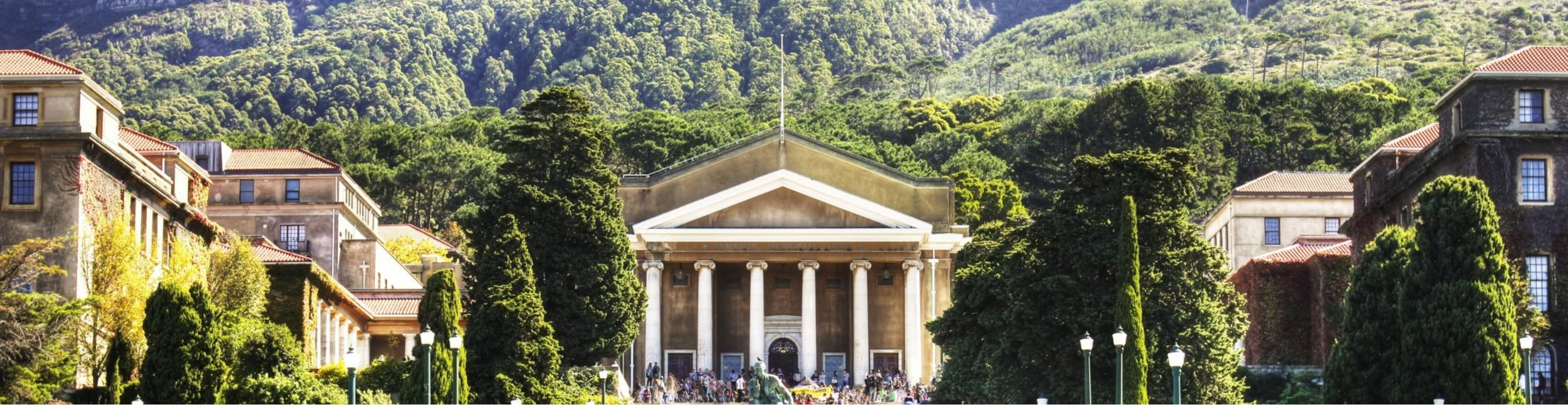


ONETEP on Graphical Processing Units

Karl Wilkinson

SCIENTIFICOMPUTING
SCIENCE THROUGH COMPUTING

University of Cape Town · iYunivesithi yaseKapa · Universiteit van Kaapstad

Graphical Processing Units

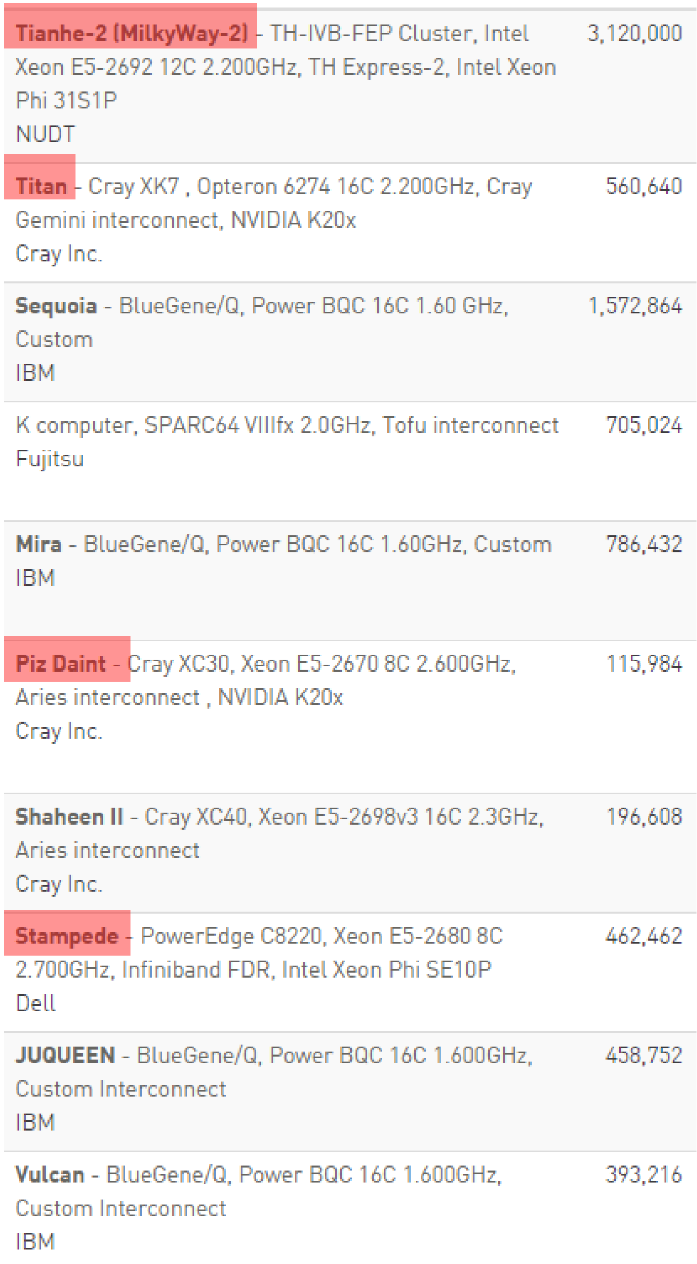
- Originally designed as 3D graphics accelerators aimed at the gaming market
- Offload suitable work from the CPU (Parallel hardware for parallel algorithms)
- Massively parallel, disruptive technology
- Pre-2006
  - Initial non-graphic software developments with Graphics application programming interface (API)
- Emergence of GPGPU: 2006
  - Unified shaders in the GeForce 8 series (G80) moved from separate functional units to a collection of stream processors
  - CUDA released
  - First Teslas released (C/S/D870)

SCIENTIFIC
COMPUTING

Heterogeneous Architectures

- Heterogeneous: Multiple distinct types of processors
  - More than 10% of top 500 supercomputer use some form of accelerator
  - Typically Central Processing Units (CPUs) coupled with Graphical Processing Units (GPUs)
  - Common in high powered workstations
  - Intel Xeon Phis emerging

