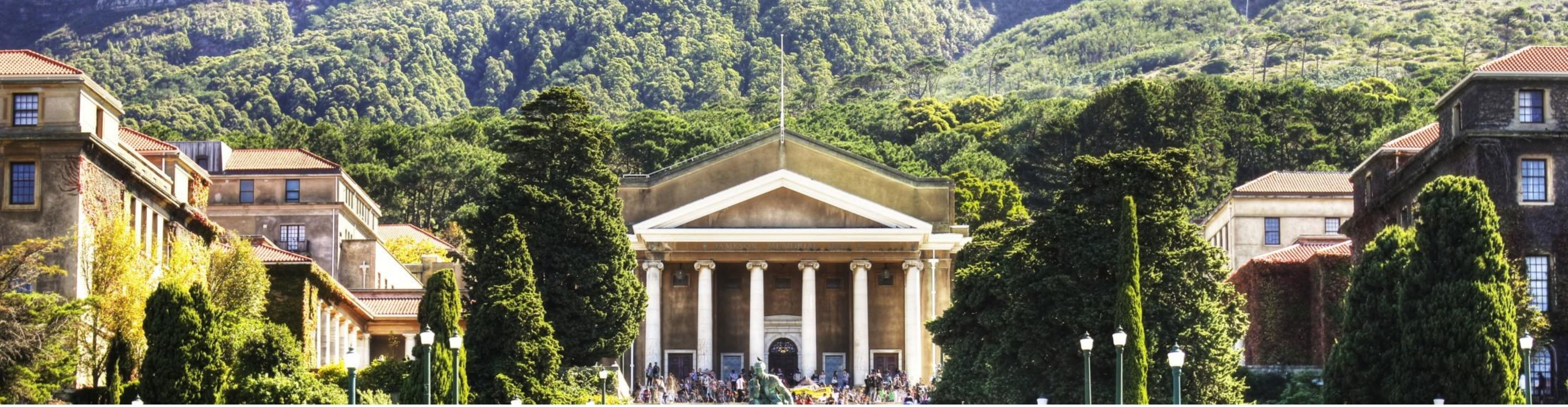


OpenMP in ONETEP

Karl Wilkinson and Nicholas Hine

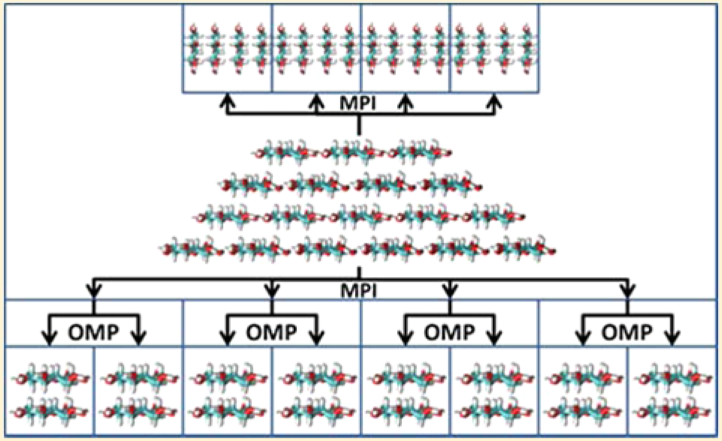
Hybrid MPI-OpenMP Parallelism in the ONETEP Linear-Scaling Electronic Structure Code: Application to the Delamination of Cellulose Nanofibrils

Karl A. Wilkinson,[†] Nicholas D. M. Hine,[‡] and Chris-Kriton Skylaris*,[†]

[†]School of Chemistry, University of Southampton, Southampton SO17 1BJ, United Kingdom

[‡]TCM Group, Cavendish Laboratory, University of Cambridge, JJ Thomson Avenue, Cambridge CB3 0HE, United Kingdom

ABSTRACT: We present a hybrid MPI-OpenMP implementation of Linear-Scaling Density Functional Theory within the ONETEP code. We illustrate its performance on a range of high performance computing (HPC) platforms comprising shared-memory nodes with fast interconnect. Our work has focused on applying OpenMP parallelism to the routines which dominate the computational load, attempting where possible to parallelize different loops from those already parallelized within MPI. This includes 3D FFT box operations, sparse matrix algebra operations, calculation of integrals, and Ewald summation. While the underlying numerical methods are unchanged, these developments represent significant changes to the algorithms used within ONETEP to distribute the workload across CPU cores. The new hybrid code exhibits much-improved strong scaling relative to the MPI-only code and permits calculations with a much higher ratio of cores to atoms. These developments result in a significantly shorter time to solution than was possible using MPI alone and facilitate the application of the ONETEP code to systems larger than previously feasible. We illustrate this with benchmark calculations from an amyloid fibril trimer containing 41,907 atoms. We use the code to study the mechanism of delamination of cellulose nanofibrils when undergoing sonification, a process which is controlled by a large number of interactions that collectively determine the structural properties of the fibrils. Many energy evaluations were needed for these simulations, and as these systems comprise up to 21,276 atoms this would not have been feasible without the developments described here.



J. Chem. Theory Comput. 2014, 10, 4782-4694

Motivation

Hardware is moving towards increasingly dense systems with large numbers of cores per socket.

The number of cores that could be used in a ONETEP calculation was subject to restrictions, reducing potential time to solution:

Restriction 1: Number of MPI processes may sometimes be less than the number of cores available on a node due to insufficient memory.

Restriction 2: Number of MPI processes must always be less than the number of atoms due to algorithmic restrictions.

MPI and OpenMP

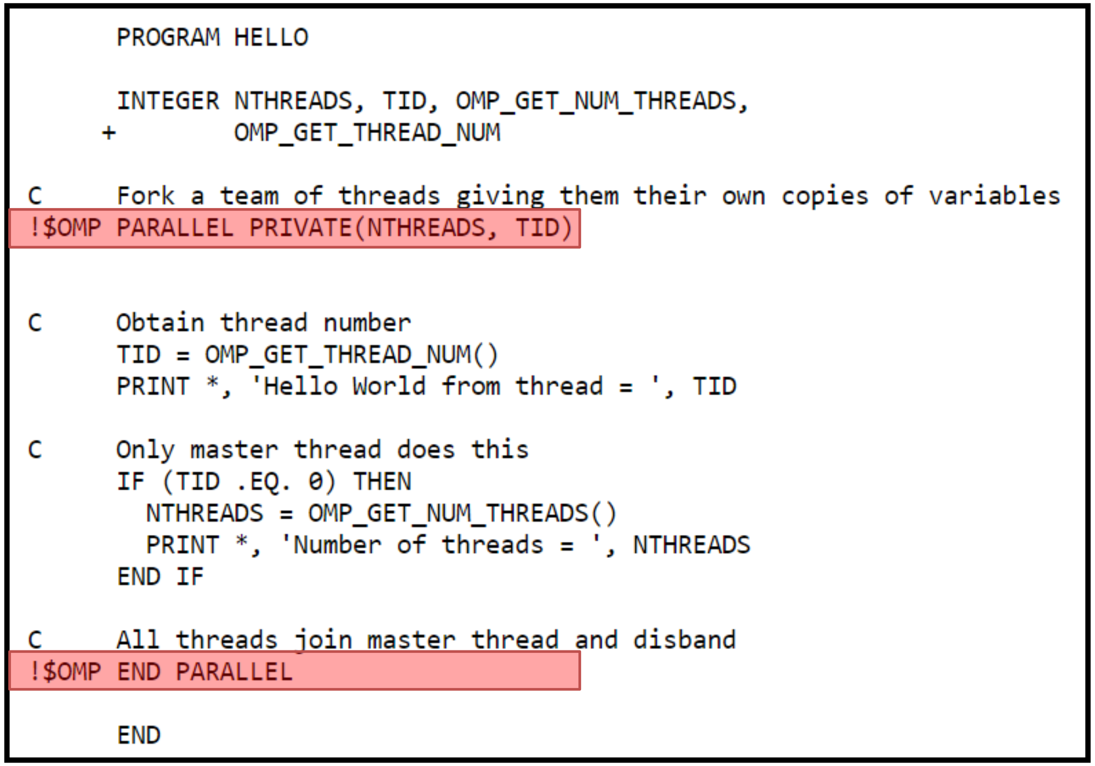
Message Passing Interface (MPI):

Each MPI “process” has its own memory space – communication of data between processes is controlled explicitly.

OpenMP:

Shared memory model, each thread can access the same memory.

“Pragma” based approach.



