good type 1 scaling
- Dynamic load balancing possible

Replicated Data MD Algorithm

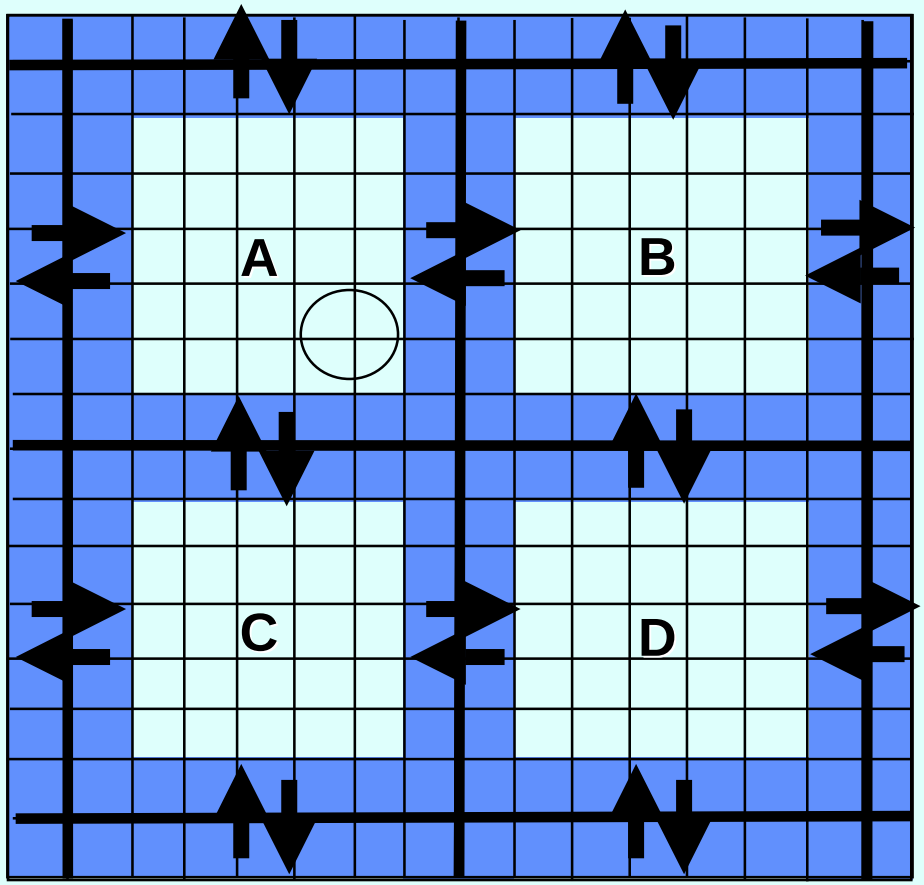
Disadvantages:

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

Link Cell Algorithm



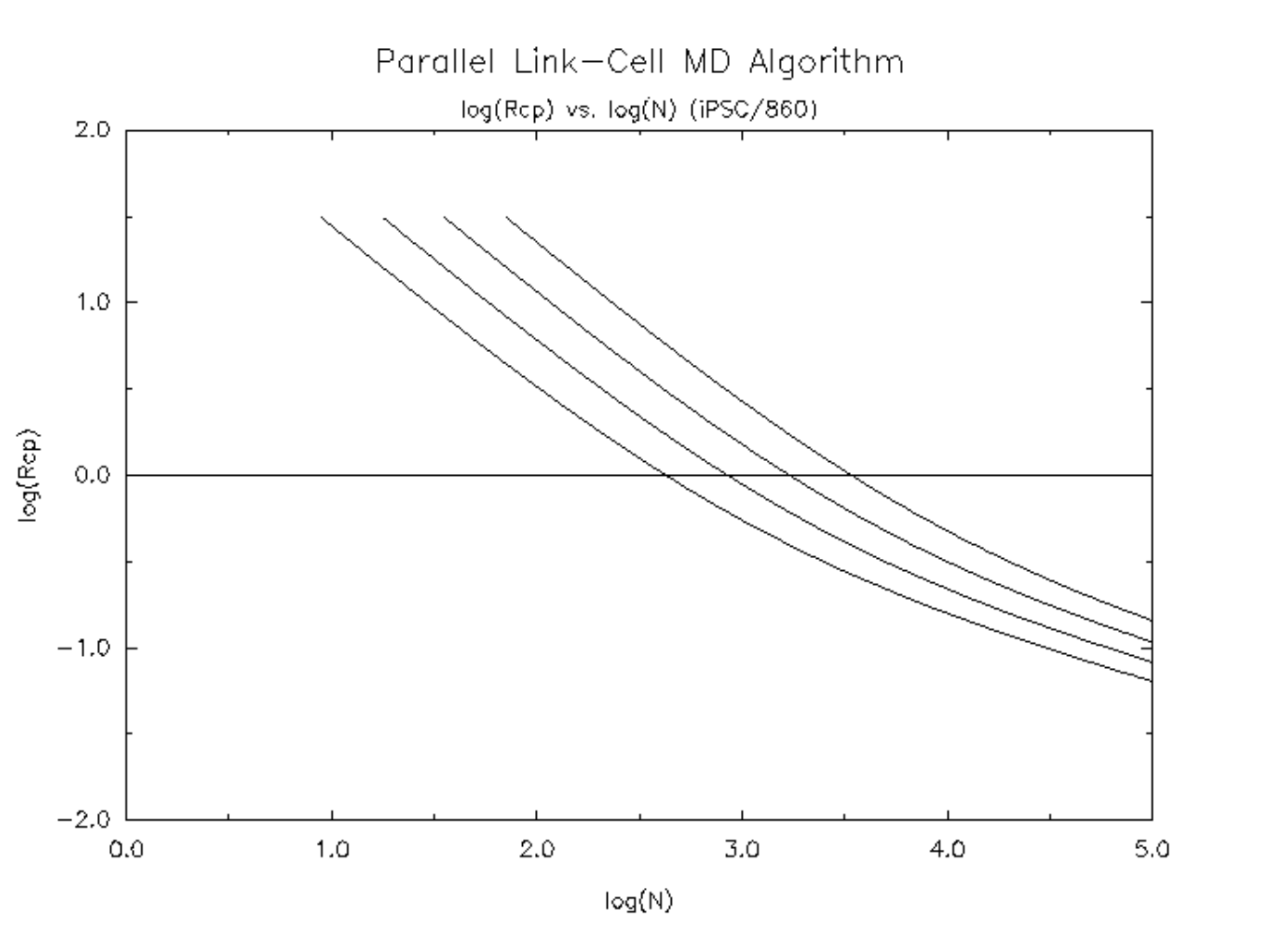
Domain Decomposition MD

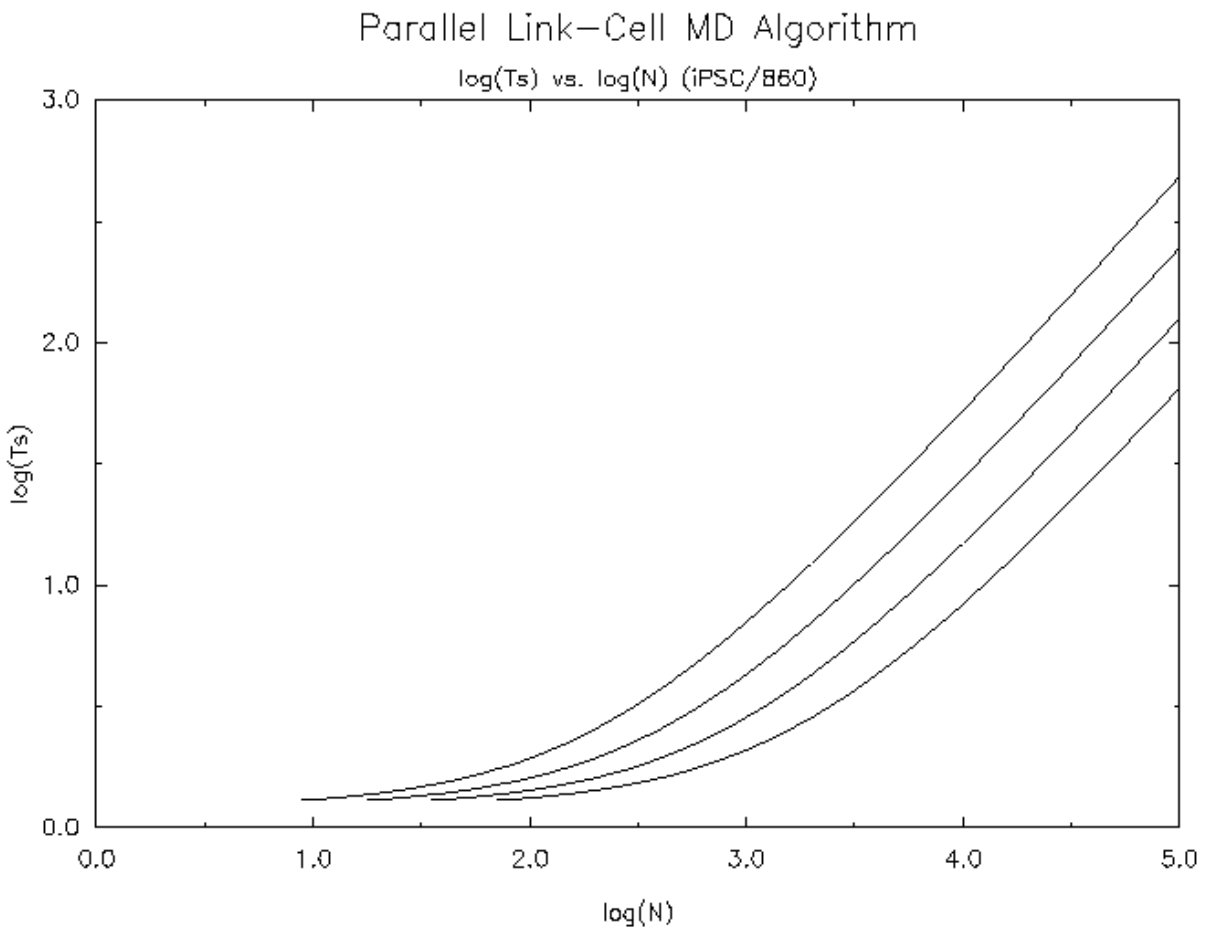


Domain Decomposition MD

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


Parallel Link–Cell MD Algorithm
log(Rcp) vs. log(N) (iPSC/860)
2.0
1.0
0.0
−1.0
−2.0
log(Rcp)
0.0
1.0
2.0
3.0
4.0
5.0
log(N)