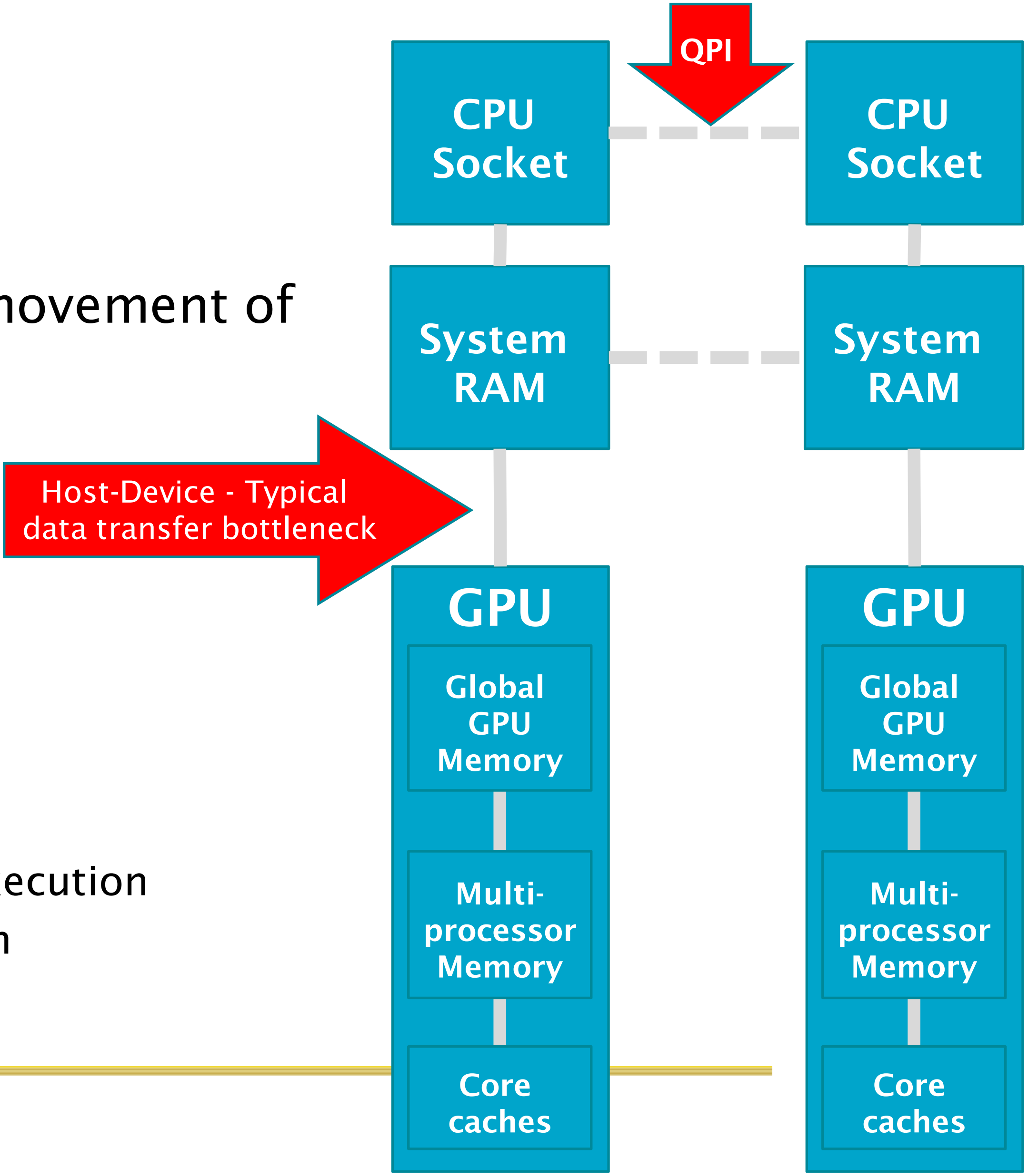
- Development considerations
  - Generally need to adapt code to take advantage of the architecture
  - Single code will contain multiple algorithms - execute on most suitable processing unit
  - May result in new bottlenecks (PCIe data transfer)



Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000
Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640
Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864
K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024
Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432
Piz Daint - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984
Shaheen II - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	196,608
Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462
JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752
Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216

Memory in Heterogeneous Machines

- Complex memory hierarchy
- GPUs designed to hide the movement of data on the GPU itself



- Major issues:
  - Host – device data transfer
  - CPU speed for data parallel execution
  - GPU speed for serial execution

Choice of language

- Hardware Considerations
  - Multiple hardware models already available
  - Uncertainty over future hardware models
- Development Considerations
  - Complexity of hardware - average developer is a domain specialist
  - Porting existing rather than writing from scratch
  - Avoid mixing languages
  - Avoid reinventing the wheel
  - Ensure maintainability of code

- Hybrid OpenACC/CUDA Fortran
  - OpenACC for Kernels
  - CUDA Fortran for data transfers
- CUFFT library
- Straightforward use of pragmas.

Scope of the Developments

- Focus on atom localized “FFT box” operations
  - Core algorithms that will benefit full range of functionality
- Maintain data structures and current communications schemes
- **Minimize disruption to rest of code**