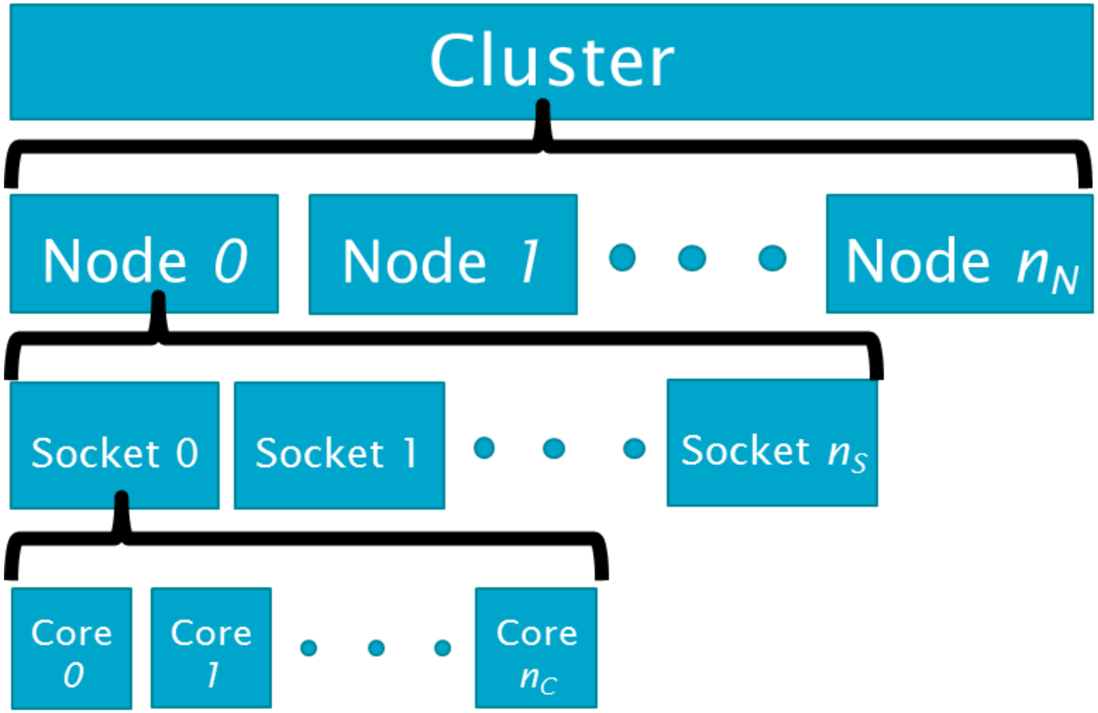
```
      PROGRAM HELLO

      INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS,
     +        OMP_GET_THREAD_NUM

C     Fork a team of threads giving them their own copies of variables
!$OMP PARALLEL PRIVATE(NTHREADS, TID)


C     Obtain thread number
      TID = OMP_GET_THREAD_NUM()
      PRINT *, 'Hello World from thread = ', TID

C     Only master thread does this
      IF (TID .EQ. 0) THEN
        NTHREADS = OMP_GET_NUM_THREADS()
        PRINT *, 'Number of threads = ', NTHREADS
      END IF

C     All threads join master thread and disband
!$OMP END PARALLEL

      END
```

MPI and OpenMP

Hybrid MPI/OpenMP reflects hardware layout:

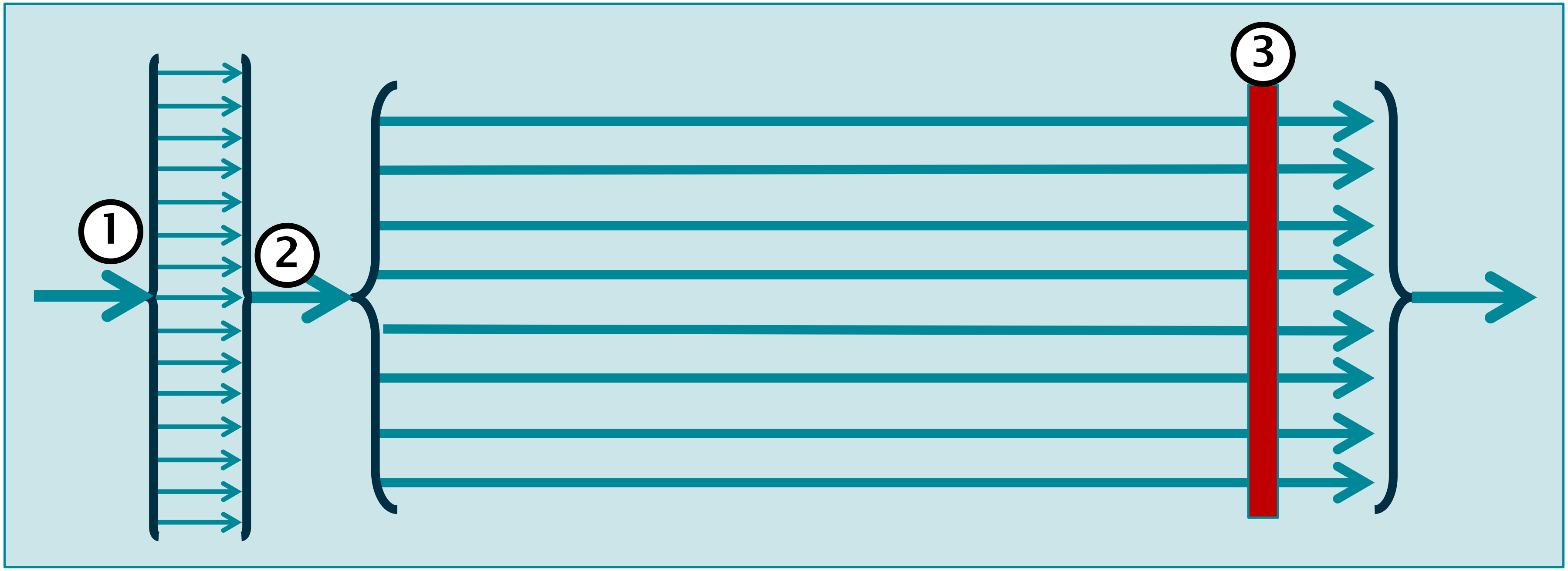
MPI to distribute work across nodes and

OpenMP to distribute work between cores on a node.