Parallel Link–Cell MD Algorithm
log(Ts) vs. log(N) (iPSC/860)
3.0
2.0
1.0
0.0
log(Ts)
0.0
1.0
2.0
3.0
4.0
5.0
log(N)

Domain Decomposition MD

Advantages:

- Good load balancing
- Good type 1 scaling
- Ideal for huge systems (10 ~ 10 atoms)
- Simple communication structure
- Fully distributed memory requirement
- Dynamic load balancing possible

Domain Decomposition MD

Disadvantages

- Problems with mapping/portability
- Sub-optimal type 2 scaling
- Requires short potential cut off
- Complex force fields tricky

Part 3

DL_POLY: Under the Hood

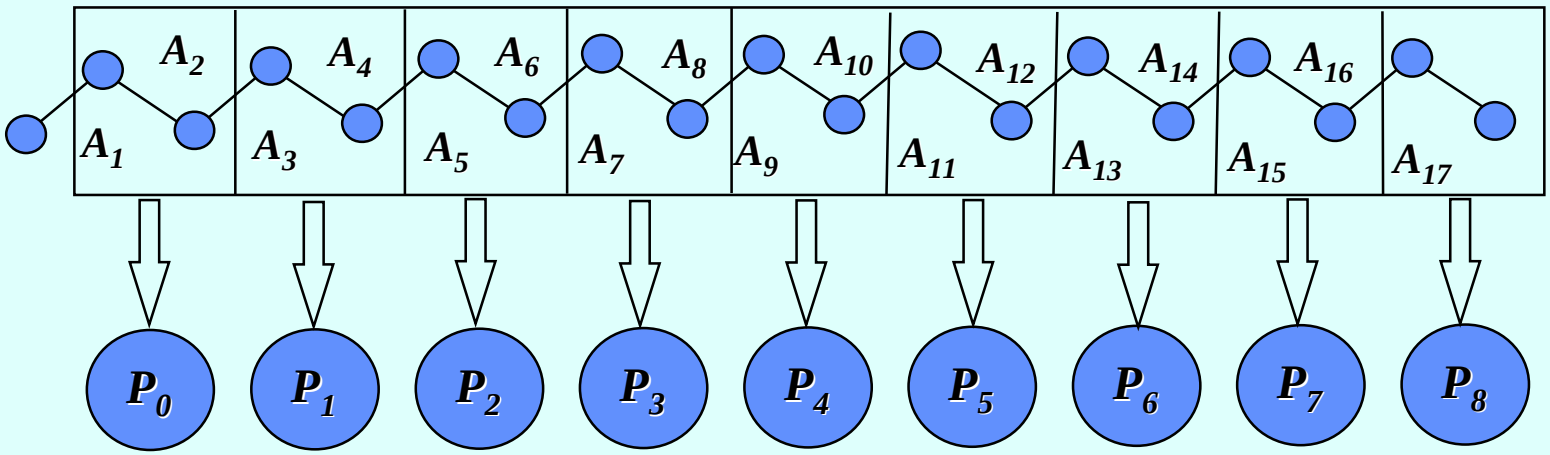
Itinerary

- Parallel force calculation
- The parallel SHAKE algorithms
- The Ewald summation
- The SPME algorithm

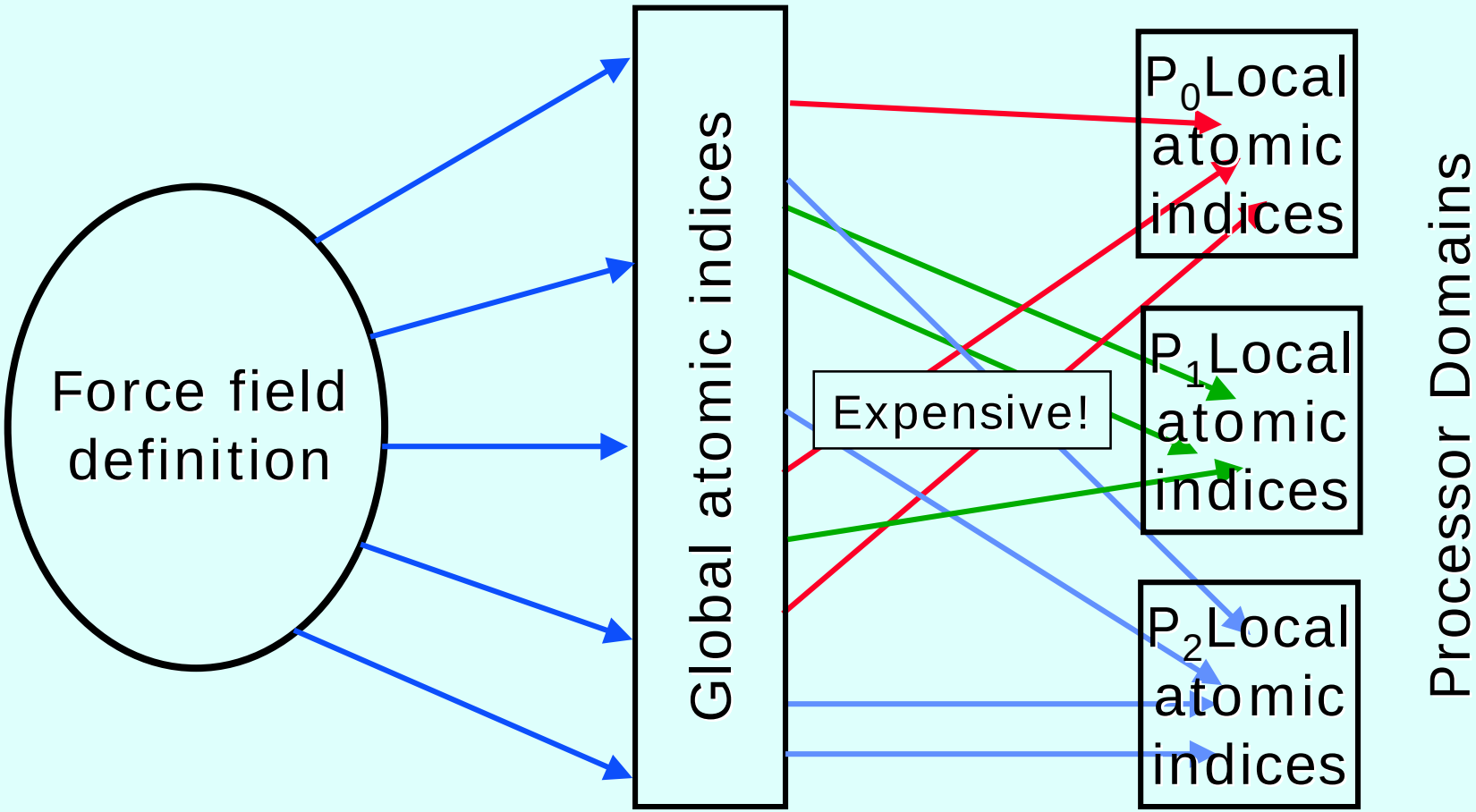
Parallel Force Calculation

- Bonded forces:
  - Algorithmic decomposition
  - Interactions managed by bookkeeping arrays i.e. *explicit bond definition*
  - Shared bookkeeping arrays
- Nonbonded forces:
  - Distributed Verlet neighbour list (pair forces)
  - Link cells (3,4-body forces)
- Implementation different in dl_poly_2&3!

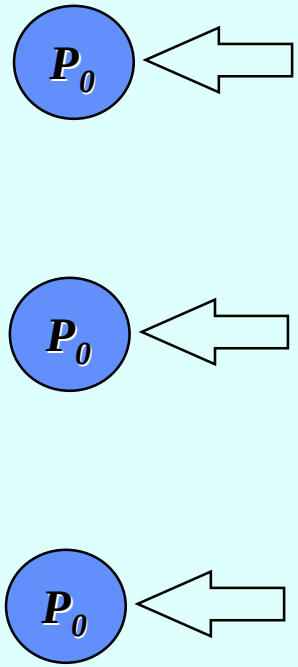
Scheme for Distributing Bond Forces



DL_POLY_3 and Bond Forces



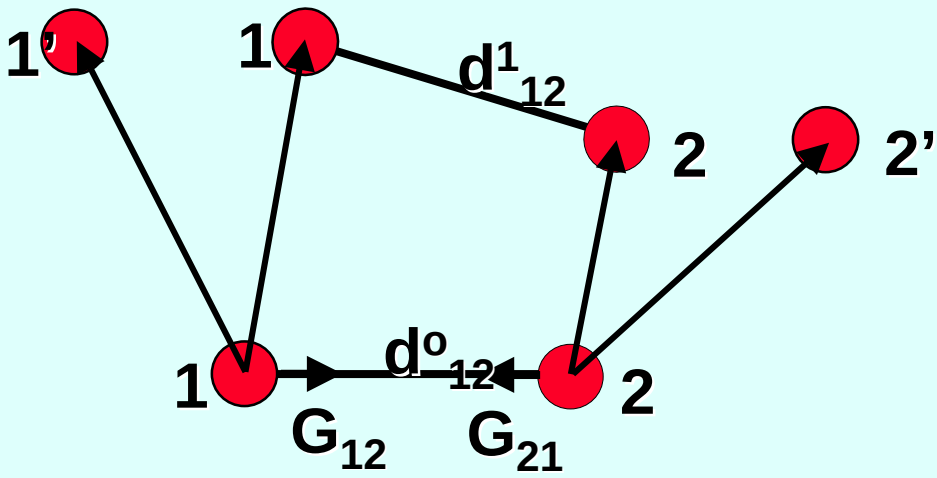
Nonbonded Forces: RD Verlet List



1,2	1,3	1,4	1,5	1,6	1,7
2,3	2,4	2,5	2,6	2,7	2,8
3,4	3,5	3,6	3,7	3,8	3,9
4,5	4,6	4,7	4,8	4,9	4,10
5,6	5,7	5,8	5,9	5,10	5,11
6,7	6,8	6,9	6,10	6,11	6,12
7,8	7,9	7,10	7,11	7,12	
8,9	8,10	8,11	8,12	8,1	
9,10	9,11	9,12	9,1	9,2	
10,11	10,12	10,1	10,2	10,3	
11,12	11,1	11,2	11,3	11,4	
12,1	12,2	12,3	12,4	12,5	

Distributed list!