J. Comput. Chem. 2013; 34(28), 2446-59.

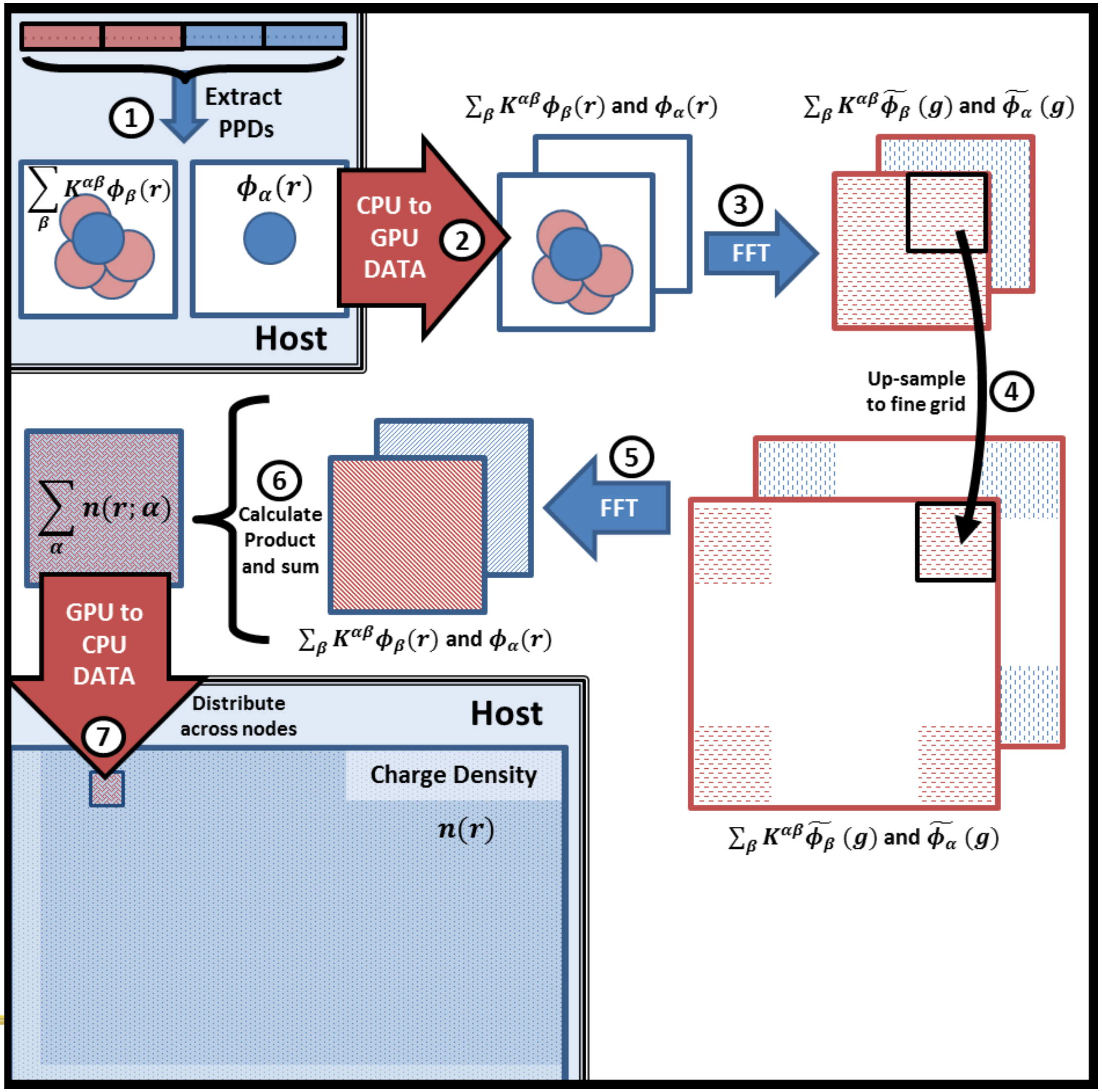
Charge Density (36% of runtime)

Diagonal elements of the one-particle density matrix:

$$\rho\left(\mathbf{r},\mathbf{r}'\right) = \sum_{\alpha}\sum_{\beta}\phi_{\alpha}\left(\mathbf{r}\right)\mathbf{K}^{\alpha\beta}\phi_{\beta}\left(\mathbf{r}'\right)$$

using $n(r) = \rho(r; r)$.

1. Extract NGWFs
2. Populate FFT boxes
3. Transform to reciprocal space
4. Up-sample to fine grid (avoid aliasing errors)
5. Transform to real space
6. Calculate product and sum
7. Deposit charge density data in simulation cell



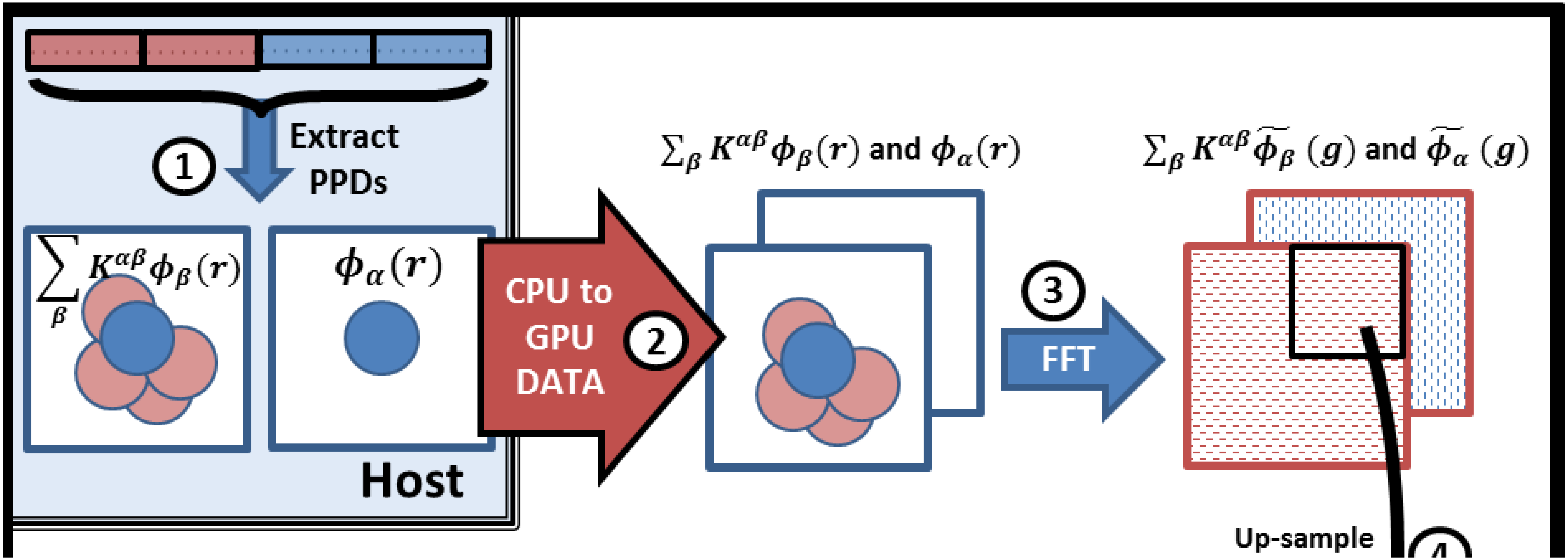
Charge Density (36% of runtime)