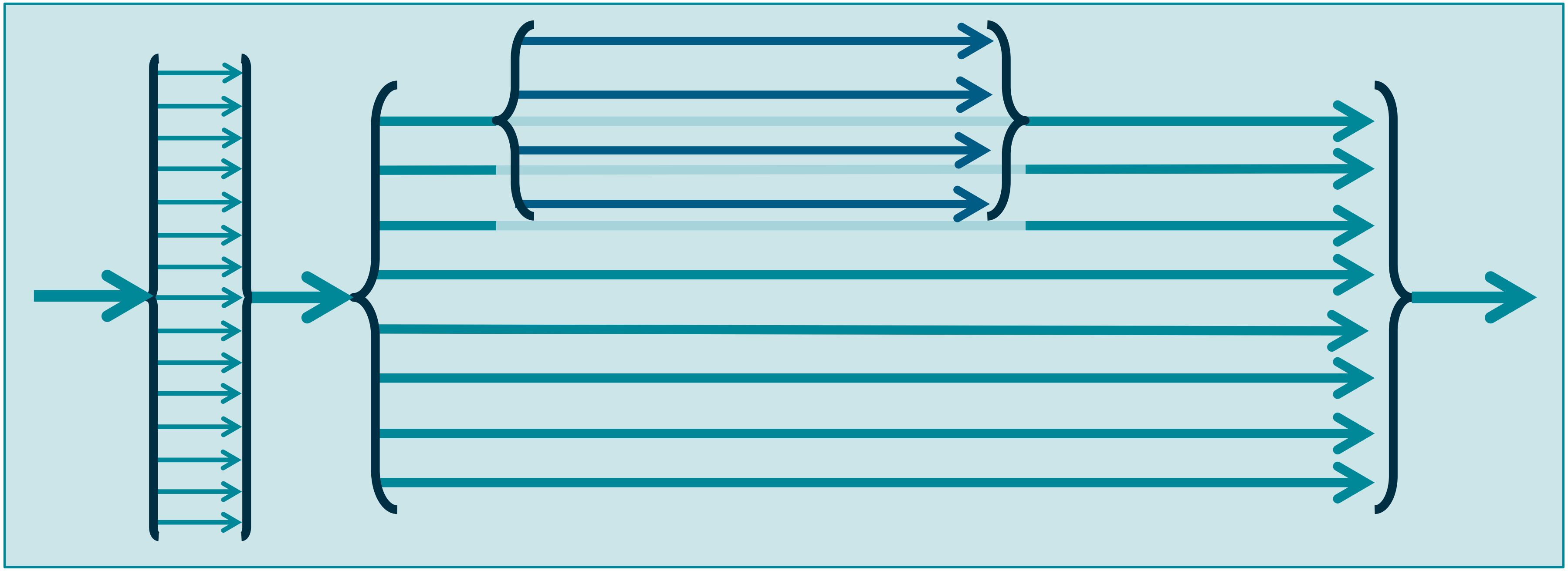
OpenMP paradigms in ONETEP

1. Thread creation	!$OMP PARALLEL DO
2. Thread merging	!$OMP END PARALLEL
3. Thread blocking	!$OMP CRITICAL

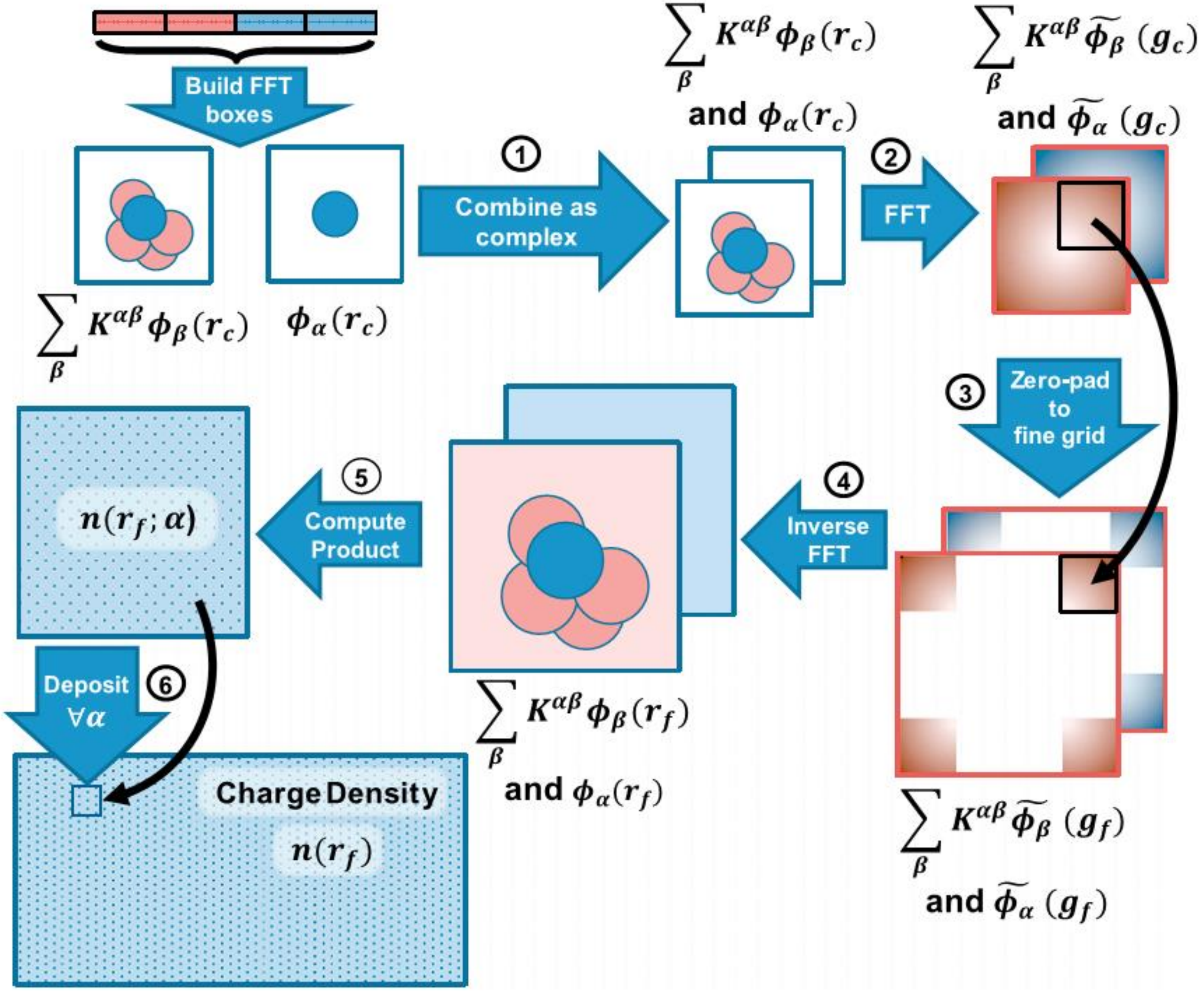


OpenMP Targets in ONETEP

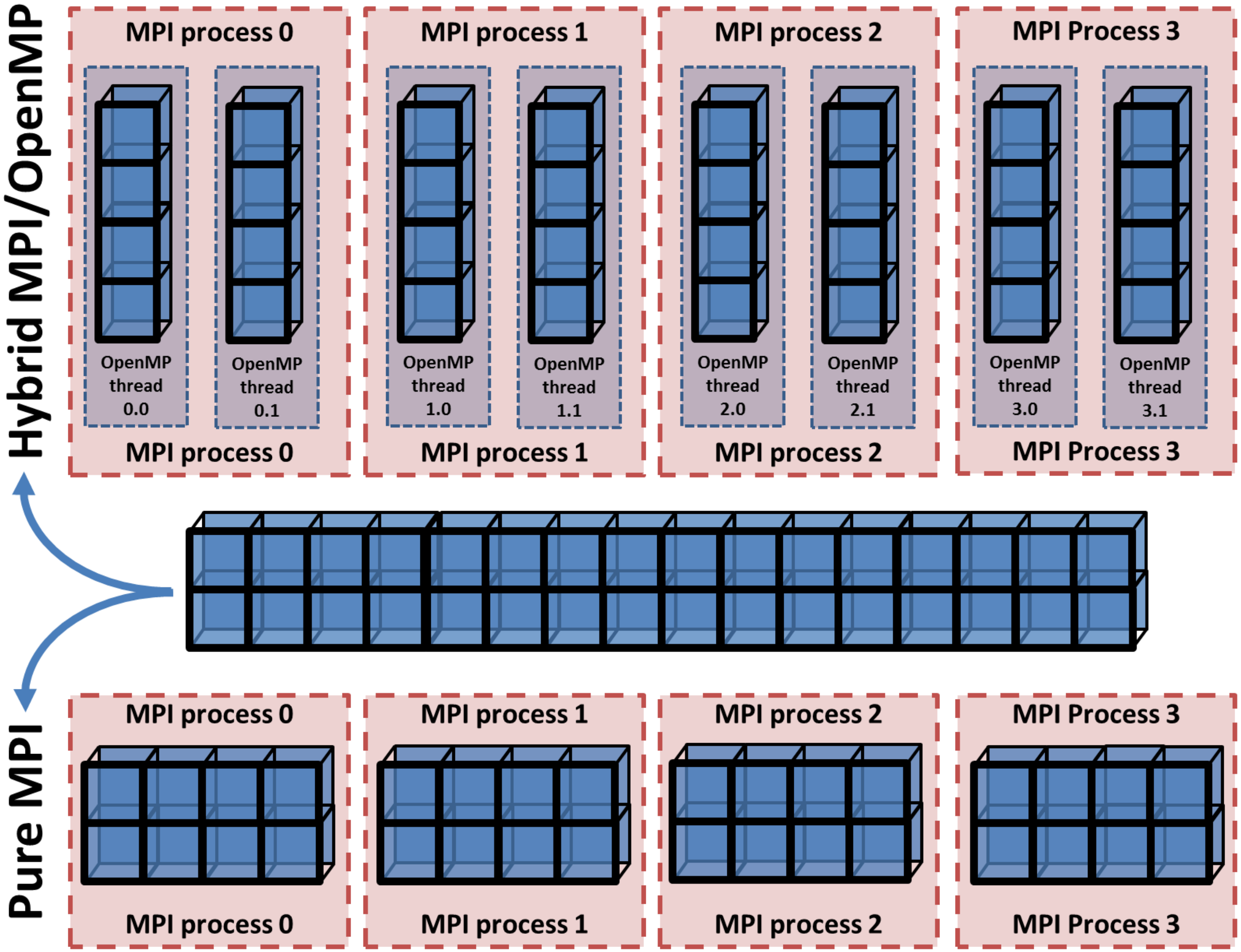
1. General parallel operations: sparse linear algebra and simulation cell FFTs
2. Operations on batches of FFT boxes
3. Operations on FFT boxes



FFT Box Operations


Build FFT boxes
$\sum_\beta K^{\alpha\beta}\phi_\beta(r_c)$
$\phi_\alpha(r_c)$
① Combine as complex
$\sum_\beta K^{\alpha\beta}\phi_\beta(r_c)$ and $\phi_\alpha(r_c)$
② FFT
$\sum_\beta K^{\alpha\beta}\widetilde{\phi_\beta}(g_c)$ and $\widetilde{\phi_\alpha}(g_c)$
③ Zero-pad to fine grid
$\sum_\beta K^{\alpha\beta}\widetilde{\phi_\beta}(g_f)$ and $\widetilde{\phi_\alpha}(g_f)$
④ Inverse FFT
$\sum_\beta K^{\alpha\beta}\phi_\beta(r_f)$ and $\phi_\alpha(r_f)$
⑤ Compute Product
$n(r_f;\alpha)$
Deposit $\forall\alpha$ ⑥
Charge Density
$n(r_f)$

FFT Box Operations



Hybrid MPI/OpenMP

MPI process 0

OpenMP thread 0.0

OpenMP thread 0.1

MPI process 0

MPI process 1

OpenMP thread 1.0

OpenMP thread 1.1

MPI process 1

MPI process 2

OpenMP thread 2.0

OpenMP thread 2.1

MPI process 2

MPI Process 3

OpenMP thread 3.0

OpenMP thread 3.1

MPI Process 3

Pure MPI

MPI process 0

MPI process 0

MPI process 1

MPI process 1

MPI process 2

MPI process 2

MPI Process 3

MPI Process 3

2 OpenMP threads per MPI process. 1 batch of 8 FFT box operations per MPI process.
8 CPU cores, 4 FFT box operations per CPU core.

