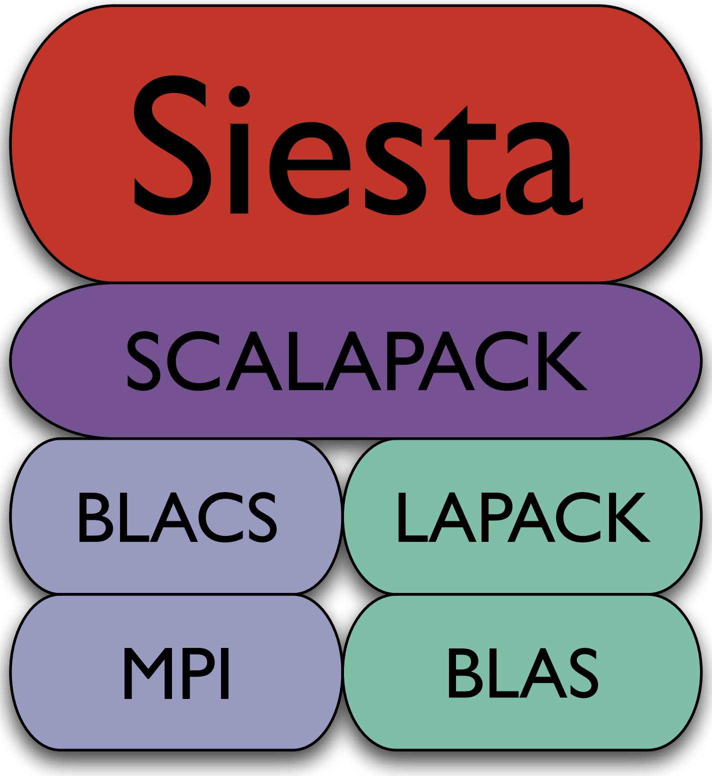
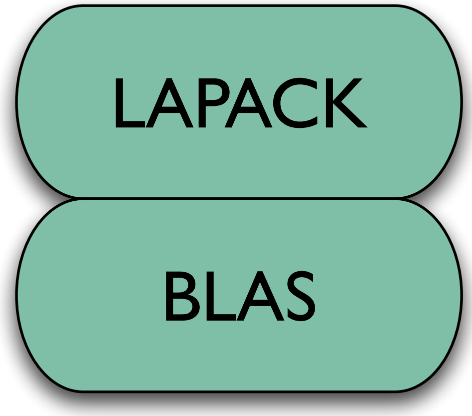
Siesta in parallel

Toby White
Dept. Earth Sciences, University of Cambridge

- How to ***build*** Siesta in parallel
- How to ***run*** Siesta in parallel
- How Siesta ***works*** in parallel
- How to use parallelism ***efficiently***

❖ How to ***build*** Siesta in parallel





Vector/matrix manipulation

Linear Algebra PACKage

`http://netlib.org/lapack/`

Basic Linear Algebra Subroutines

`http://netlib.org/blas/`

ATLAS BLAS:

http://atlas.sf.net

Free, open source (needs separate LAPACK)

GOTO BLAS:

http://www.tacc.utexas.edu/resources/software/#blas

Free, registration required, source available (needs separate LAPACK)

Intel MKL (Math Kernel Library):

Intel compiler only

Not cheap (but often installed on supercomputers)

ACML (AMD Core Math Library)

http://developer.amd.com/acml.jsp

Free, registration required. Versions for most compilers.

Sun Performance Library

http://developers.sun.com/sunstudio/perflib_index.html

Free, registration required. Only for Sun compilers (Linux/Solaris)

IBM ESSL (Engineering & Science Subroutine Library)

http://www-03.ibm.com/systems/p/software/essl.html

Free, registration required. Only for IBM compilers (Linux/AIX)

MPI

Parallel communication

Message Passing Infrastructure

http://www.mcs.anl.gov/mpi/

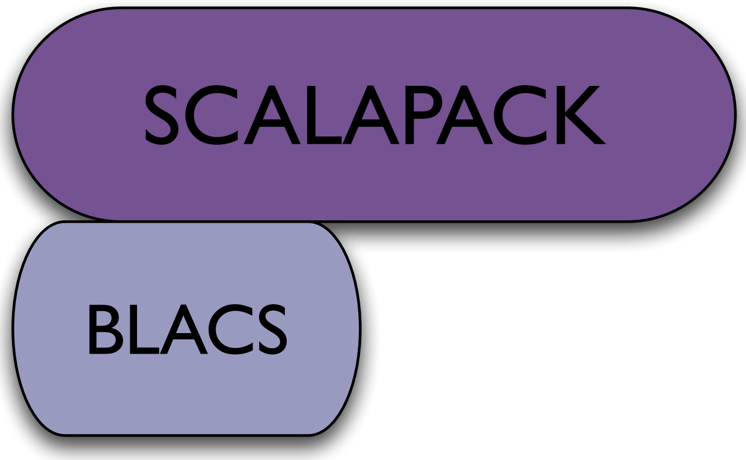
You probably don't care - your supercomputer will have (at least one) MPI version installed.

Just in case, if you are building your own cluster:

MPICH2:
http://www-unix.mcs.anl.gov/mpi/mpich2/

Open-MPI:
http://www.open-mpi.org

And a *lot* of experimental super-fast versions ...