1. Extract NGWFs
2. Populate FFT boxes
3. Transform to reciprocal space

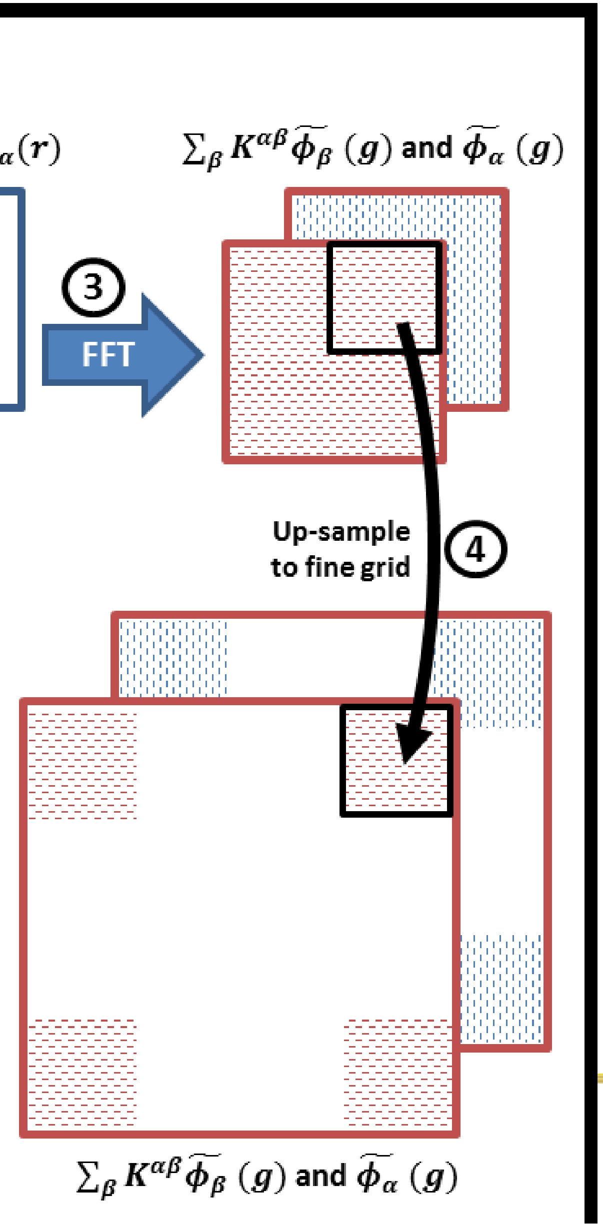
```
!Data transfer to GPU not shown

!$acc data region
!$acc region
   coarse_work(1:n1,1:n2,1:n3) = scalefac * cmplx(           &
                  fftbox_batch(:,:,:,is,3,batch_count),           &
                  fftbox_batch(:,:,:,is,4,batch_count),kind=DP)
!$acc end region

   ! Fourier transform to reciprocal space on coarse grid
   call cufftExec(cufftplan_coarse,planType,coarse_work,   &
                     coarse_work,CUFFT_FORWARD
```

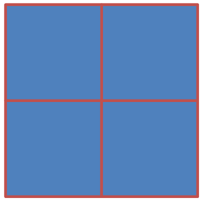


Charge Density (36% of runtime)