32 FFT box operations - 4 batches of 8 FFT box operations

1 batch of 8 FFT box operations per MPI process.
4 CPU cores, 8 FFT box operations per CPU core.

Practical Considerations

Compilation

- Enable threading in MPI library
  - At time of MPI library compilation (--enable-mpi-thread-multiple for OpenMPI)

- FFT library
  - Enable threading during library compilation (--enable-openmp)
  - Link to threaded FFTW in ONETEP config file (-lfftw3_omp -lfftw3 –lm)

- Use OpenMP during ONETEP compilation:
    - Intel: -openmp
    - Gnu: -fopenmp
    - PGI: -mp

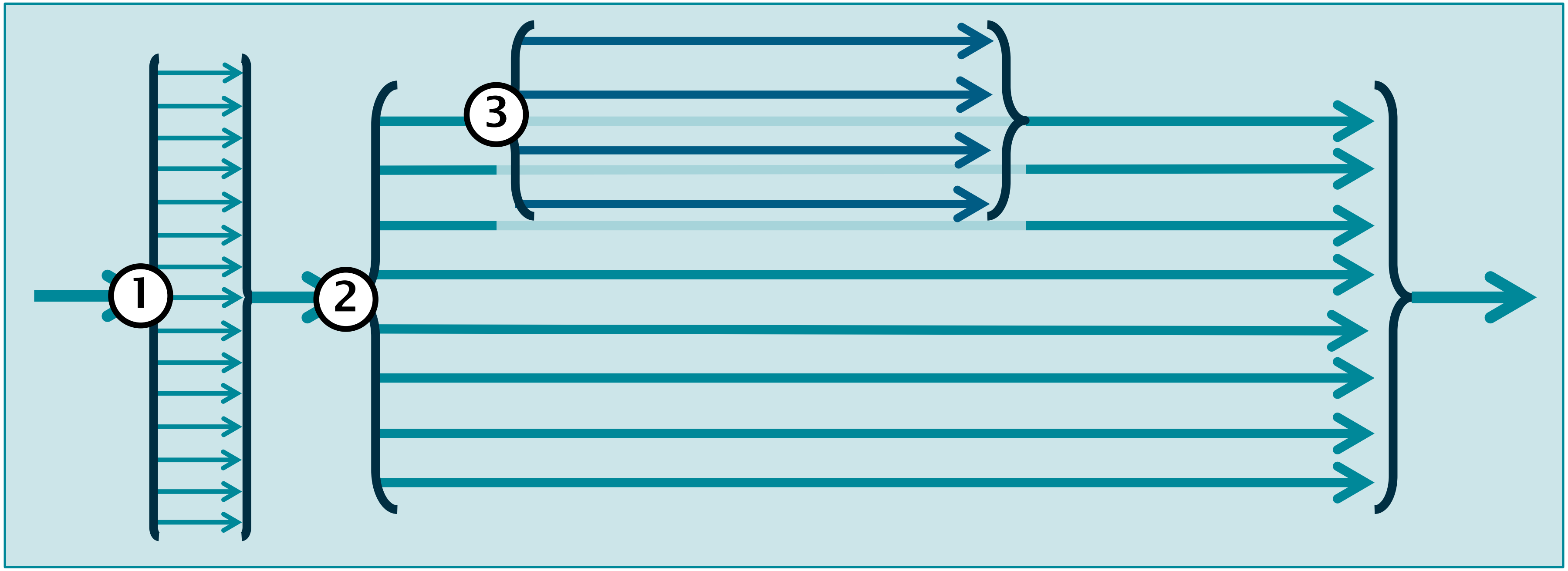
SCIENTIFIC COMPUTING

Practical Considerations

ONETEP Keywords

1. General parallel operations: threadsmax and threadspercellfft
2. Operations on FFT box batches: threadsnumfftboxes and fftboxbatchsize
3. *Operations on FFT boxes threadsperfftbox*

4. Communications: comms_group_size



SCIENTIFIC COMPUTING

Practical Considerations

Job Submission

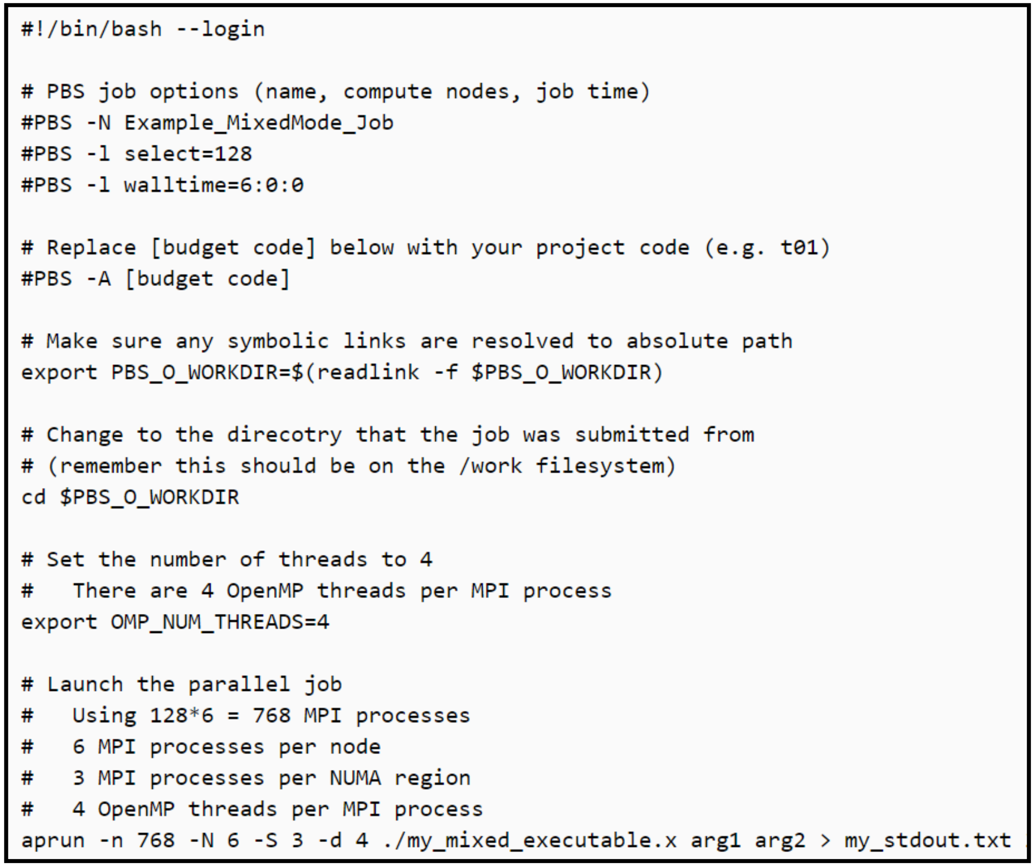
Machine dependent, typically well documented

ARCHER example:

- **128 Nodes, 3072 cores**
- **4 OpenMP threads per MPI process**

Avoid:

- **Oversubscription**
- **Crossing NUMA regions (Non-uniform memory access)**



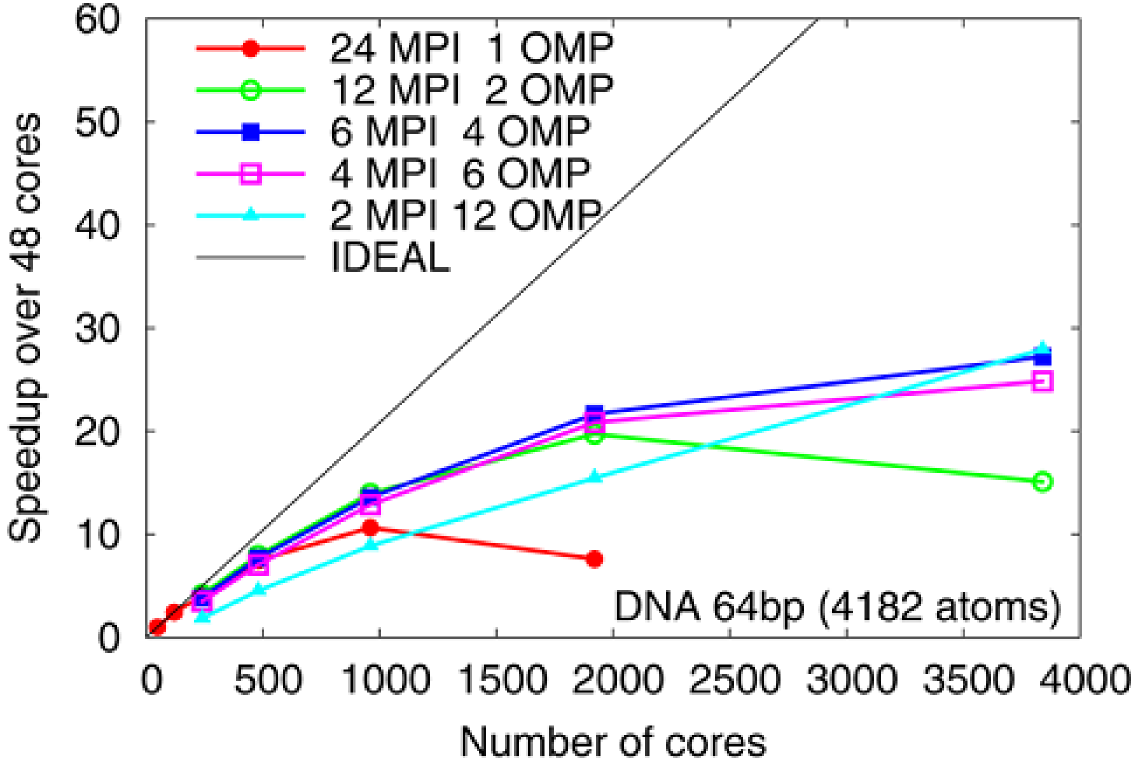
```
#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N Example_MixedMode_Job
#PBS -l select=128
#PBS -l walltime=6:0:0

# Replace [budget code] below with your project code (e.g. t01)
#PBS -A [budget code]

# Make sure any symbolic links are resolved to absolute path
export PBS_O_WORKDIR=$(readlink -f $PBS_O_WORKDIR)

# Change to the direcotry that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

# Set the number of threads to 4
#   There are 4 OpenMP threads per MPI process
export OMP_NUM_THREADS=4

# Launch the parallel job
#   Using 128*6 = 768 MPI processes
#   6 MPI processes per node
#   3 MPI processes per NUMA region
#   4 OpenMP threads per MPI process
aprun -n 768 -N 6 -S 3 -d 4 ./my_mixed_executable.x arg1 arg2 > my_stdout.txt
```

Strong Scaling

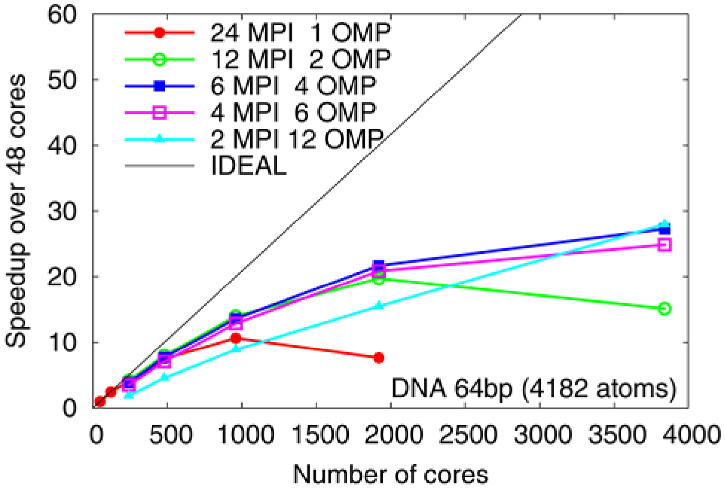
- Increase number of cores for a fixed system size
- Direct measure of performance
- Tested for different ratios of MPI processes to OpenMP threads for multiple systems containing ~4000 atoms

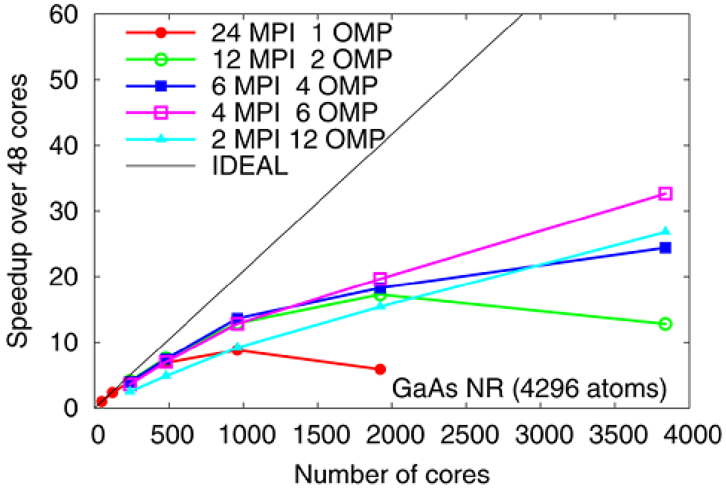
Calculations performed on ARCHER using production quality settings:
50 a0 density cutoff
8.0 a0 NGWF radii
700-800 eV KE cutoff