The SHAKE Algorithm



$$\vec{r}_1 = \vec{r}_1' + \frac{\Delta t^2}{m_1}\vec{G}_{12} \qquad \vec{r}_2 = \vec{r}_2' + \frac{\Delta t^2}{m_2}\vec{G}_{21}$$

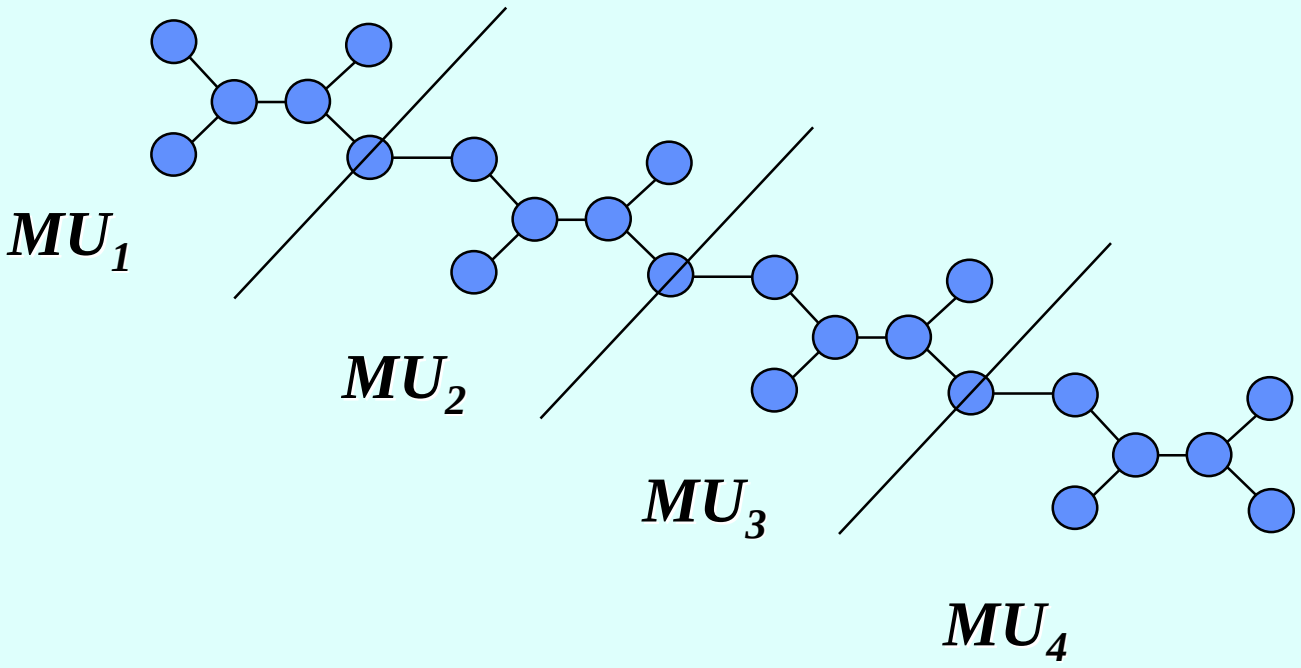
$$\vec{r}_1 - \vec{r}_2 = \vec{r}_1' - \vec{r}_2' + \Delta t^2 \vec{G}_{12}\left(\frac{1}{m_1} + \frac{1}{m_2}\right)$$

$$\vec{d}_{12} = \vec{d}_{12}' + \frac{\Delta t^2}{\mu_{12}} g_{12}\vec{d}^{\,0}_{12} \qquad \vec{G}_{12} = g_{12}\vec{d}^{\,0}_{12}$$

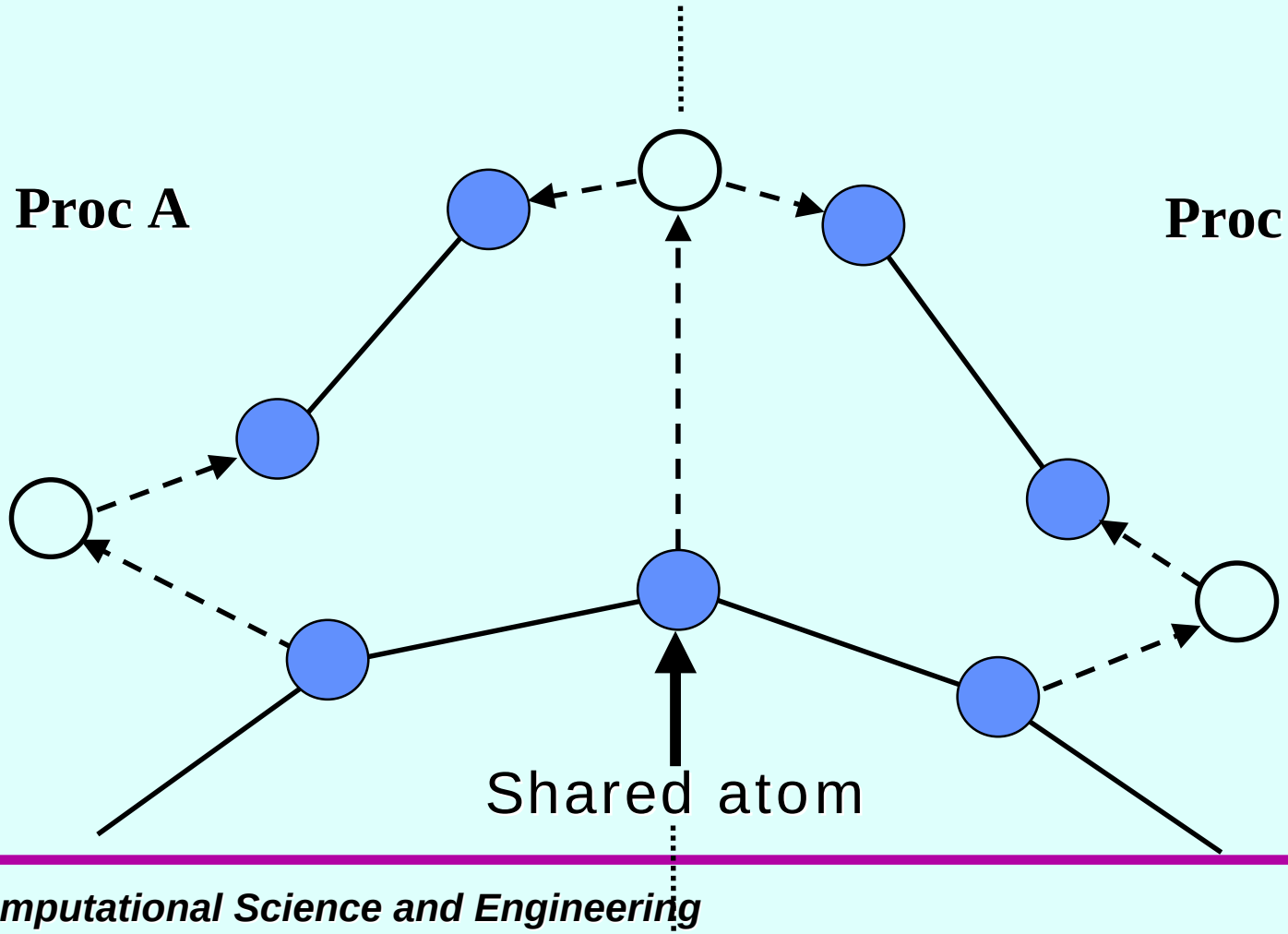
$$d_{12}^2 = d_{12}'^2 + 2\frac{\Delta t^2}{\mu_{12}} g_{12}\vec{d}^{\,0}_{12}\cdot\vec{d}_{12}' + O(g_{12}^2)$$

$$g_{12} \approx \frac{\mu_{12}(d_{12}^2 - d_{12}'^2)}{2\Delta t^2 \vec{d}^{\,0}_{12}\cdot\vec{d}_{12}'}$$