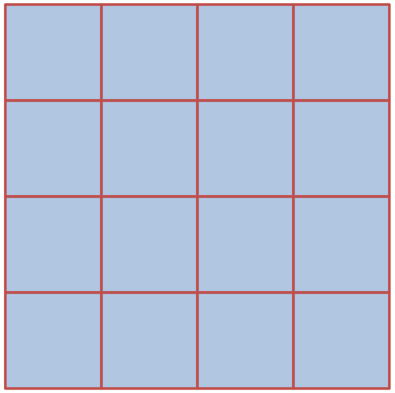


4. Up-sample to fine grid (avoid aliasing errors)

- Populate corners of a cube

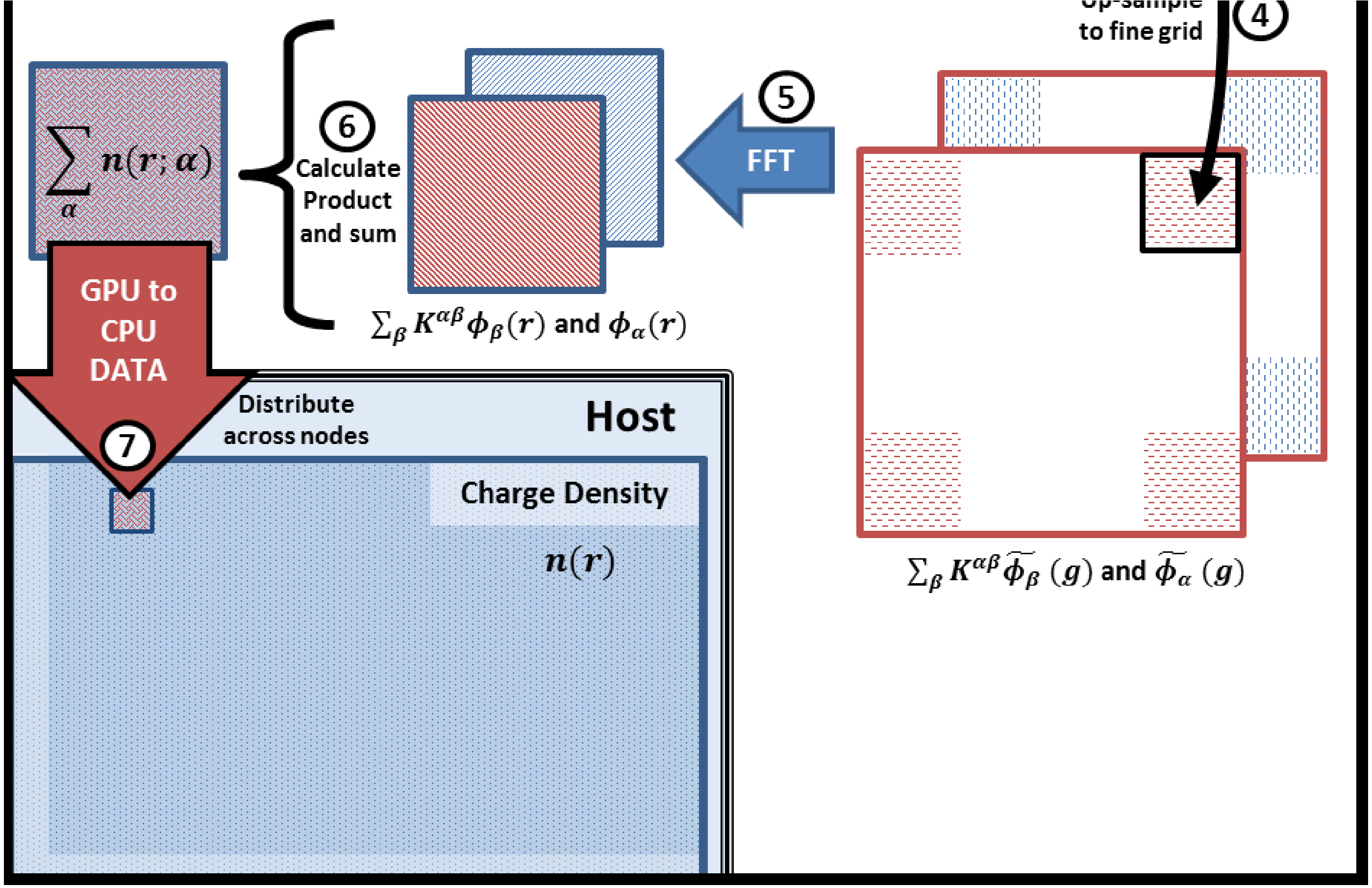


Coarse array



Fine array

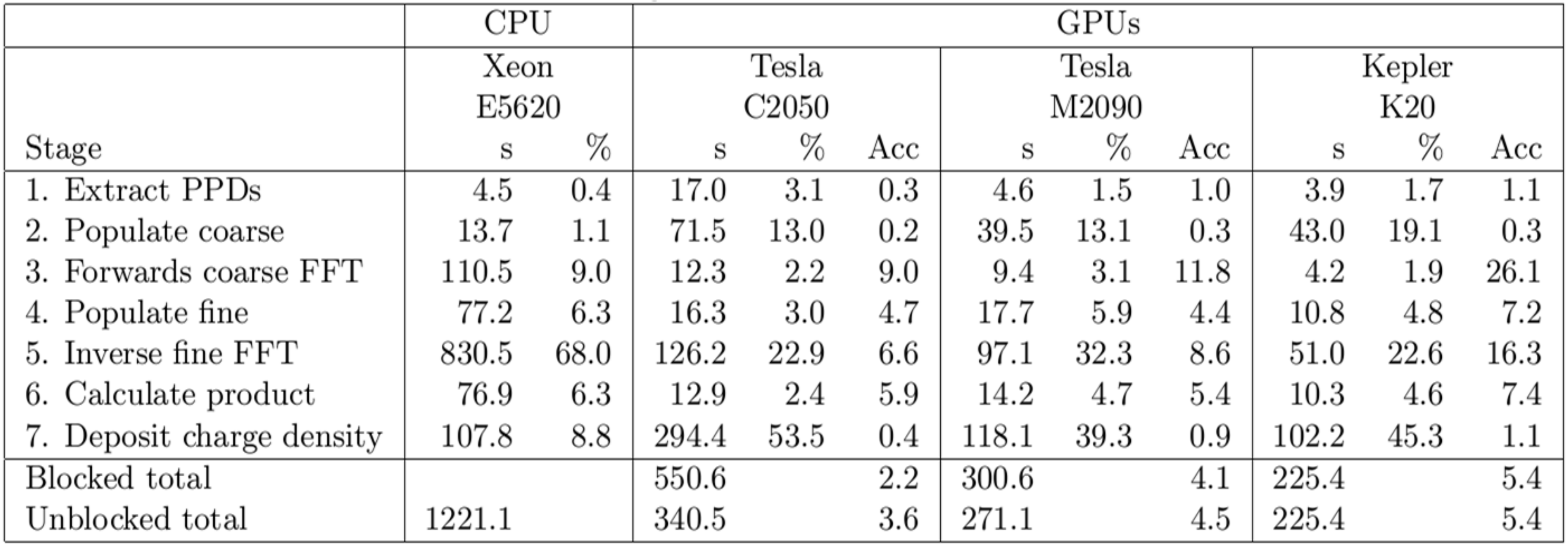
- Typical dimensions of course grid: 75-125 (odd numbers), double this for the fine grid



5. Transform to real space
6. Calculate product and sum (if common atom)
7. Deposit charge density data in simulation cell