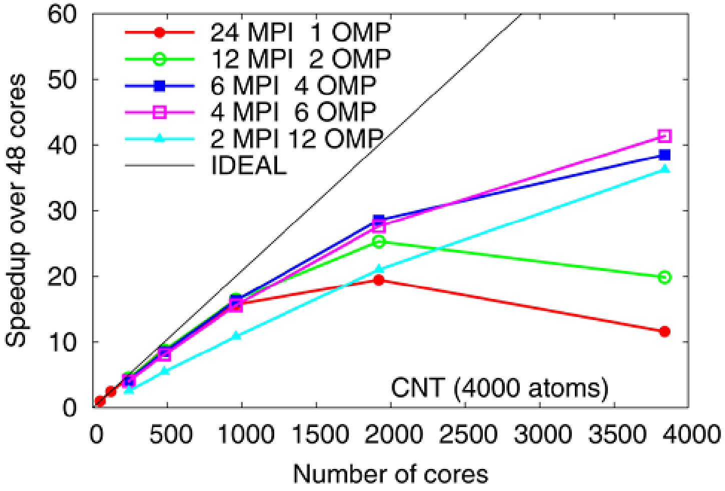
Limited to 1 NGWF iteration

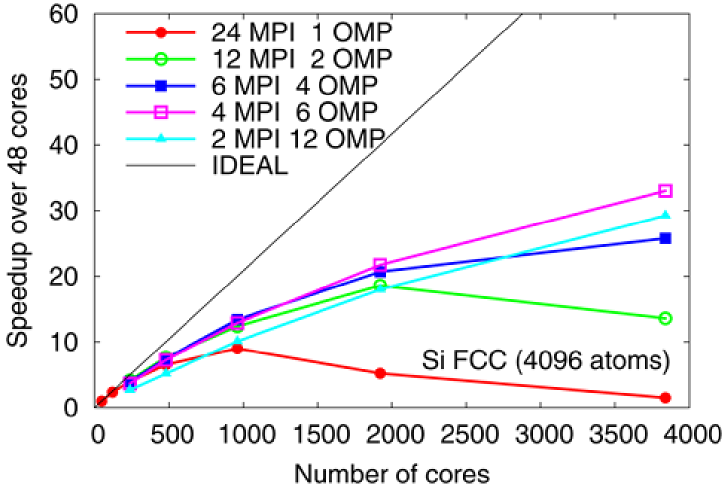


Strong Scaling



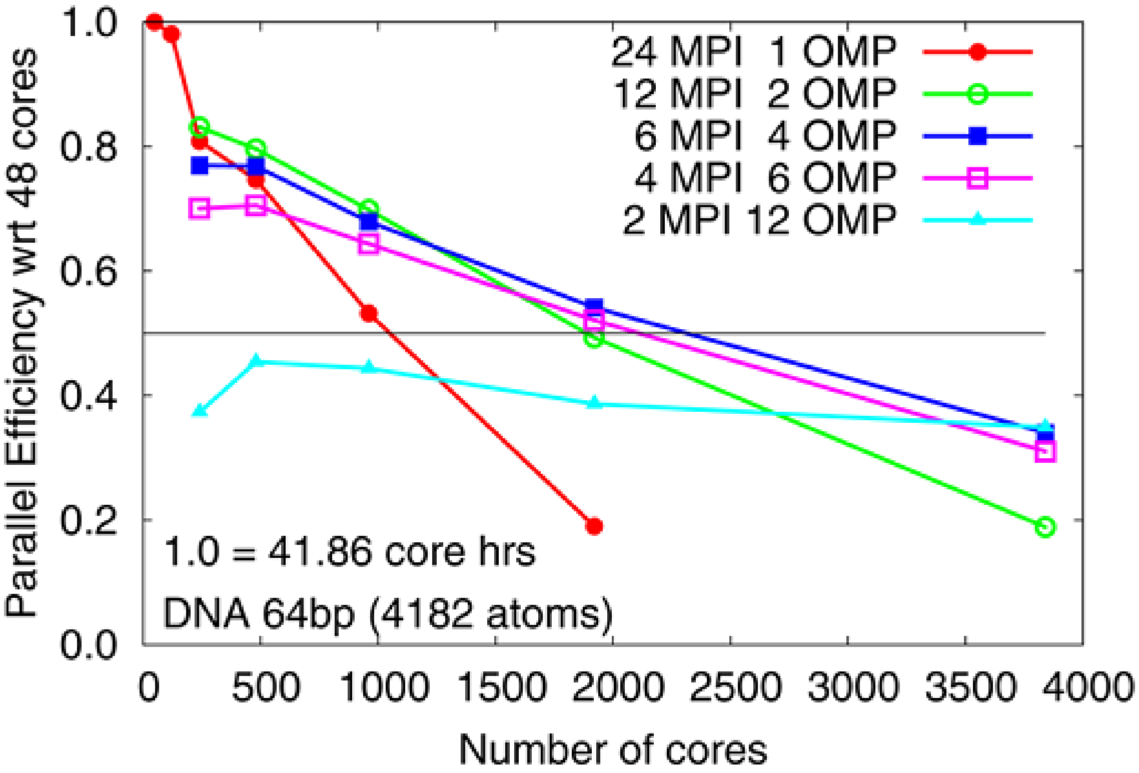




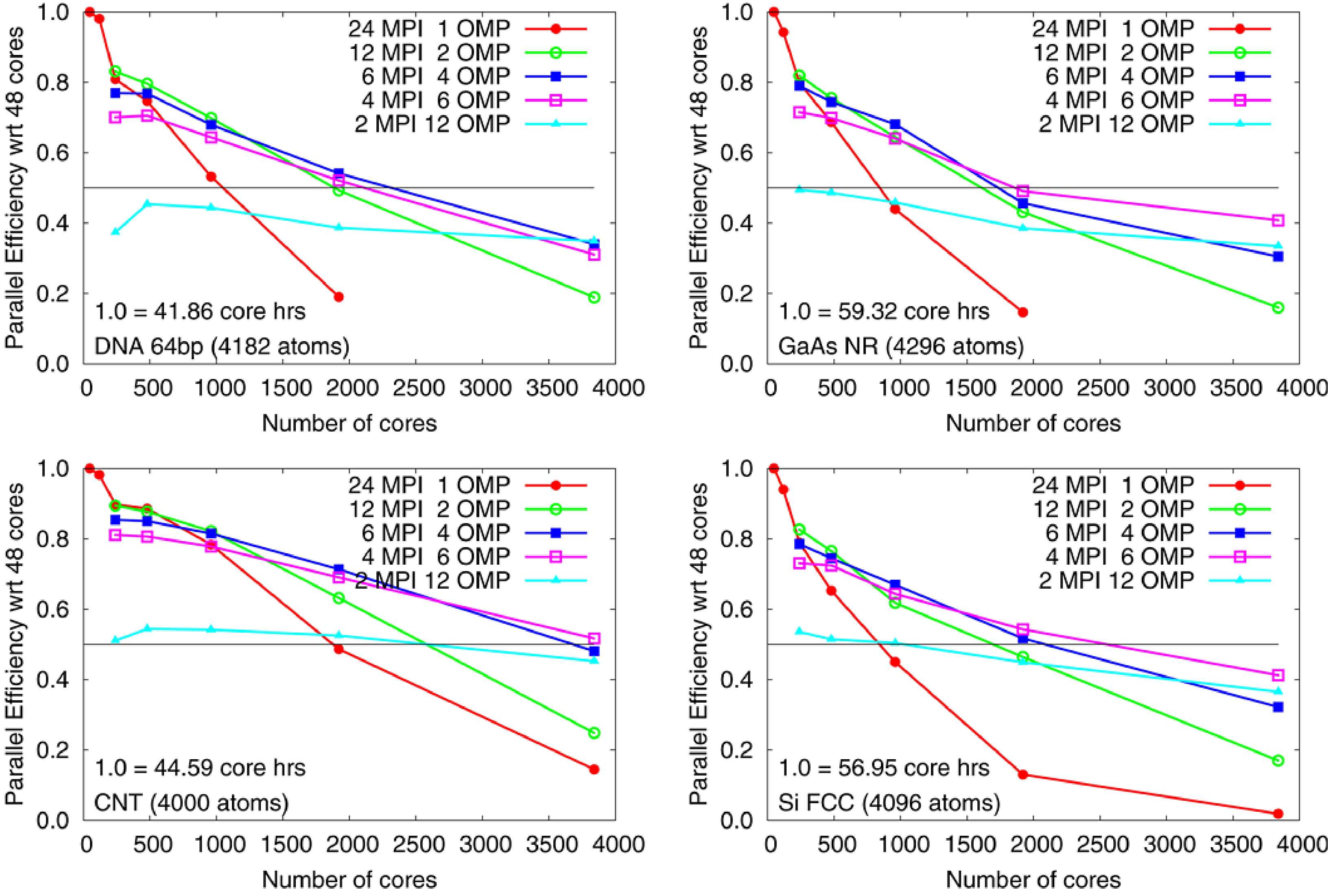


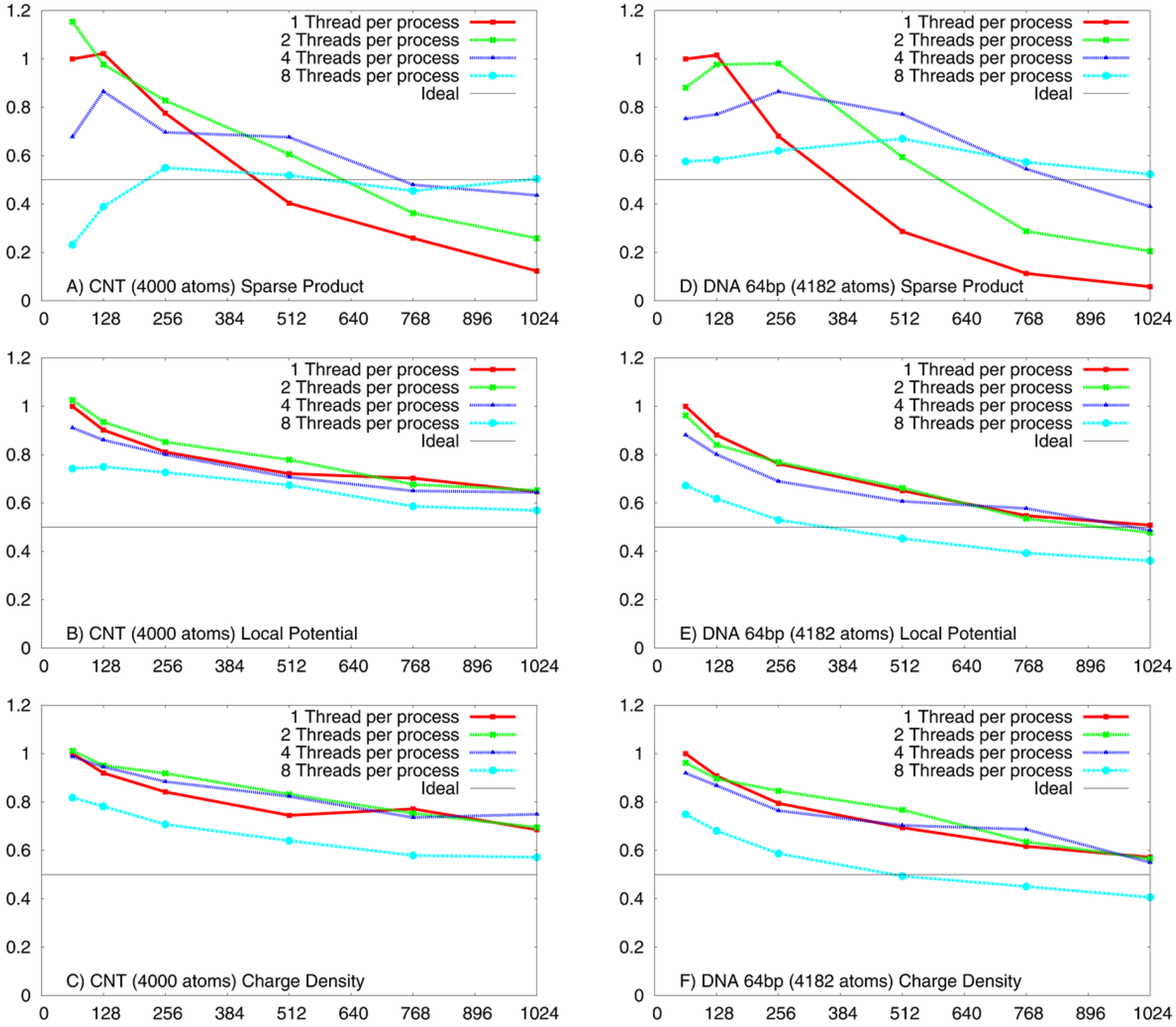
Parallel Efficiency

- Compare performance against the ideal, given the number of cores used
- Target is >50%