Parallel
linear algebra

Basic Linear Algebra Communication Subprograms

`http://netlib.org/blacs/`

SCAlable LAPACK

`http://netlib.org/scalapack/`

Intel MKL (Math Kernel Library):
Intel compiler only

AMCL (AMD Core Math Library)
http://developer.amd.com/acml.jsp

S3L (Sun Scalable Scientific Program Library)
http://www.sun.com/software/products/clustertools/

IBM PESSL (Parallel Engineering & Science Subroutine Library)
http://www-03.ibm.com/systems/p/software/essl.html

Previously mentioned libraries are *not* the only ones
- just most common.

Compilation instructions different for each
- not trivial ! ! ☹

See `Sys/` directory for examples of use.

If you are completely lost, SIESTA can guess:

```
./configure --enable-mpi
```

but not usually successfully

Do NOT mix & match compilers and libraries

Some supercomputers have multiple versions of everything installed - several compilers, several sets of numerical libraries.

Keep everything consistent ! !

❖ How to ***run*** Siesta in parallel

I can't tell you.

But - no change *needed* in `input.fdf`

On command line, or in jobscript:

```
mpirun -np 4 ./siesta < input.fdf > output.out
```

Sometimes `mprun`, sometimes omitted

Sometimes different flag ...

Sometimes need explicit full path

Sometimes standard input fails on MPI

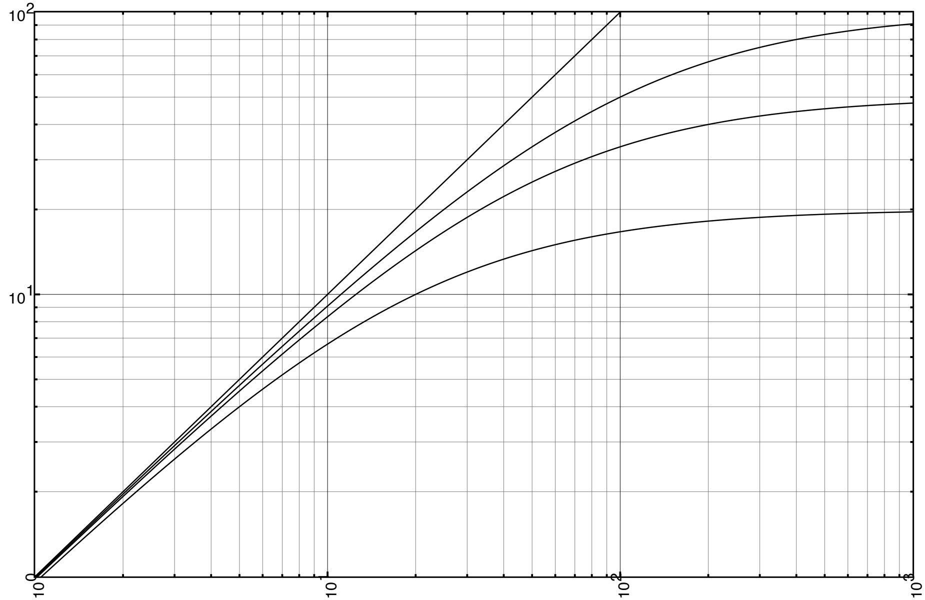
Sometimes standard output fails on MPI

Read your supercomputer documentation!

❖ Principles of parallel programming

Speedup

$\frac{T_1}{T_n}$



nodes

Amdahl's Law:

$$T = T_{\boldsymbol{s}} + T_{\boldsymbol{p}}$$

$$T_{\boldsymbol{s}} \neq 0$$

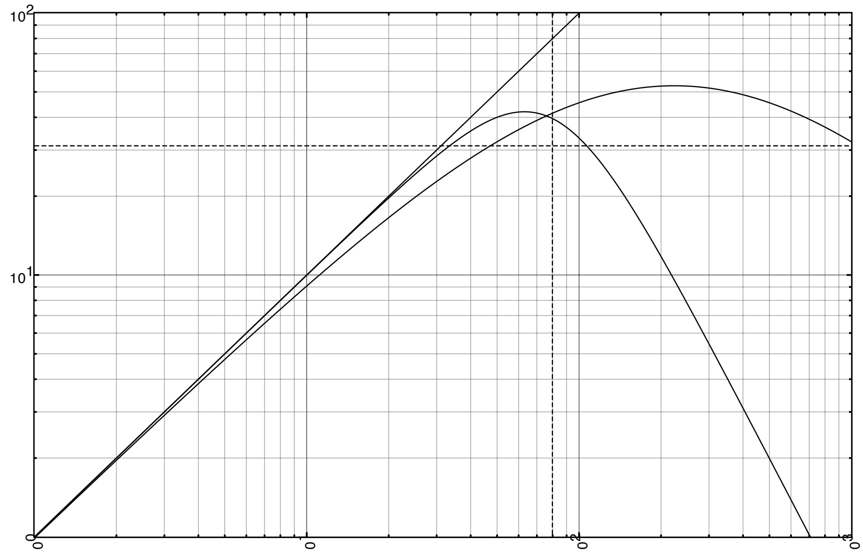
$$T_{\boldsymbol{p}} \geq k/n$$

In the best case, for high enough *n*,
serial time always dominates.