Charge Density Performance

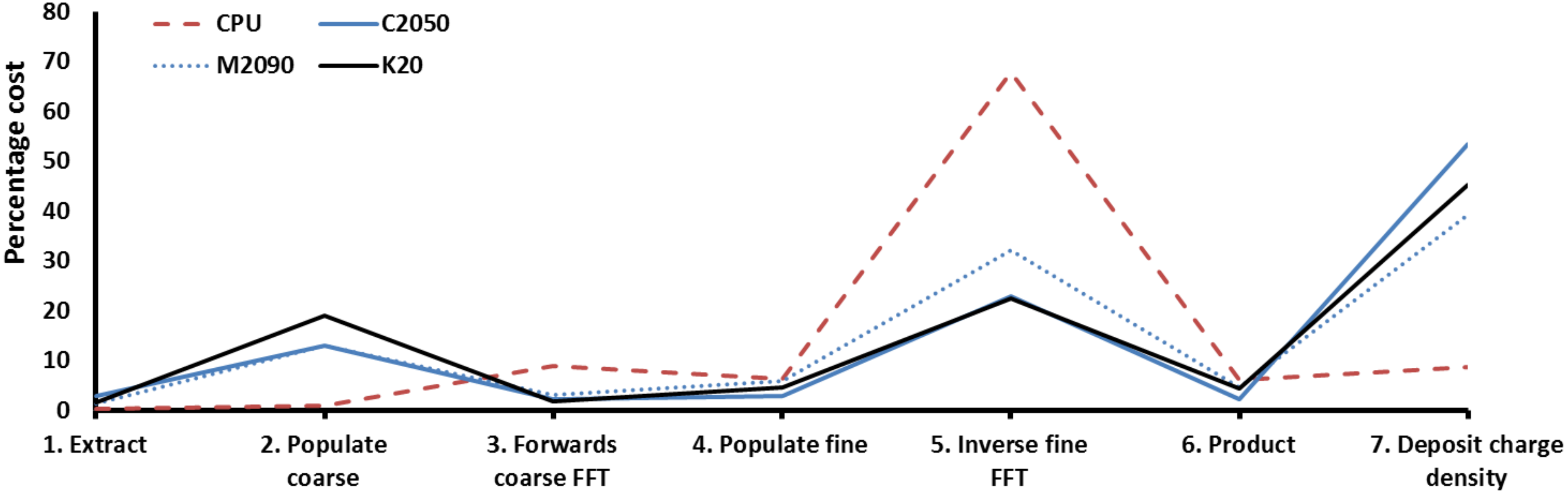
- Tested on Iridis3, Emerald and Nvidia PSG cluster
- Detailed timings are blocking
  - No asynchronous kernel execution and data transfer
- Data transfer stages (2 and 7) problematic, larger quantity of data in step 7

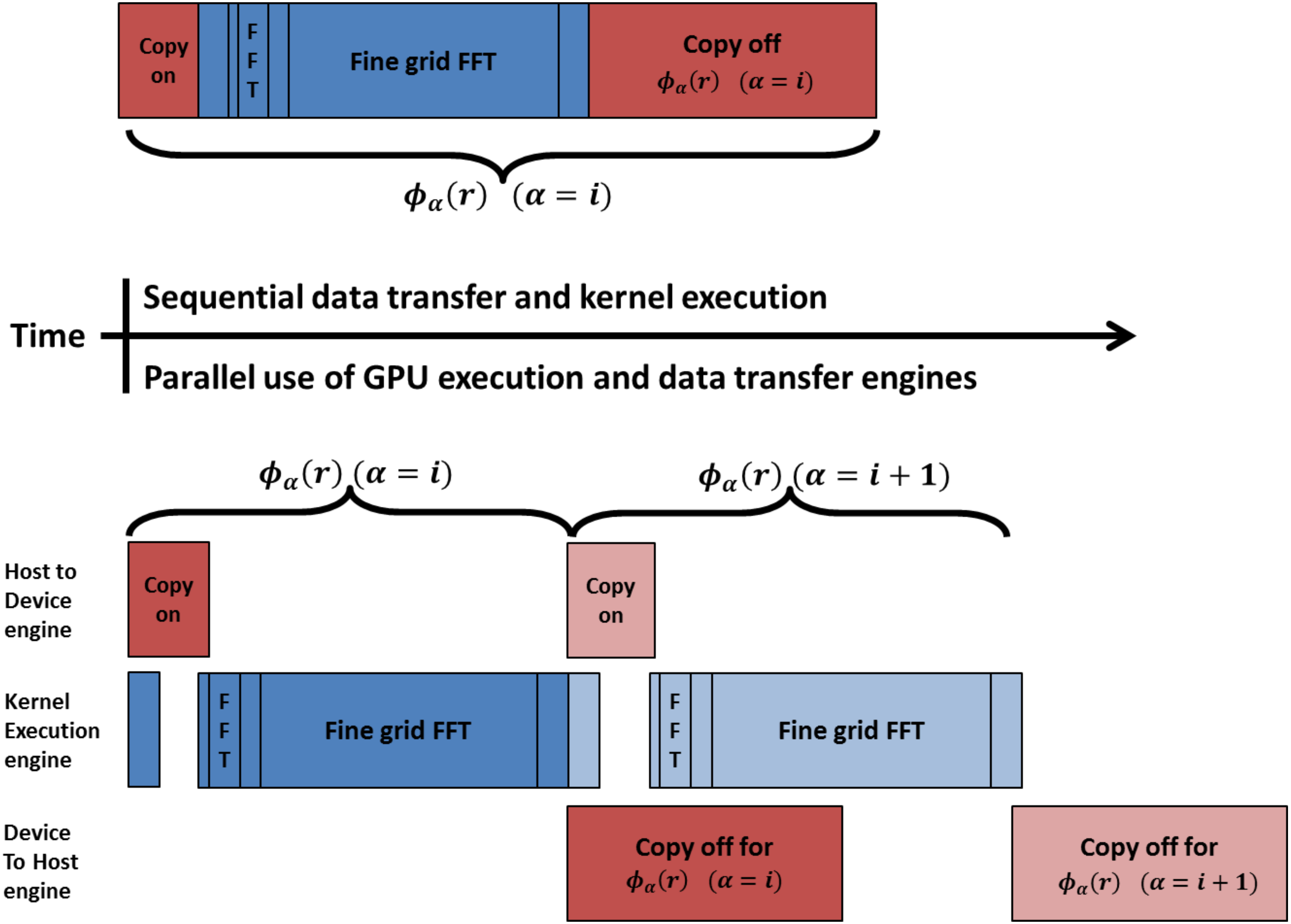


	CPU		GPUs								
	Xeon E5620		Tesla C2050			Tesla M2090			Kepler K20		
Stage	s	%	s	%	Acc	s	%	Acc	s	%	Acc
1. Extract PPDs	4.5	0.4	17.0	3.1	0.3	4.6	1.5	1.0	3.9	1.7	1.1
2. Populate coarse	13.7	1.1	71.5	13.0	0.2	39.5	13.1	0.3	43.0	19.1	0.3
3. Forwards coarse FFT	110.5	9.0	12.3	2.2	9.0	9.4	3.1	11.8	4.2	1.9	26.1
4. Populate fine	77.2	6.3	16.3	3.0	4.7	17.7	5.9	4.4	10.8	4.8	7.2
5. Inverse fine FFT	830.5	68.0	126.2	22.9	6.6	97.1	32.3	8.6	51.0	22.6	16.3
6. Calculate product	76.9	6.3	12.9	2.4	5.9	14.2	4.7	5.4	10.3	4.6	7.4
7. Deposit charge density	107.8	8.8	294.4	53.5	0.4	118.1	39.3	0.9	102.2	45.3	1.1
Blocked total			550.6		2.2	300.6		4.1	225.4		5.4
Unblocked total	1221.1		340.5		3.6	271.1		4.5	225.4		5.4