Parallel SHAKE



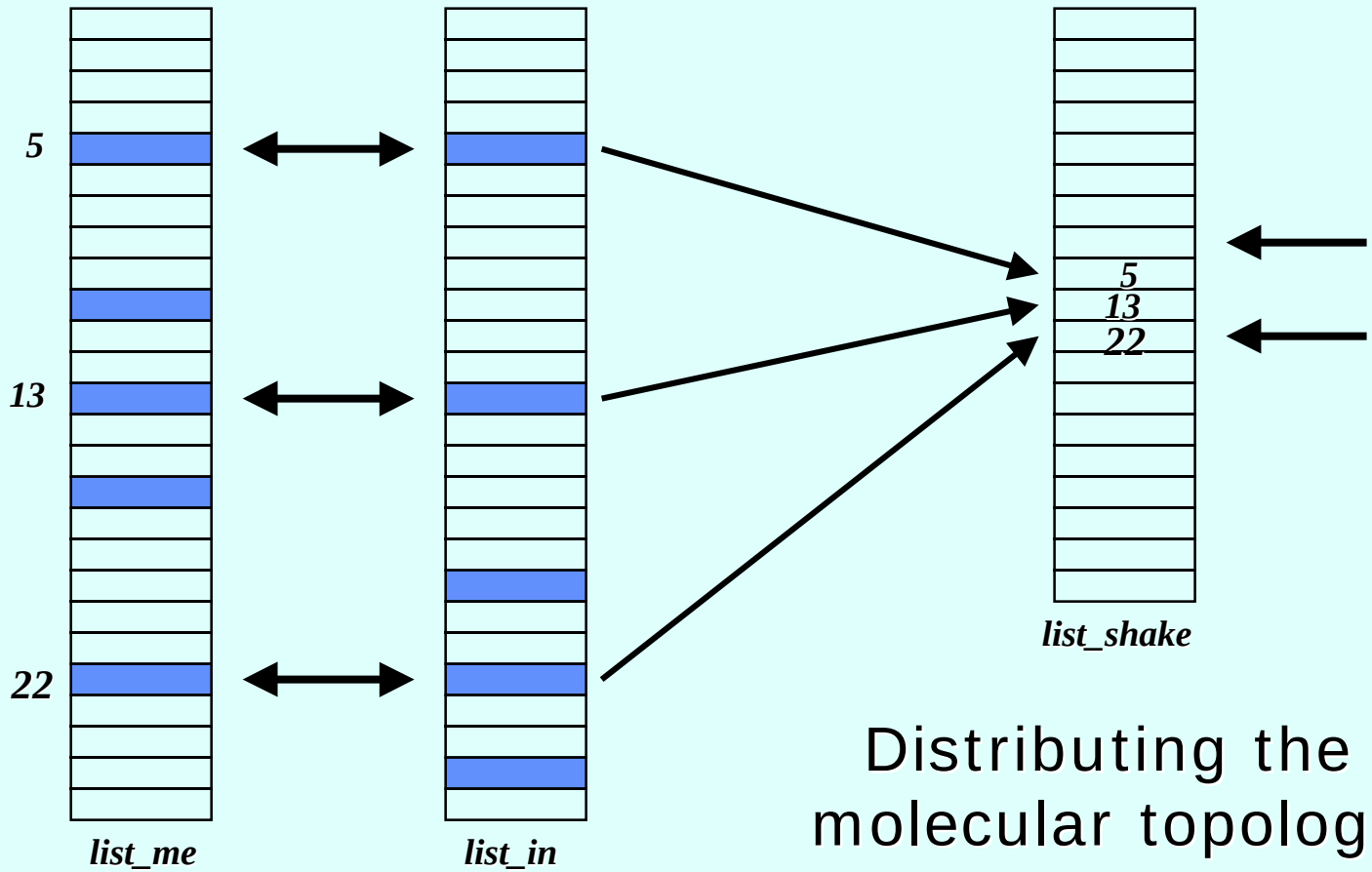
Parallel SHAKE



Parallel SHAKE

- Initialisation:
  - Distribute molecular topology
  - Identify shared atoms
  - Reallocate bonds to reduced shared atoms

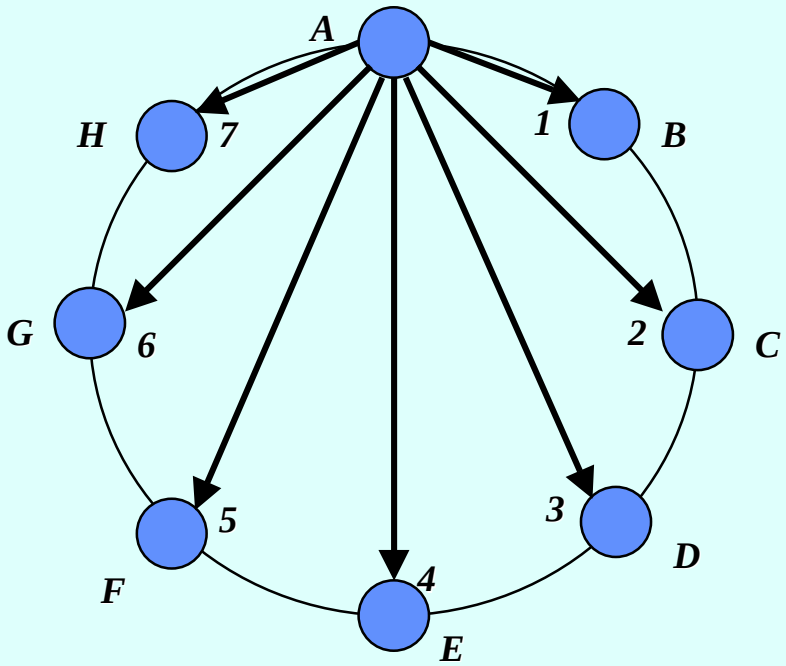
Parallel SHAKE



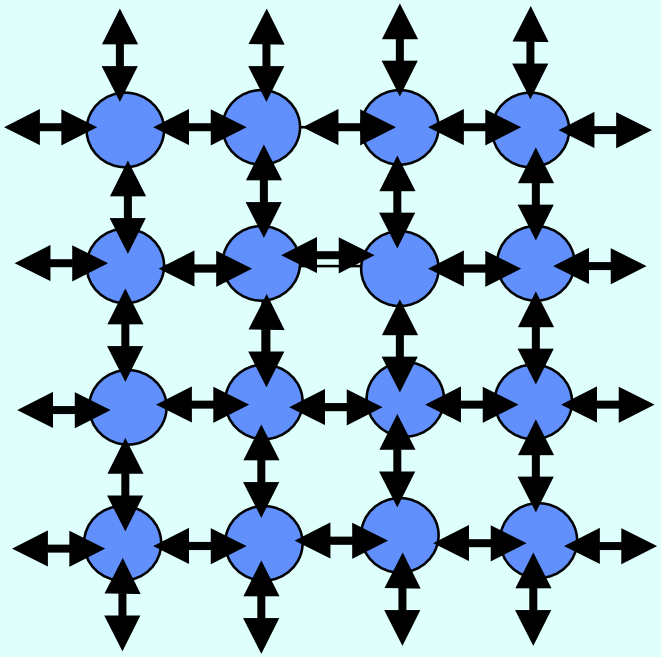
Distributing the molecular topology

Parallel SHAKE

Distributing the molecular topology



Replicated data



Domain decomposition

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

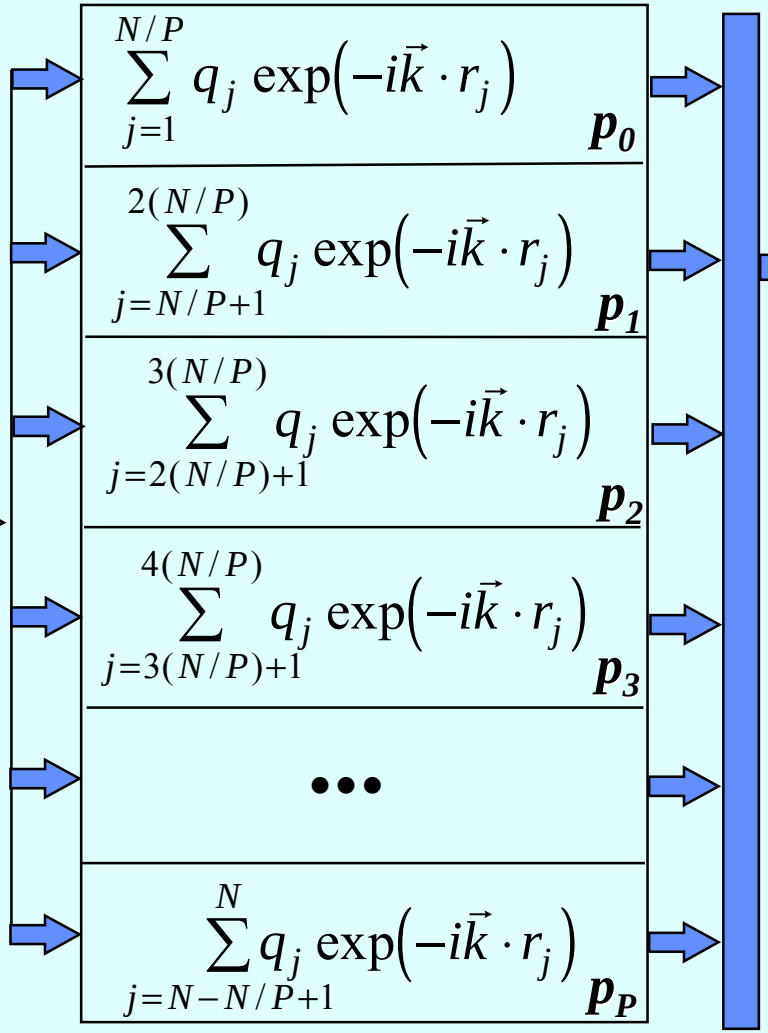
Parallel Ewald Summation

- Self interaction correction - as is.
- Real Space terms:
  - Handle using parallel Verlet neighbour list
  - For excluded atom pairs replace erfc by -erf
- 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



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:

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

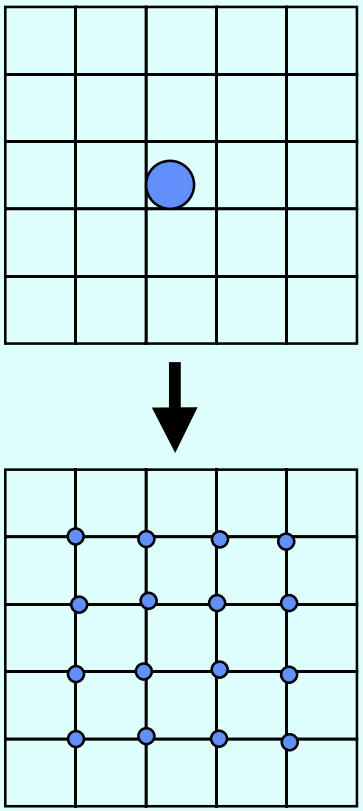
becomes:

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

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

Part 4

DL_POLY on HPCx

HPCx Miscellanea

- Register on-line:
  - Need project code & password from PI
  - http://www.hpcx.ac.uk/projects/new_users/
  - Register, then PI sends notification
  - Get userid & password from HPCx website
- Login
  - ssh -l userid -X login.hpcx.ac.uk
- Tools
  - emacs, vi, loadleveller…

Compiling DL_POLY on HPCx

- Copy Makefile from `build' to`source'
- Use `make hpcx' to compile
- Executable in `execute' directory
  - DLPOLY.X is DL_POLY_2 executable
  - DLPOLY.Y is DL_POLY_3 executable
- Standard executables available in
  - /usr/local/packages/dlpoly/DL_POLY_2/execute/DLPOLY.X
  - /usr/local/packages/dlpoly/DL_POLY_3/execute/DLPOLY.Y