Latency:

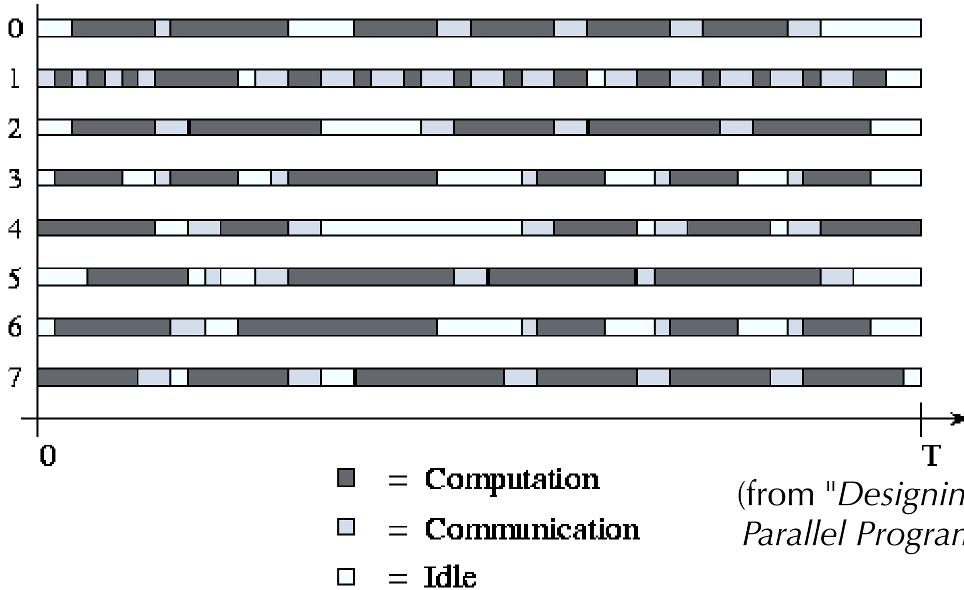
$T_s \neq 0$

$T_p = k_1/n + k_2 n^{k_3}$



For high enough *n*, communication time always dominates.

Load balancing:



(from "*Designing and Building Parallel Programs*", Ian Foster)

$$T = T_{comp} + T_{comm} + T_{idle}$$

Amdahl's Law

Communications Latency

Load balancing

Total CPU time!!