Charge Density Performance

- Clear shift in bottlenecks to data transfer stages





Copy on

F F T

Fine grid FFT

Copy off $\phi_\alpha(r) \quad (\alpha = i)$

$\phi_\alpha(r) \quad (\alpha = i)$

Time

Sequential data transfer and kernel execution

Parallel use of GPU execution and data transfer engines

$\phi_\alpha(r)\,(\alpha = i)$

$\phi_\alpha(r)\,(\alpha = i + 1)$

Host to Device engine

Copy on

Copy on

Kernel Execution engine

F F T

Fine grid FFT

F F T

Fine grid FFT

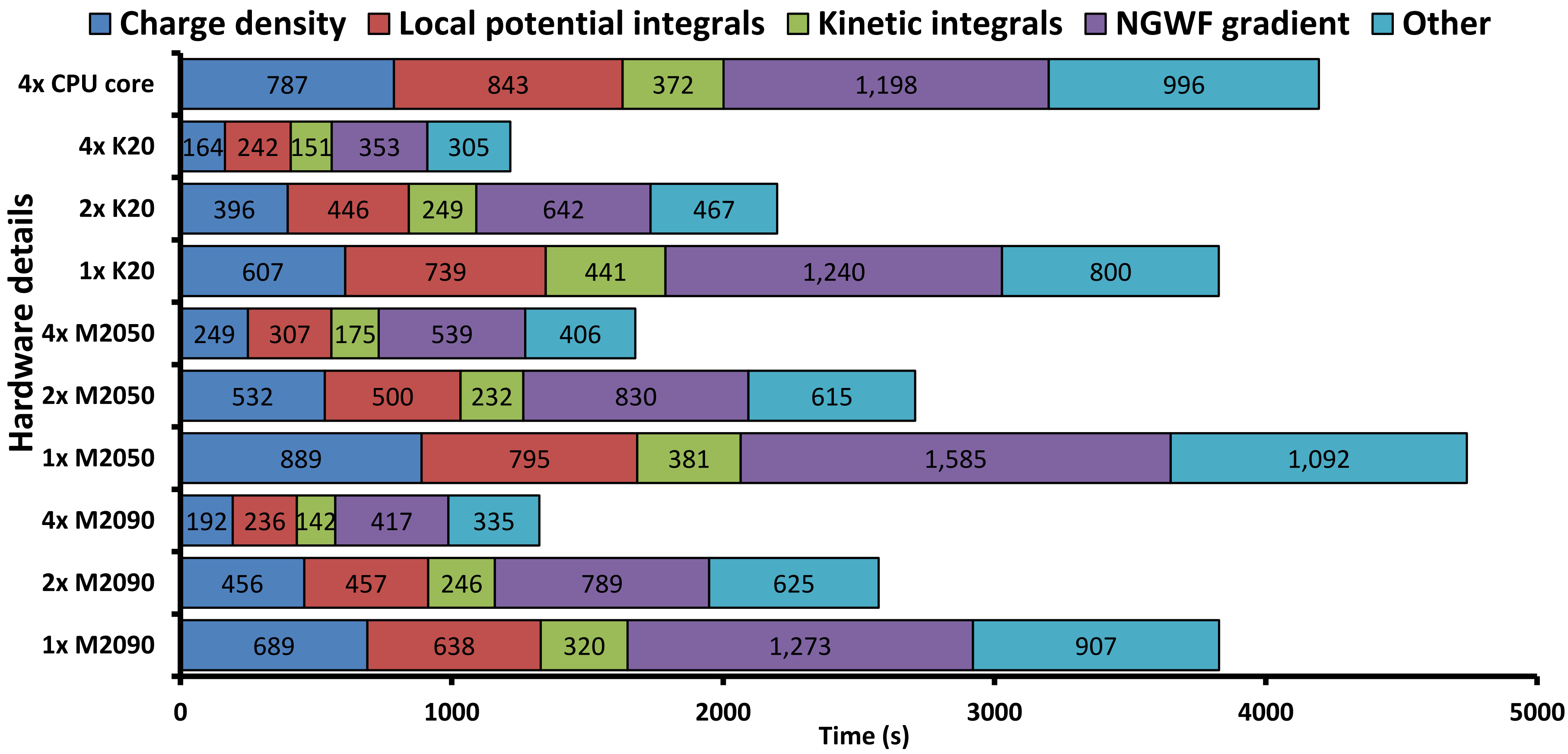
Device To Host engine

Copy off for $\phi_\alpha(r) \quad (\alpha = i)$

Copy off for $\phi_\alpha(r) \quad (\alpha = i + 1)$

SCIENTIFIC COMPUTING

Scaling Performance

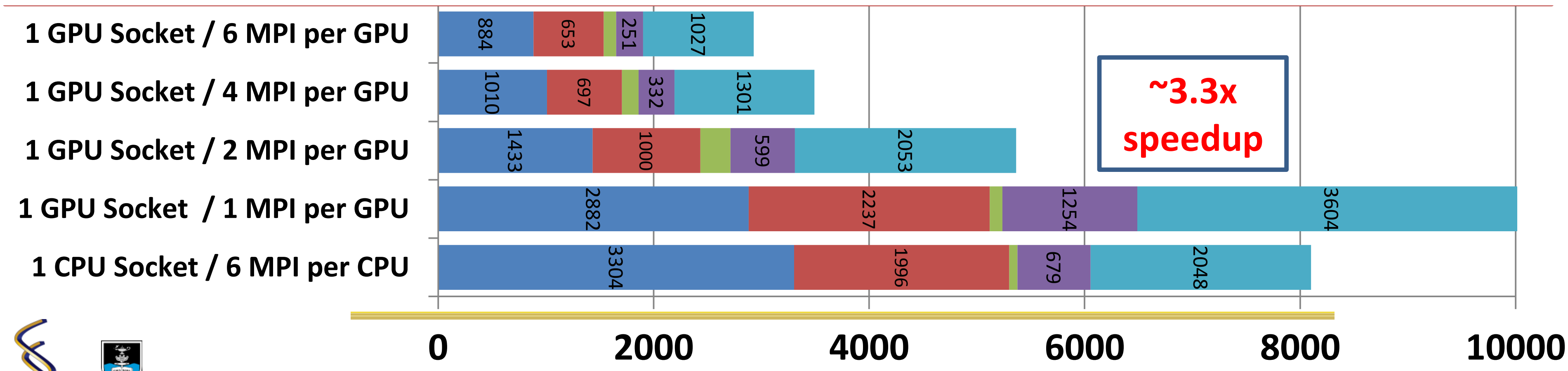


CUDA Multi-Process Service (MPS)

- Released in CUDA 5.5, July 2013. Became commonly available in clusters in 2014
- “Hypervisor”: Software that controls the way MPI processes see GPUs