Running DL_POLY on HPCx

- Script:

```
#@  shell = /bin/ksh
#
#@ job_type = parallel
#@ job_name = gopoly
#
#@ tasks_per_node = 8
#@ node = 8
#
#@ node_usage = not shared
#@ network.MPI = csss,shared,US
#
#@  wall_clock_limit = 00:59:00
#@ account_no = xxxx
#
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue
#
export MP_SHARED_MEMORY=yes
poe      ./DLPOLY.Y
```

Running DL_POLY on HPCx

- Job submission:

  llsubmit *job_script*

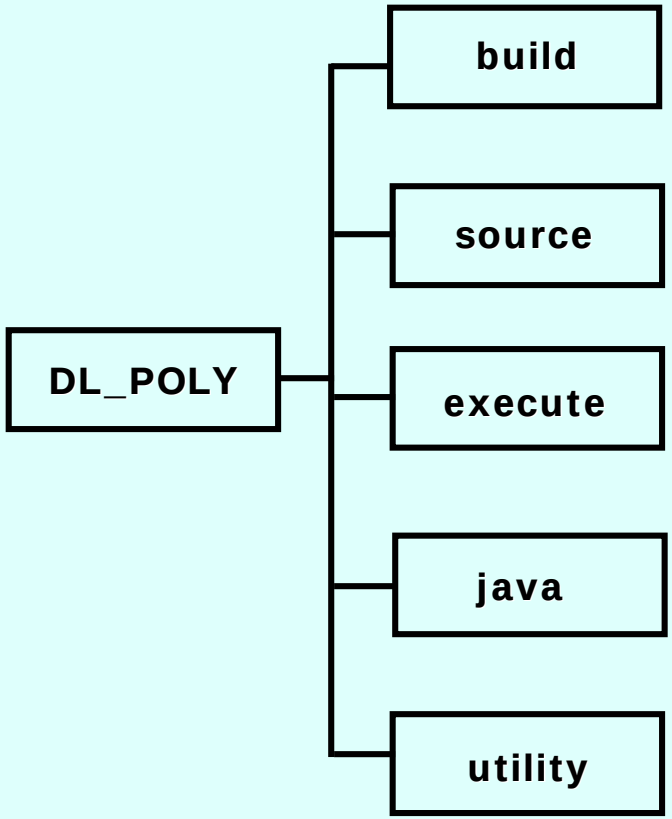
- Job status:

  llq -u *user_id*

- Job cancel:

  llcancel *job_id*

DL_POLY Directories



Home of makefiles

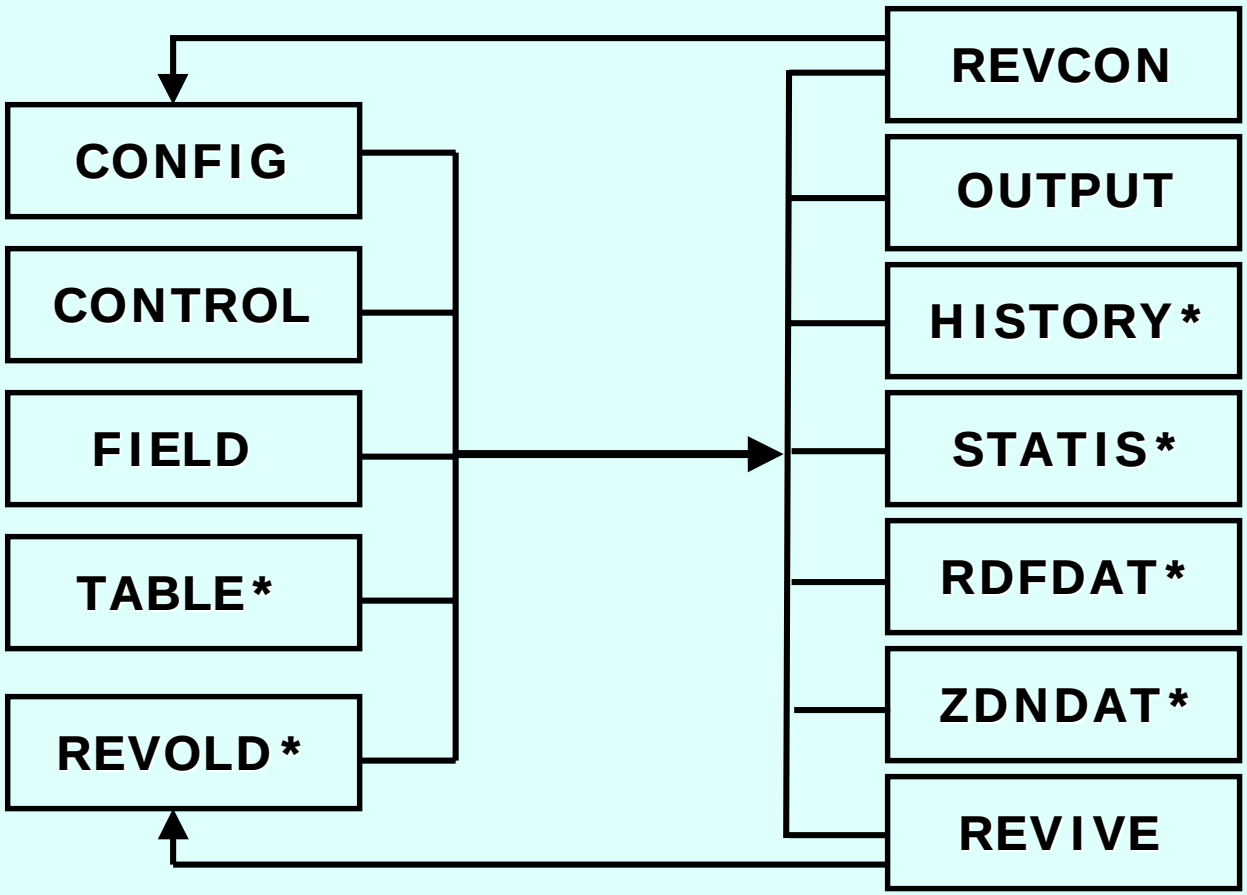
DL_POLY source code

Home of executable & Working Directory

Java GUI source code

Utility codes

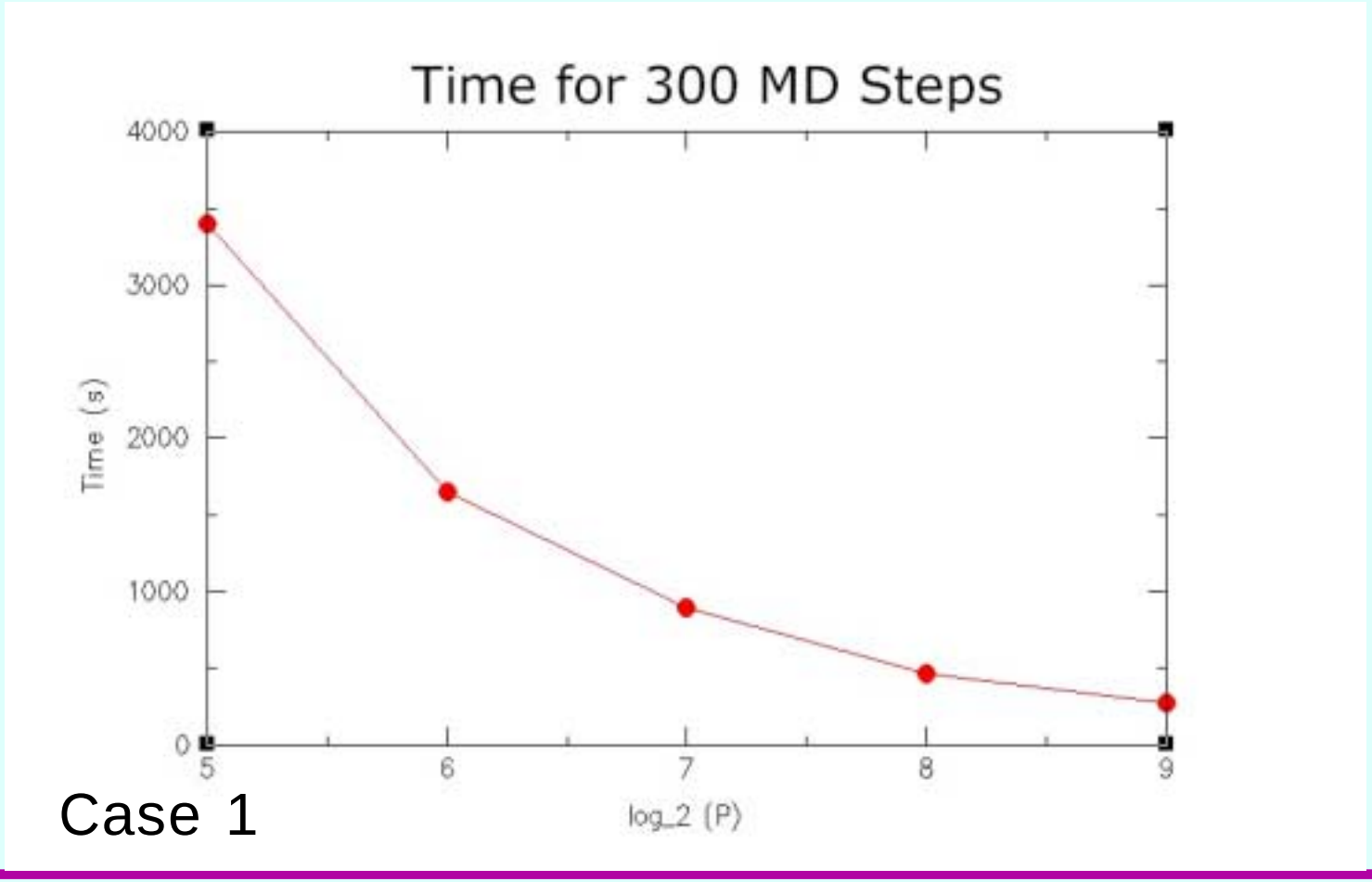
DL_POLY I/O Files



DL_POLY_3 on HPCx

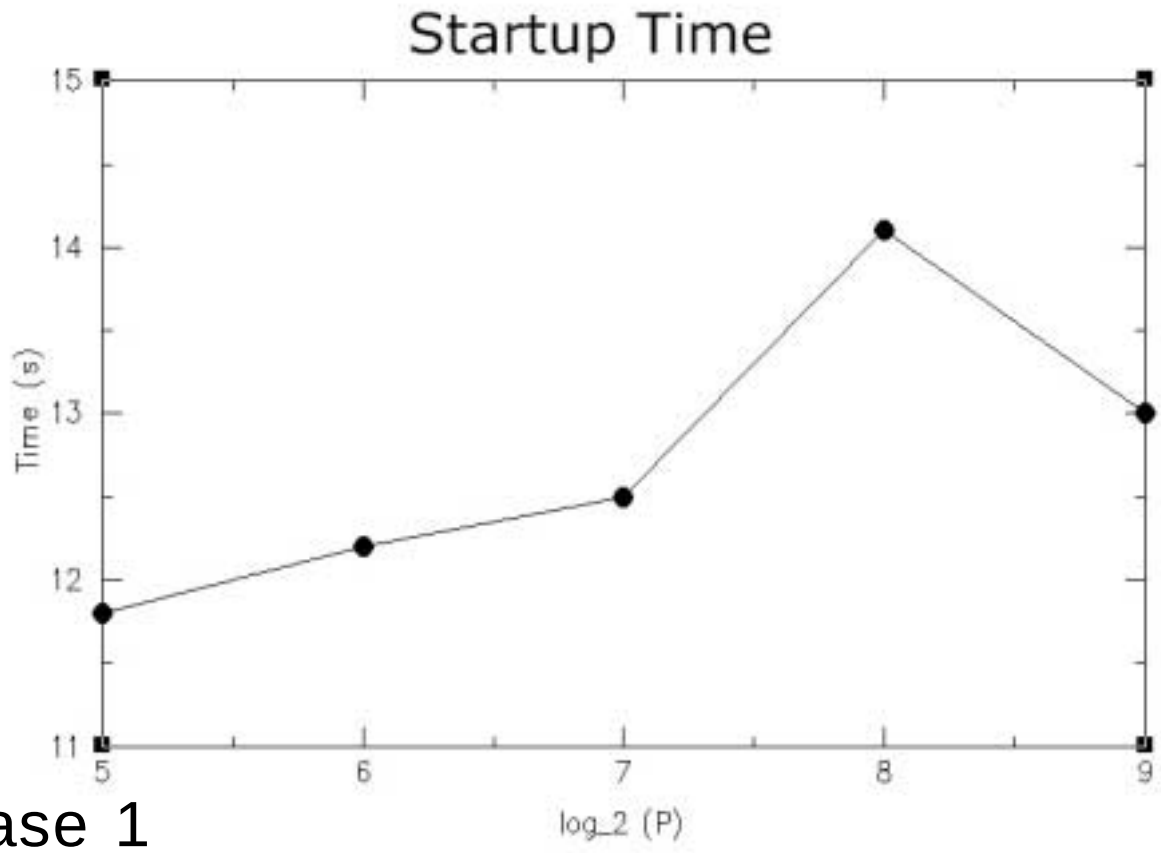
- Test case 1 (552960 atoms, 300Δt)
  - $NaKSi_2O_5$ - disilicate glass
  - SPME (128^3grid)+3 body terms, 15625 LC)
  - 32-512 processors (4-64 nodes)
- Test case 2 (792960 atoms, 10Δt)
  - 64xGramicidin(354)+256768 H_2O
  - SHAKE+SPME(256^3 grid),14812 LC
  - 16-256 processors (2-32 nodes)