❖ How Siesta ***works*** in parallel

`ParallelOverK` → kPoint parallelization

`Diagon` → matrix parallelization

`OrderN` → spatial decomposition

kPoint parallelization

Almost perfect parallelism

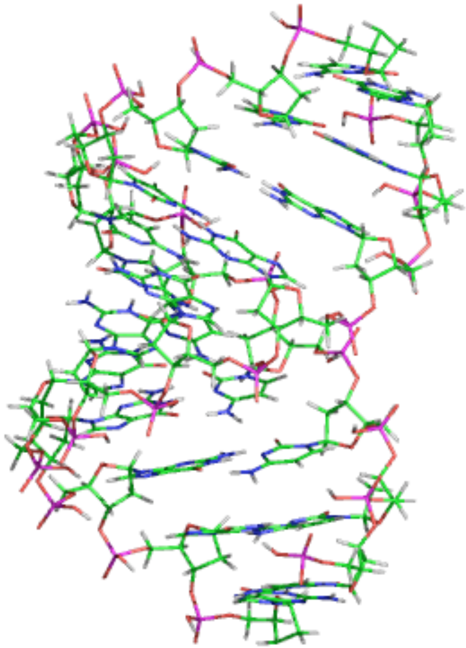
small *T***s**

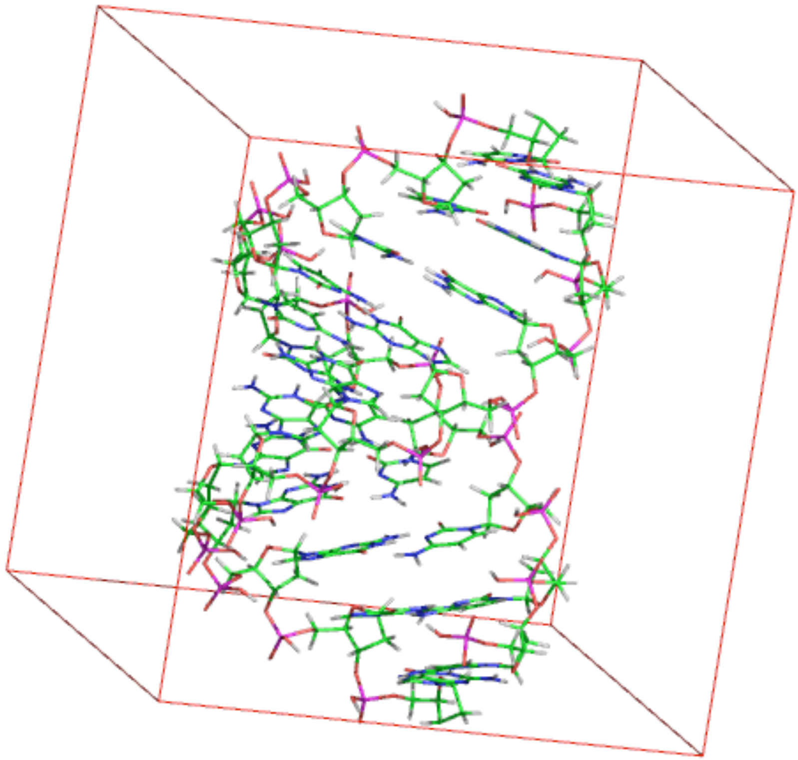
small *latency*

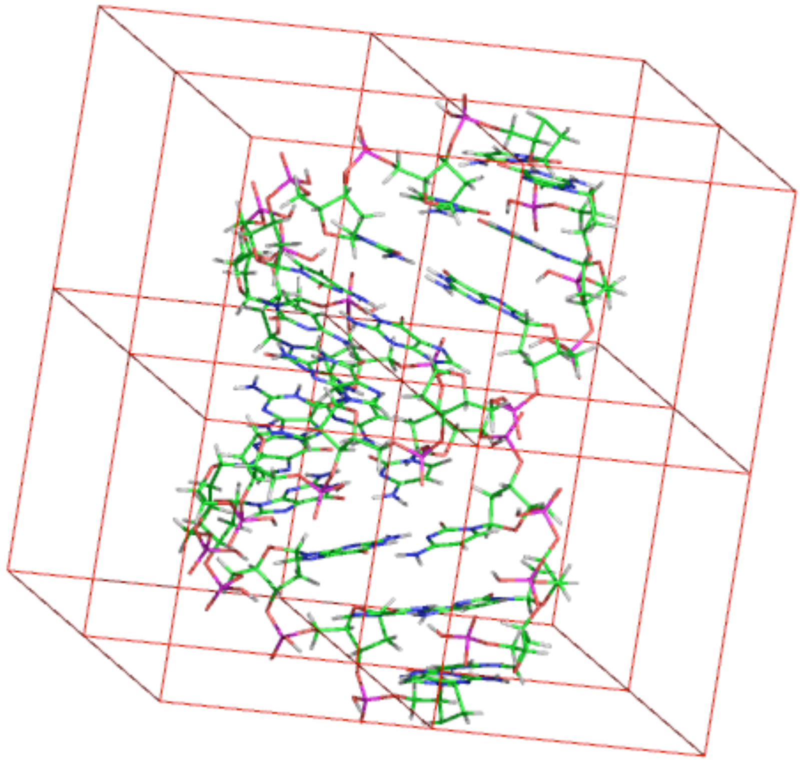
(Number of kPoints) **>** (Number of Nodes) `diagon` matrix parallelization