Parallel Efficiency




1.2
1
0.8
0.6
0.4
0.2
0
0
128
256
384
512
640
768
896
1024
1 Thread per process
2 Threads per process
4 Threads per process
8 Threads per process
Ideal
A) CNT (4000 atoms) Sparse Product
D) DNA 64bp (4182 atoms) Sparse Product
B) CNT (4000 atoms) Local Potential
E) DNA 64bp (4182 atoms) Local Potential
C) CNT (4000 atoms) Charge Density
F) DNA 64bp (4182 atoms) Charge Density

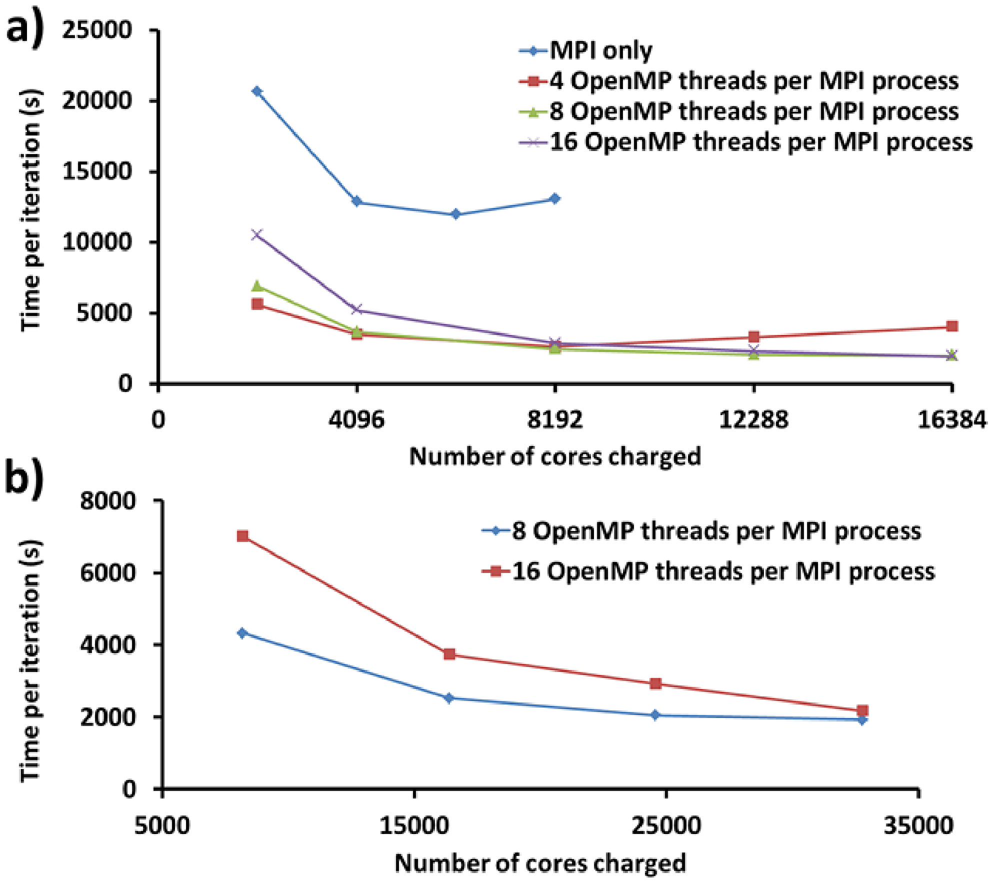
Performance: Large Systems

a) 13,969 atom beta amyloid fibril on BlueJoule

- Large difference in performance as pure MPI cannot use all cores on a node due to memory restriction.
- Pure MPI can only be executed on up to 8192 cores.

b) 41,907 atom beta amyloid fibril on BlueJoule

- Pure MPI does not scale to this level



Summary

- Eases restrictions on number of atoms per core
- Significantly increases the scale of ONETEP calculations

- Parallel efficiency is system dependent: More ordered = more efficient

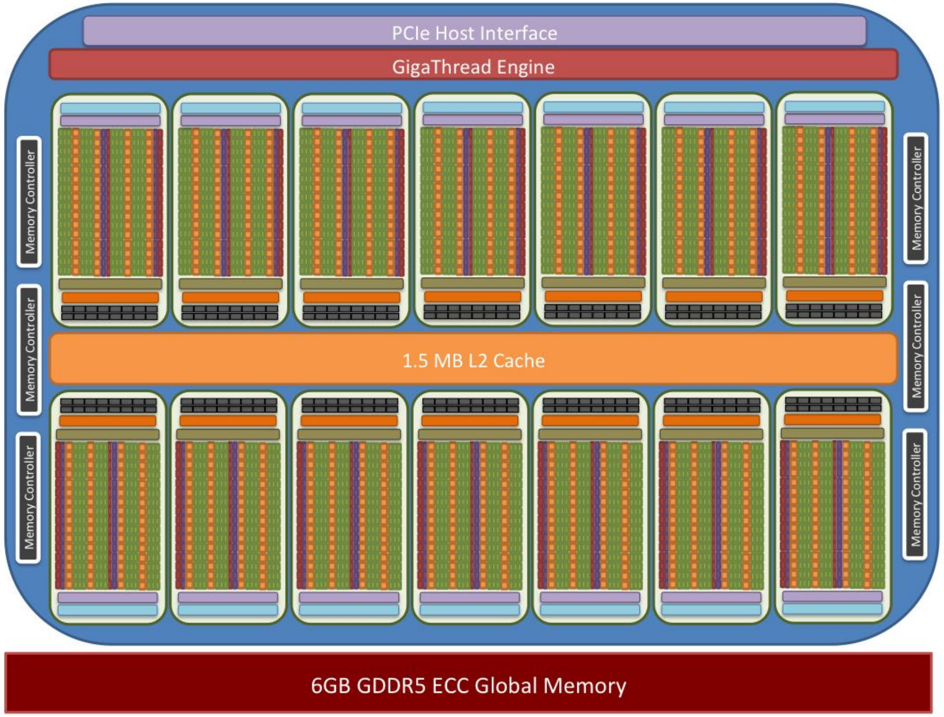
- Reduces time to solution for a given problem, but requires more cores.
  - May be more efficient to run jobs in parallel, each on fewer cores.
  - Overall benefit of using large numbers of cores depends on costing model at HPC center: Develop execution model.

- Still work to do:
  - Further optimization
  - “Data parallel” code
  - Intel Xeon Phi