DL_POLY_3 on HPCx



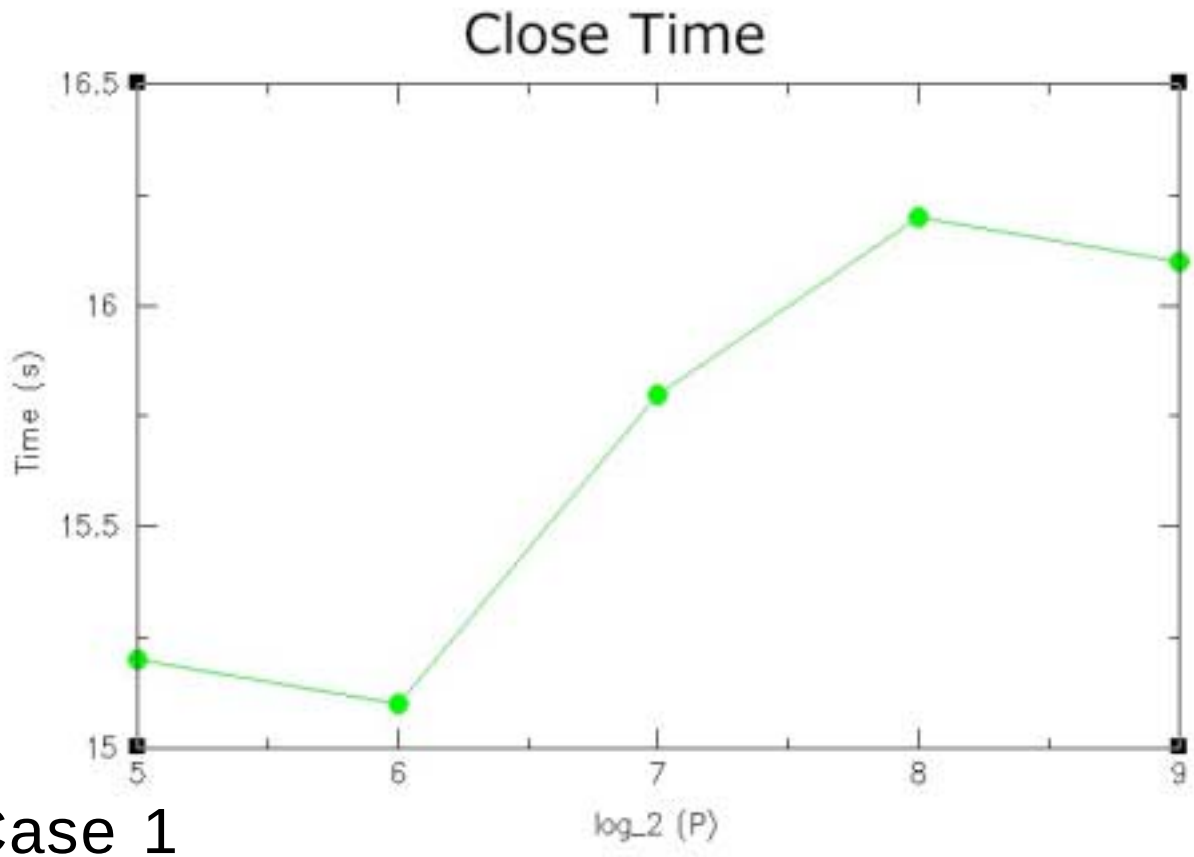
Case 1

DL_POLY_3 on HPCx



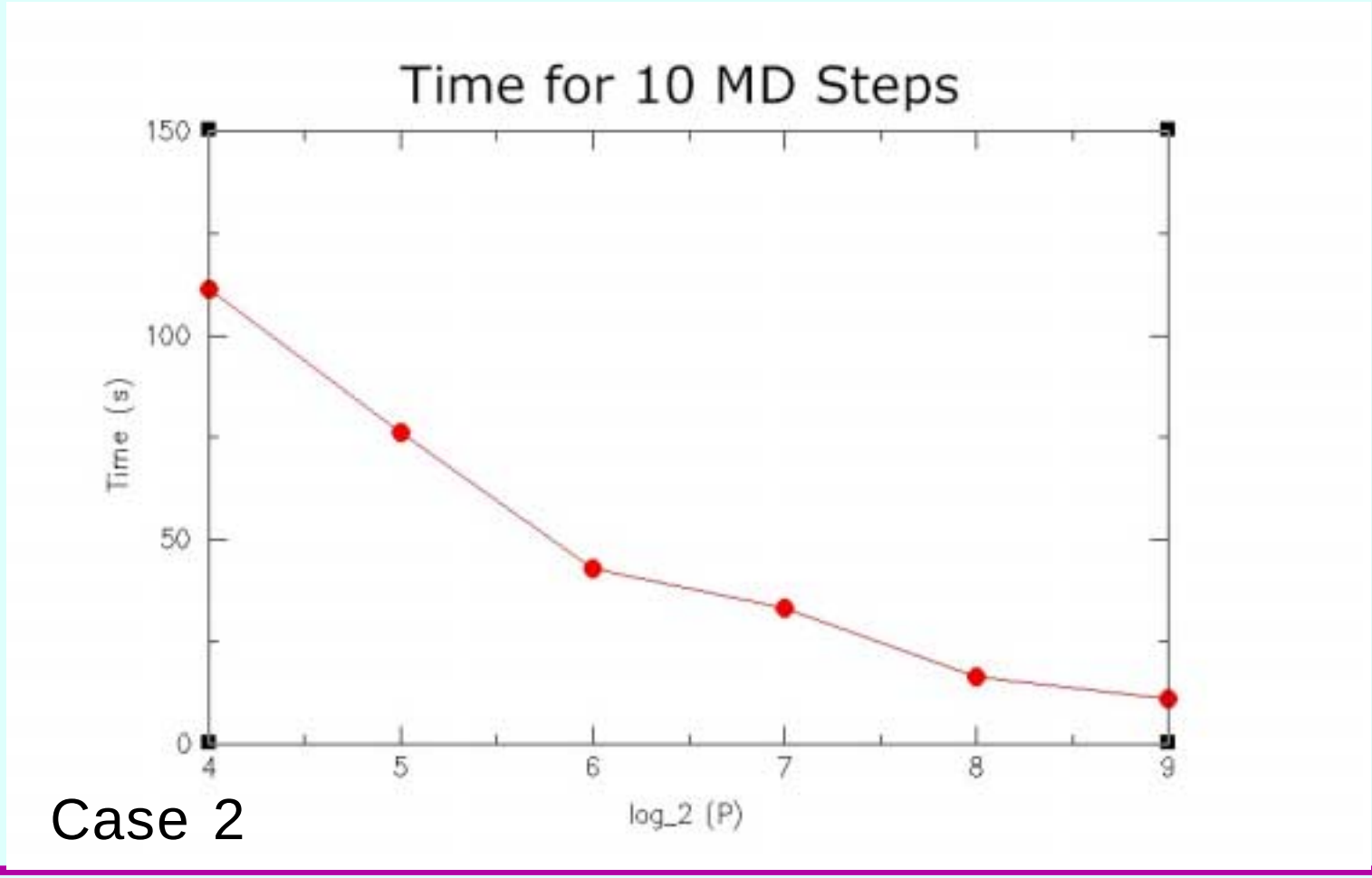
Case 1

DL_POLY_3 on HPCx



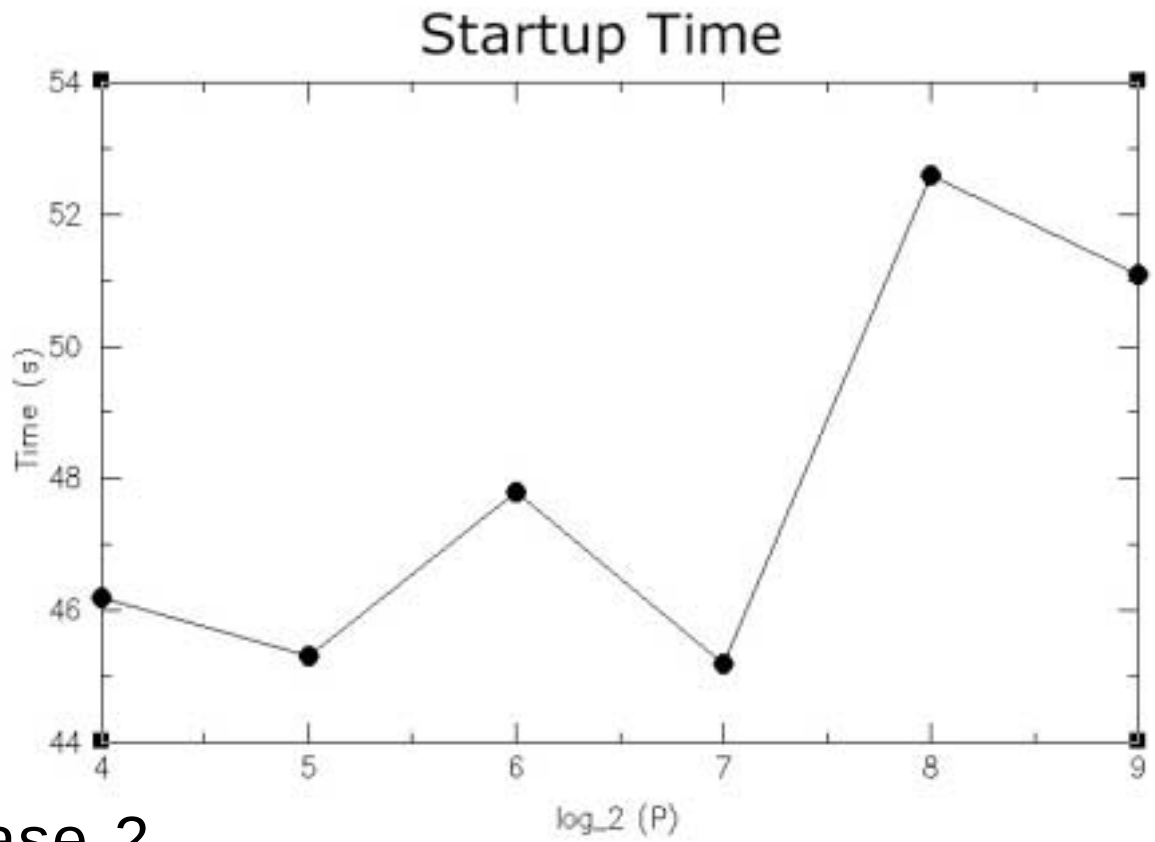
Case 1

DL_POLY_3 on HPCx



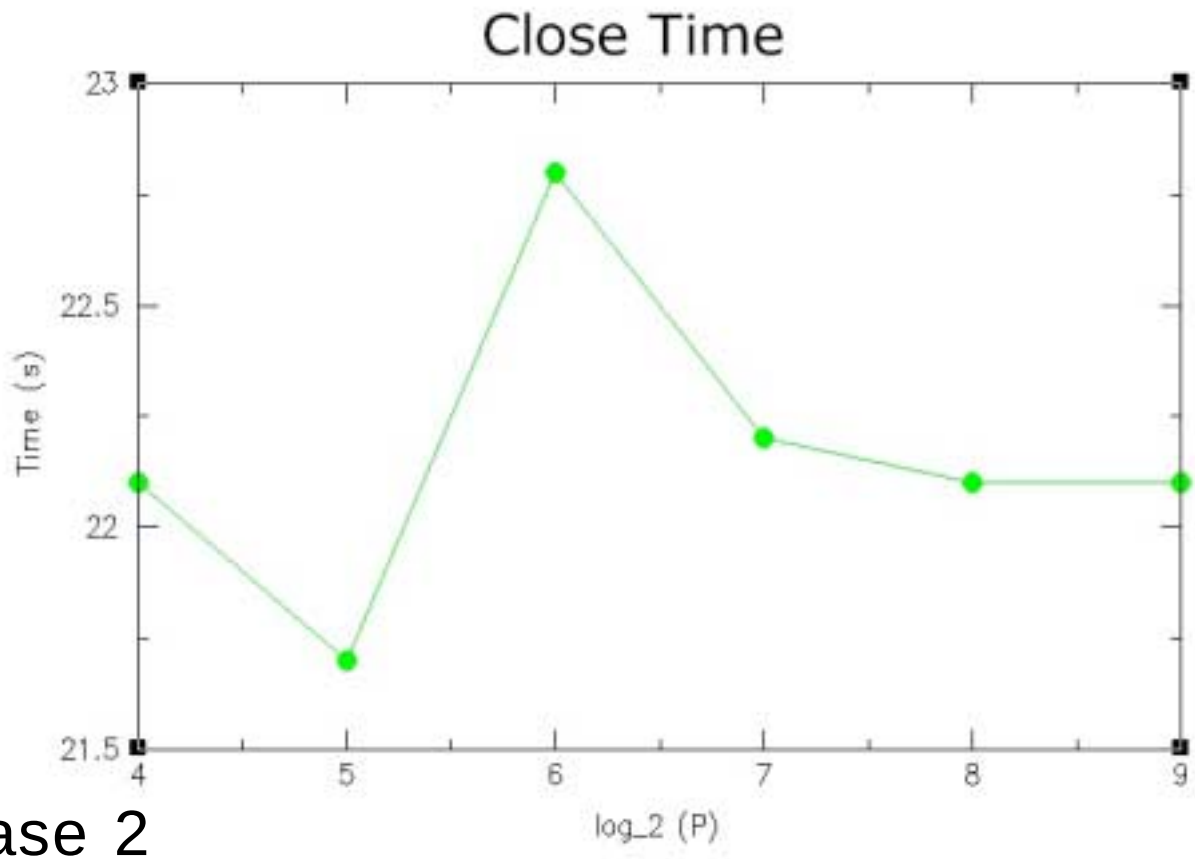
Case 2

DL_POLY_3 on HPCx



Case 2

DL_POLY_3 on HPCx



Case 2

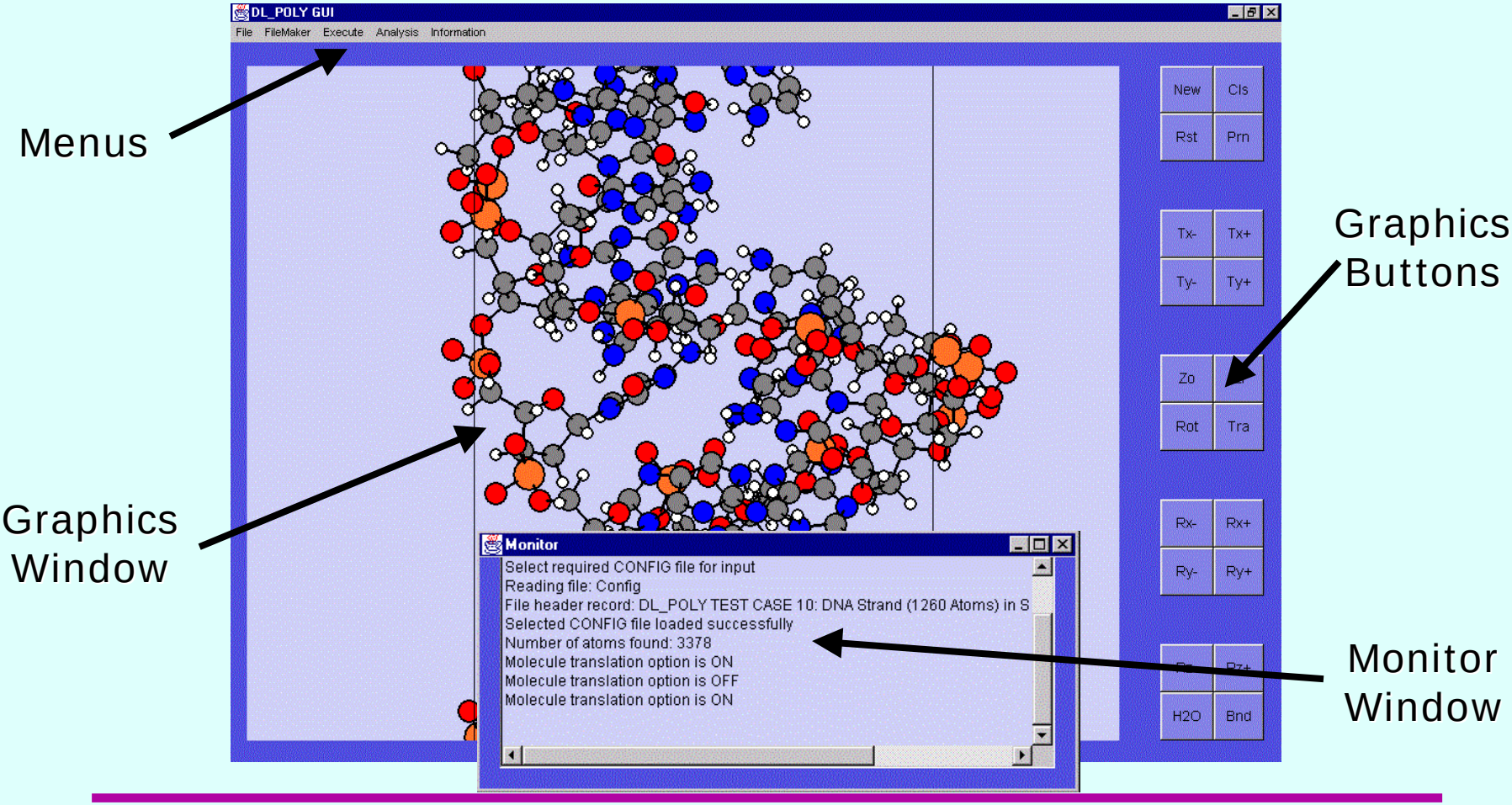
Part 5

The DL_POLY Java GUI

GUI Overview

- Java - Free!
- Facilitate use of code
- Selection of options (control of capability)
- Construct (model) input files
- Control of job submission
- Analysis of output
- Portable and easily extended by user