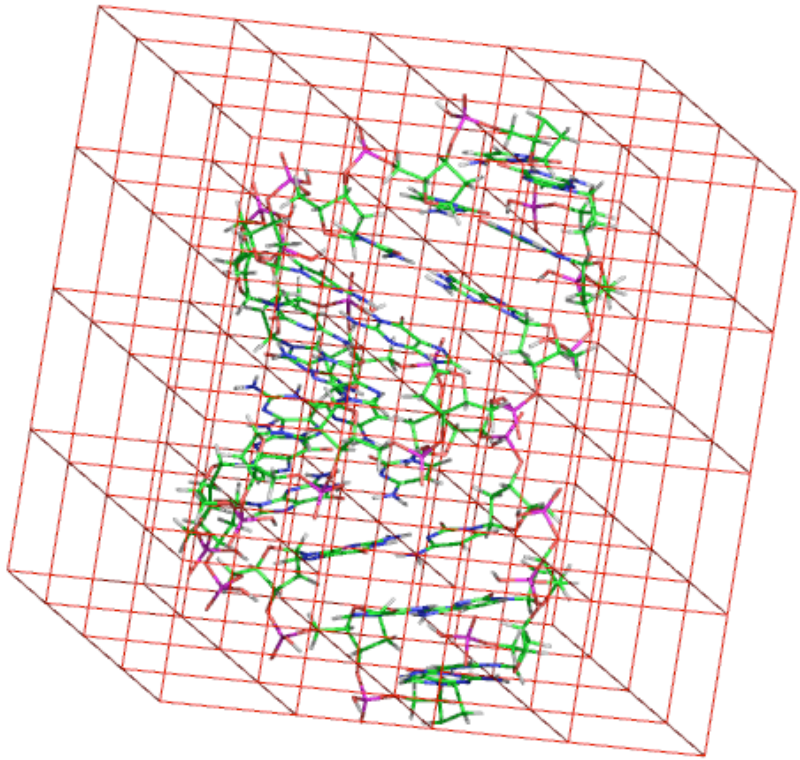
SCALAPACK

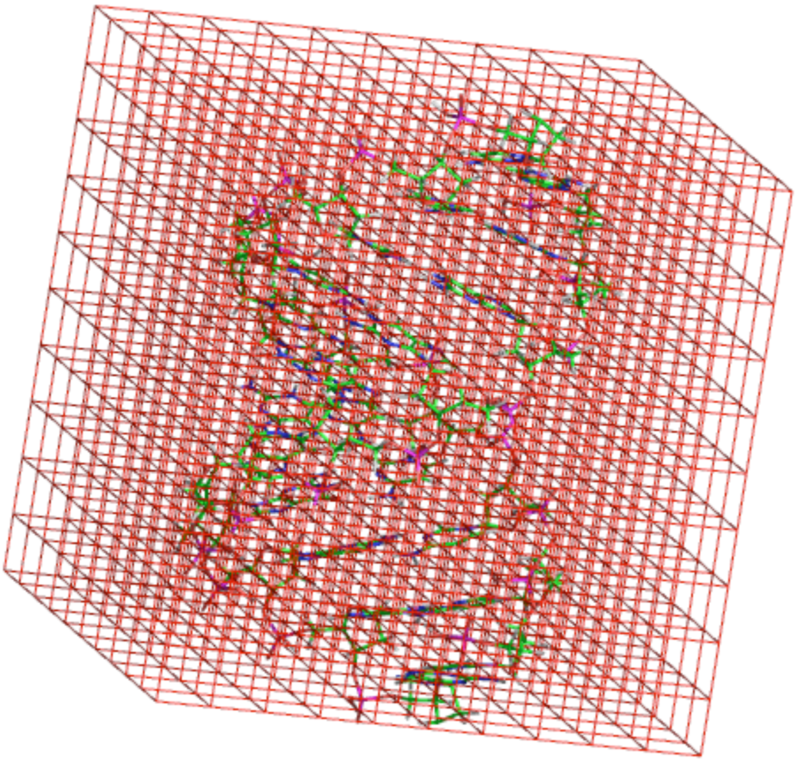
`ordern` parallelization

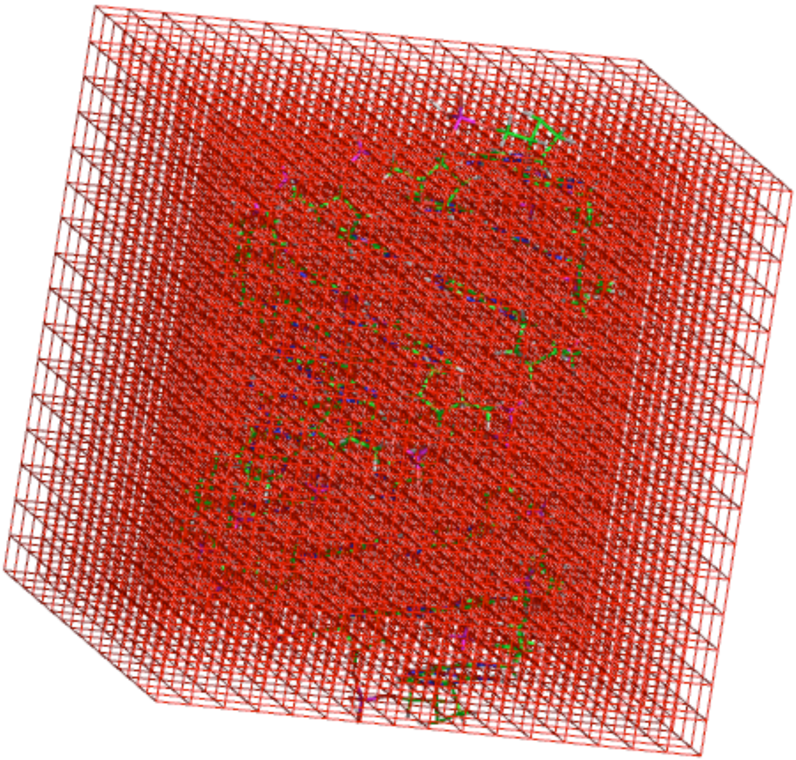


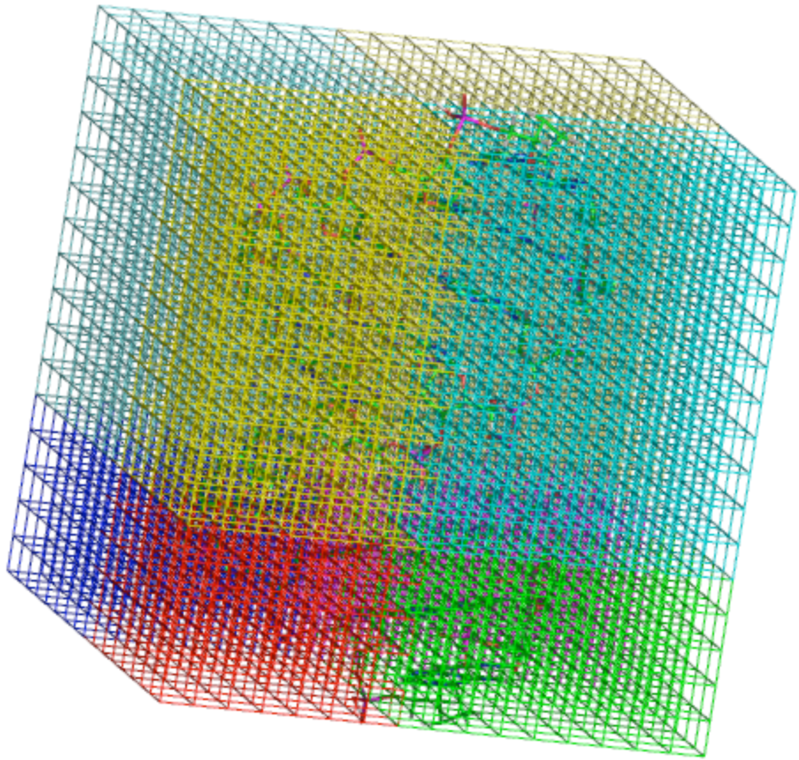


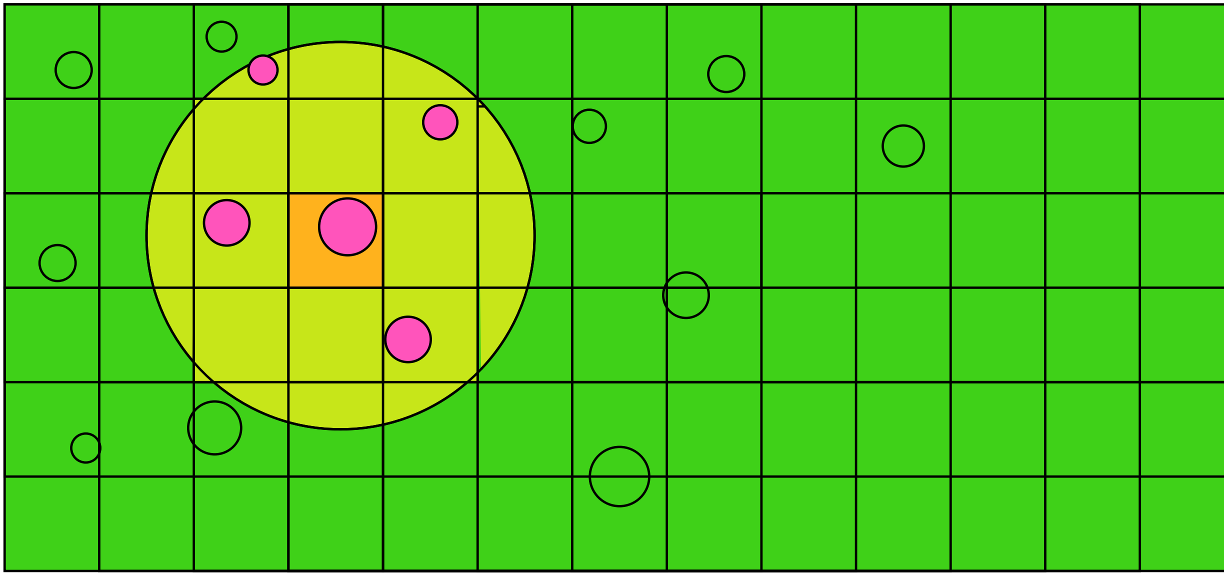












❖ How to use parallelism ***efficiently***

Constrained memory

Constrained time

Constrained politics

Serial?

`Diag.ParallelOverK`

No particular options

Orbital parallelization

`Blocksize`

(control load balancing/communications)

`Diag.Memory`