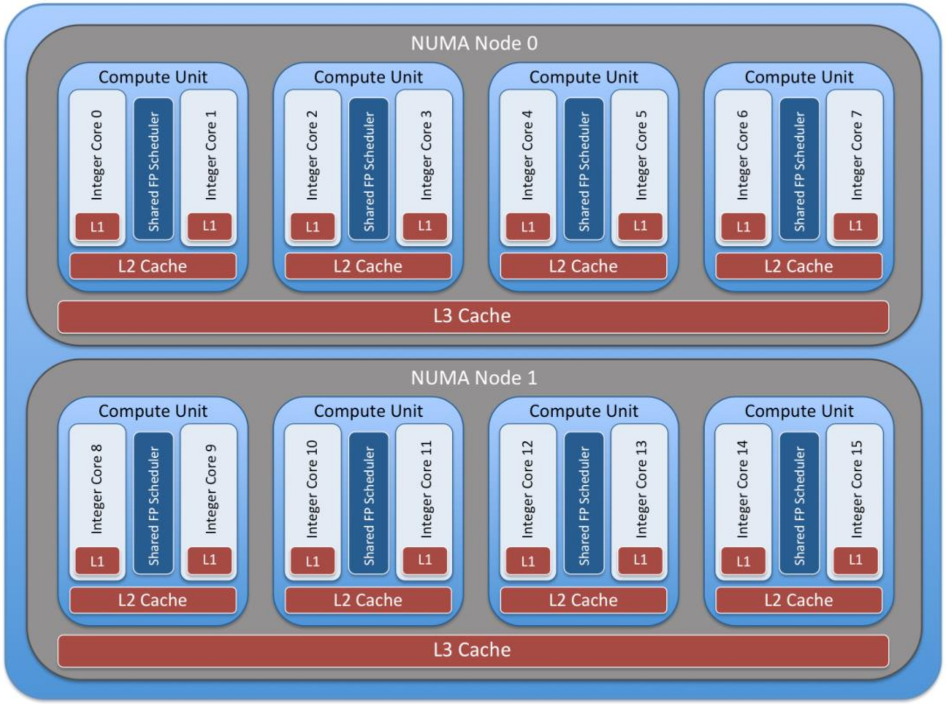
Charging on TITAN: The TITAN hour

- One node hour = 30 TITAN hours

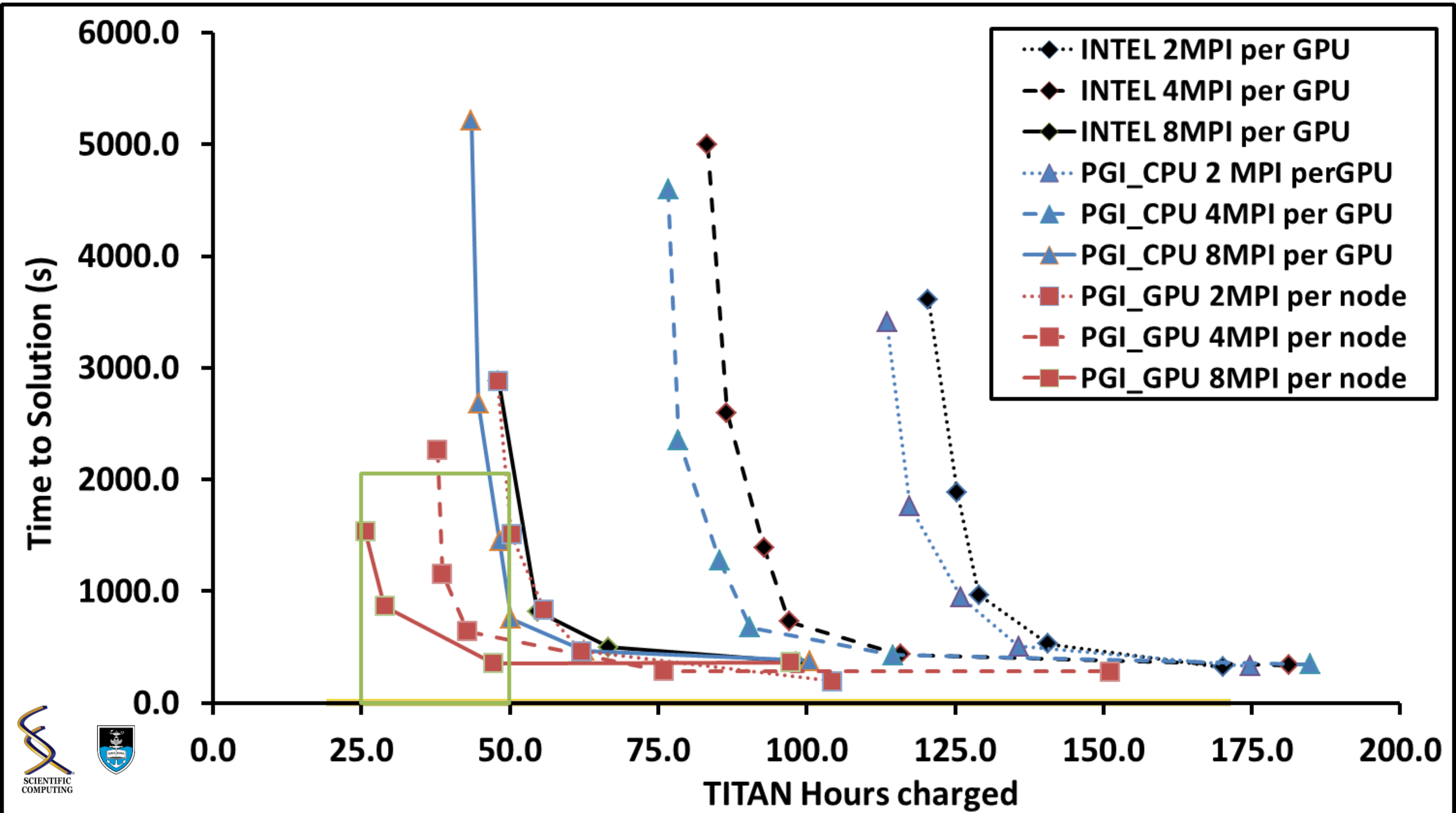
- 16 CPU compute units (Shared FP scheduler)
- 14 GPU compute units (Streaming Multiprocessors)



AMD Opteron™ 6274 (Interlagos) CPU

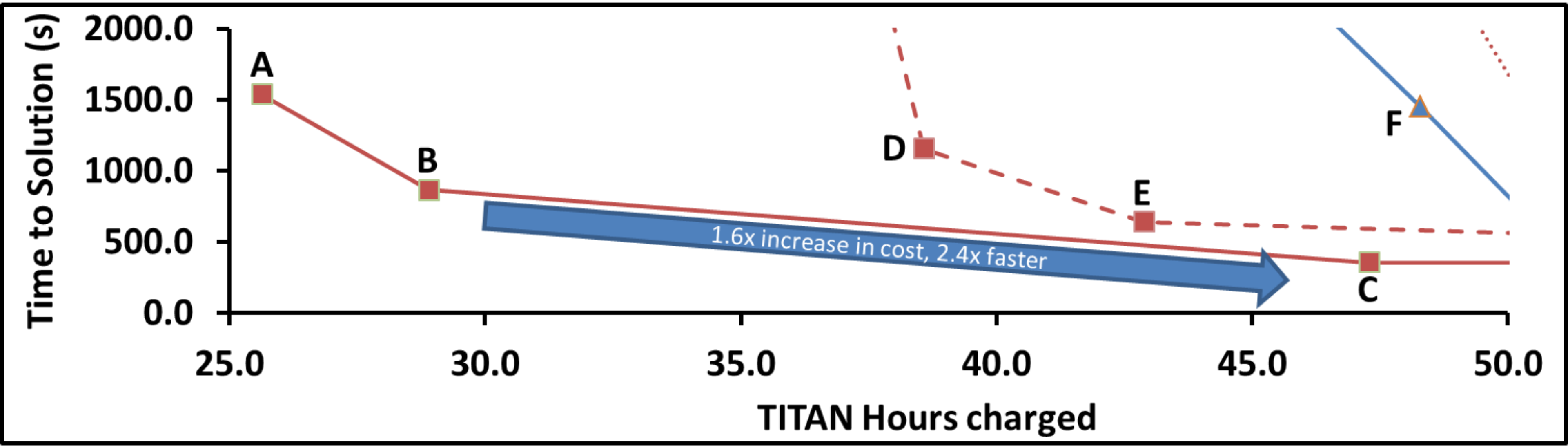


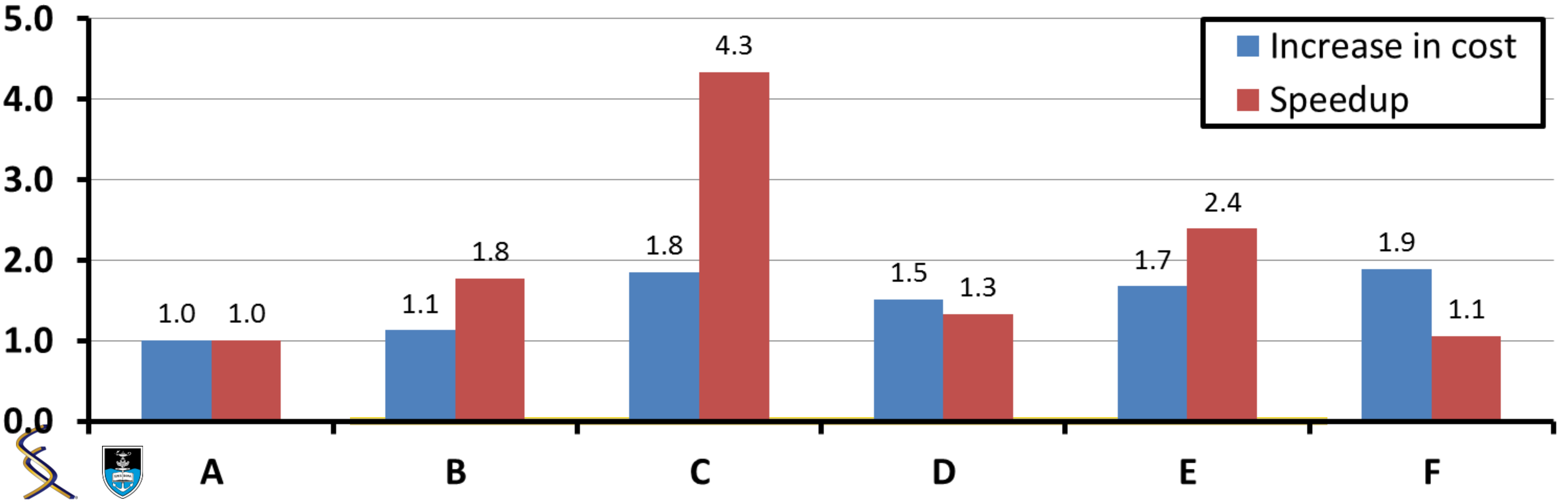
Cost vs Time



SCIENTIFIC COMPUTING

Cost vs Time





SCIENTIFIC COMPUTING