GUI Appearance



Invoking the GUI

- Invoke the GUI from within the *execute* directory (or equivalent):

  java -jar ../java/GUI.jar

- Colour scheme options:

  java -jar ../java/GUI.jar -*colourscheme*

  with *colourscheme* one of:

  monet, vangoch, picasso, cezanne, mondrian (default picasso).

Compiling/Editing the GUI

- Edit source in *java* directory
- Edit using vi,emacs, *whatever*
- Compile in *java* directory:

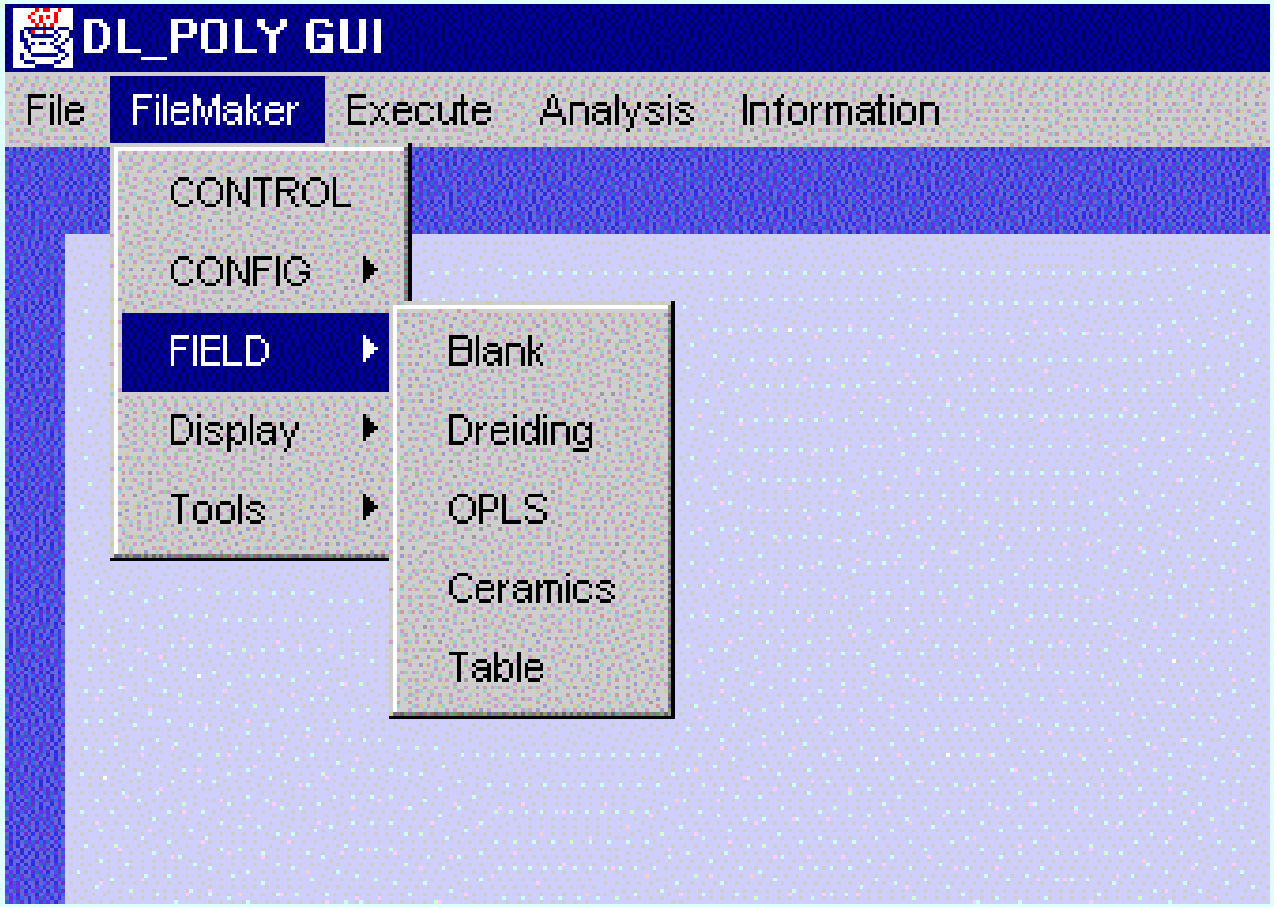
  javac *.java

  jar -cfm GUI.jar manifesto *.class
- Executable is *GUI.jar*

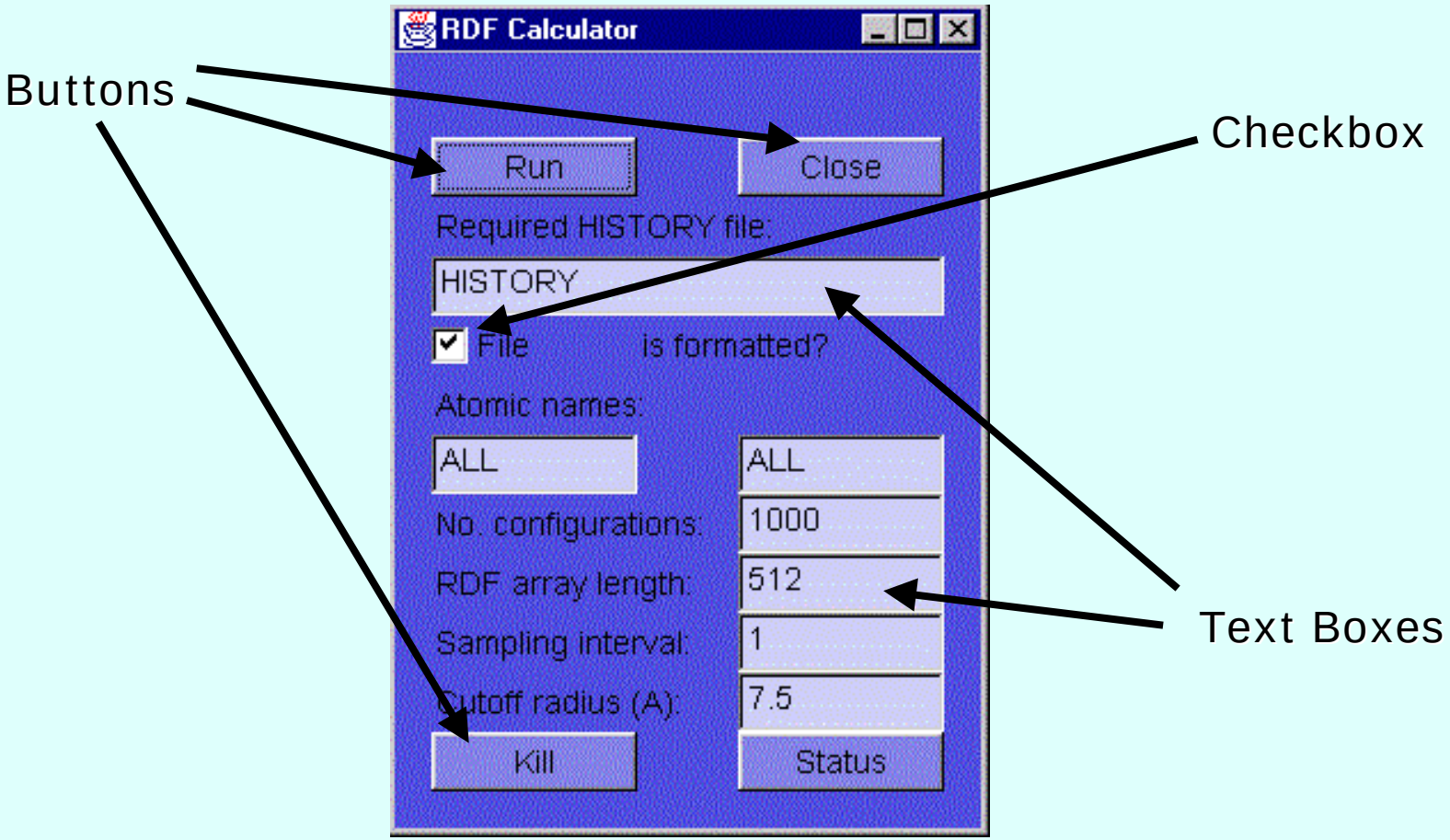
Menus Available

- File - Simple file manipulation, exit etc.
- FileMaker - make input files:
  - CONTROL, FIELD, CONFIG, TABLE
- Execute
  - Select/store input files, run job
- Analysis
  - Static, dynamic,statistics,viewing,plotting
- Information
  - Licence, Force Field files, disclaimers etc

Using the Menus



A Typical GUI Panel



Part 6

DL_POLY Hands-On

"Hands-On Session"

This will consist of three components:

- A demonstration of the Java GUI
- Trying some DL_POLY simulations:
  - prepared exercises, or
  - creative play
- DL_POLY clinic - what's up doc?

Hands-on Session Info

- Invoke Netscape
- Access:

  http://www.cse.clrc.ac.uk/msi/software/DL_POLY/COURSES/Tutorial
- Follow instructions therein!