(only in case of failure)

Fine-tuning/debugging only:

```
Diag.Use2D
Diag.NoExpert
Diag.DivideAndConquer
Diag.AllInOne
```

OrderN parallelization

`RcSpatial`

(control load balancing/communications)

`ON.LowerMemory`

(does what it says!)

Grid portions of code

(for all parallel options)

`ProcessorY`

(control load balancing)

System orientation

(control load balancing)

Memory usage
(for all parallel options)

`DirectPhi`

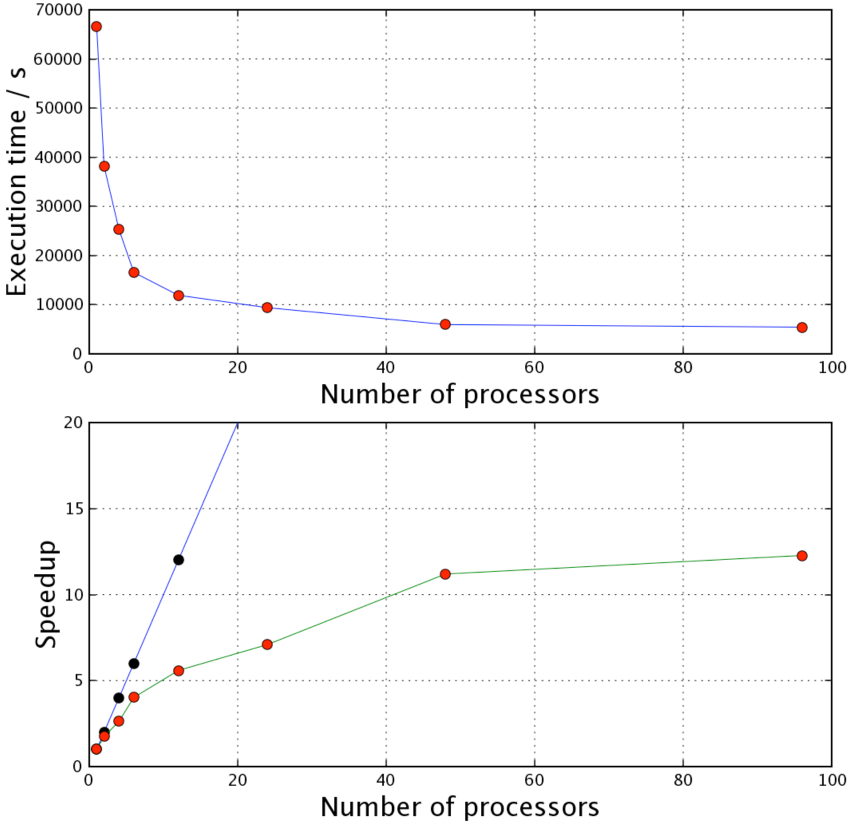
(cache orbital values)

`SaveMemory`

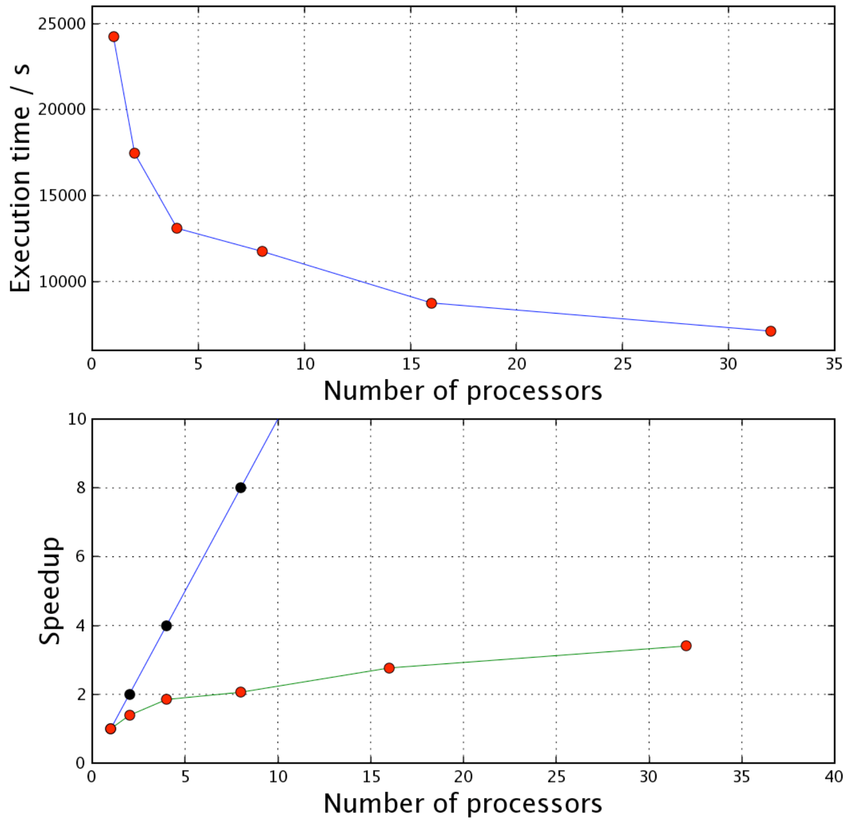
(does what it says!)

Trade off memory usage for speed

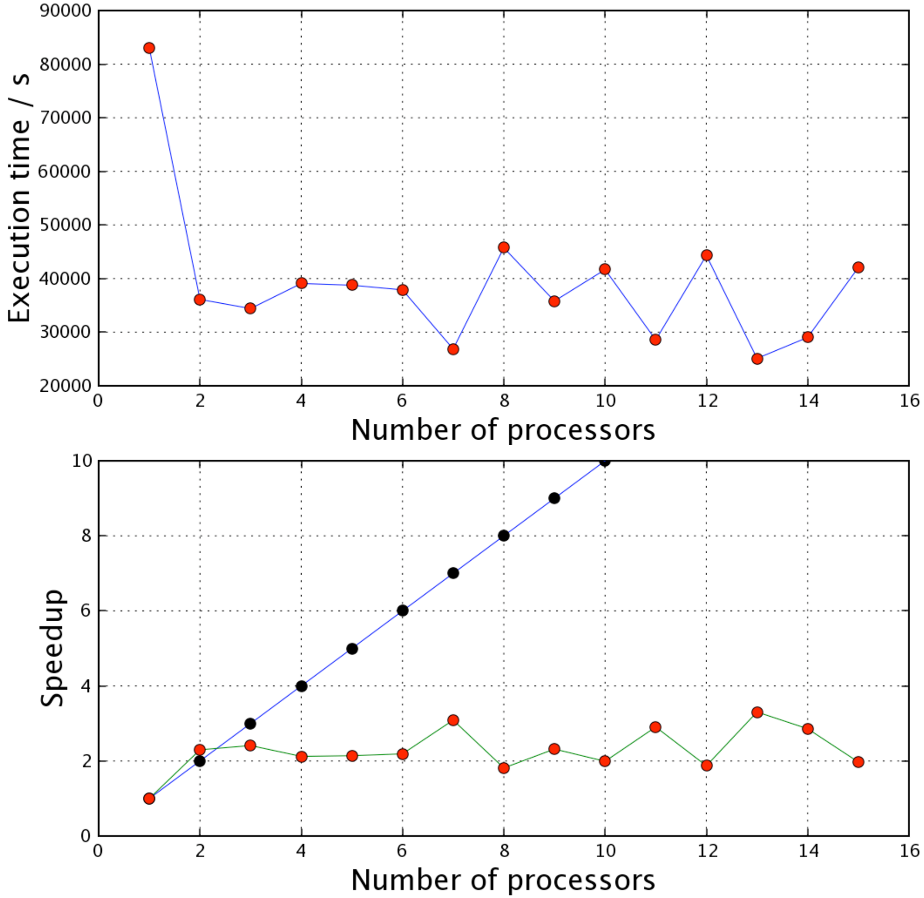
Sunfire 15K

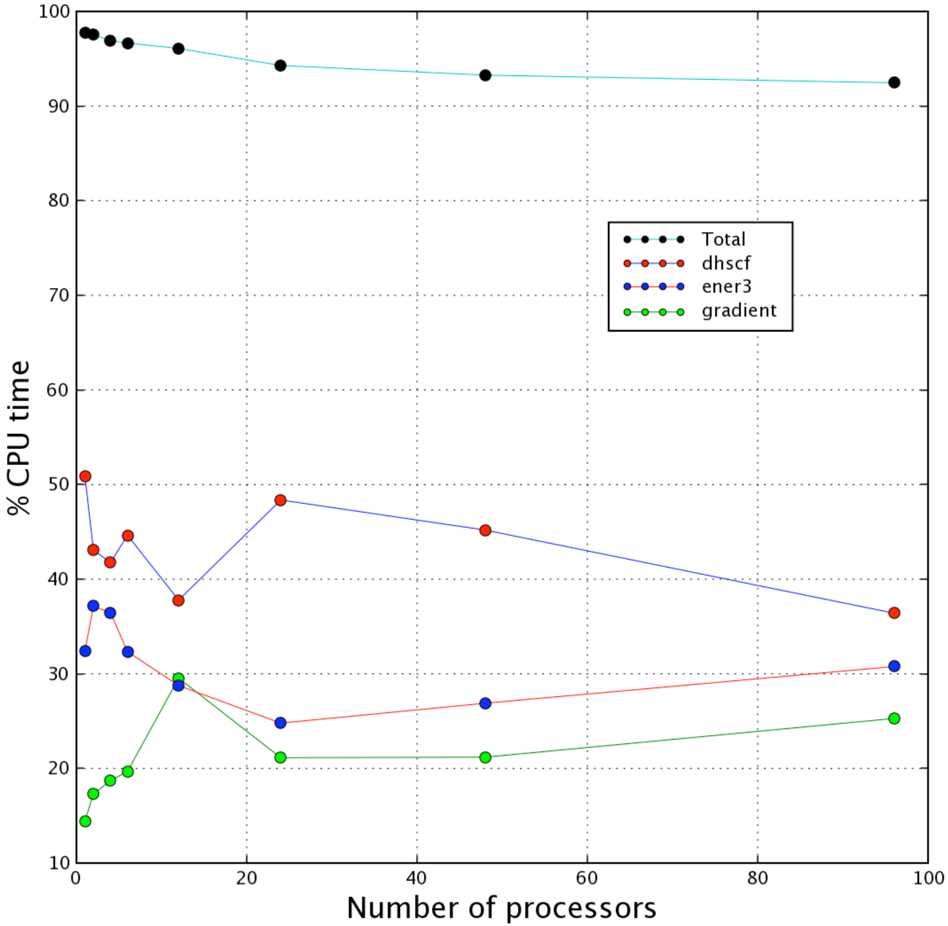

Execution time / s
70000
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
Number of processors
Speedup
20
15
10
5
0
0
20
40
60
80
100
Number of processors

Opteron cluster/Myrinet



Intel cluster/Gigabyte ethernet





% CPU time
Number of processors
Total
dhscf
ener3
gradient
100
90
80
70
60
50
40
30
20
10
0
20
40
60
80
100

Choose parallelization strategy (including serial!)

Look at timings:
scaling by nodes
- for whole program
- for each routine

Play with options
and observe
results

```
* Maximum dynamic memory allocated : Node    2 =   341 MB

* Maximum memory occured during globalise

timer: CPU execution times:
timer:  Routine        Calls   Time/call    Tot.time        %
timer:  siesta             1   31607.699   31607.699   100.00
timer:  Setup              1       0.680       0.680     0.00
timer:  bands              1       0.000       0.000     0.00
timer:  writewave          1       0.000       0.000     0.00
timer:  KSV_init           1       0.000       0.000     0.00
timer:  IterMD            10    3160.701   31607.009   100.00
timer:  hsparse           11      19.964     219.608     0.69
timer:  overfsm           20       1.608      32.151     0.10
timer:  IterSCF          110     285.160   31367.577    99.24
timer:  kinefsm           20       1.634      32.687     0.10
timer:  nlefsm            20       6.829     136.578     0.43
timer:  DHSCF            110      79.841    8782.531    27.79
timer:  DHSCF1             1       0.830       0.830     0.00
timer:  DHSCF2            10      55.891     558.908     1.77
timer:  REORD            680       0.012       8.352     0.03
timer:  POISON           120       4.515     541.752     1.71
timer:  DHSCF3           110      68.240    7506.443    23.75
timer:  rhoofd           110      20.537    2259.027     7.15
timer:  cellXC           110      30.186    3320.431    10.51
timer:  vmat             110      12.794    1407.332     4.45
timer:  setglobal         19       0.014       0.269     0.00
timer:  setglobalB        10       0.904       9.039     0.03
timer:  setglobalF        10       2.798      27.983     0.09
timer:  gradient         734      15.281   11216.541    35.49
timer:  globaliseF      1468       2.329    3419.667    10.82
timer:  globaliseB      1468       1.925    2825.608     8.94
timer:  globaliseC       743       0.611     454.166     1.44
timer:  ener3            634      15.340    9725.756    30.77
timer:  denmat           100      12.498    1249.832     3.95
timer:  DHSCF4            10      71.518     715.178     2.26
timer:  dfscf             10      63.564     635.640     2.01
```