- Allows multiple MPI processes to use a single GPU using the “hyperqueue” scheduler
- Dramatically increases efficiency of use

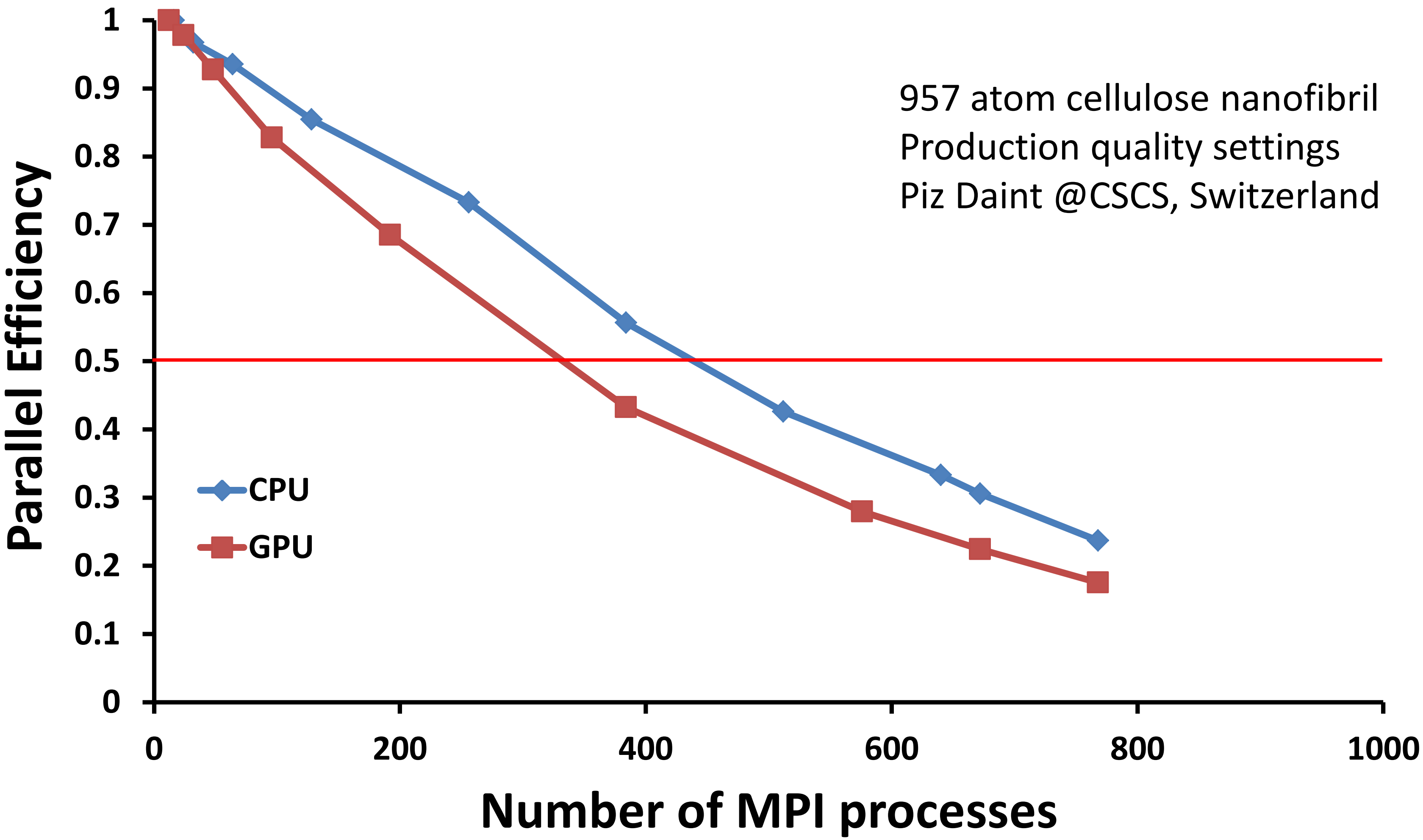
MPS Timings

181 atom tennis ball dimer, production quality settings
Wilkes cluster at University of Cambridge



~3.3x speedup

Parallel Efficiency



957 atom cellulose nanofibril
Production quality settings
Piz Daint @CSCS, Switzerland

Benchmarking ONETEP on TITAN

TITAN Details

- Oak Ridge National Laboratory

- Number 2 in TOP 500
- Peak performance of >20 Petaflops

- 18,688 physical nodes
- Each node contains:
  - 1x 16 core AMD Opteron@2.2GHz
  - 32GB RAM
  - 1x NVIDIA Tesla K20 (6GB RAM)

- Gemini high speed interconnect
- Lustre based file system: Spider

- Access through directors discretionary grant CHM112 (4M TITAN core hours)

- Simulation of dynamic properties of metal organic frameworks

- Preparation for an INCITE proposal (~60 Million TITAN core hours)

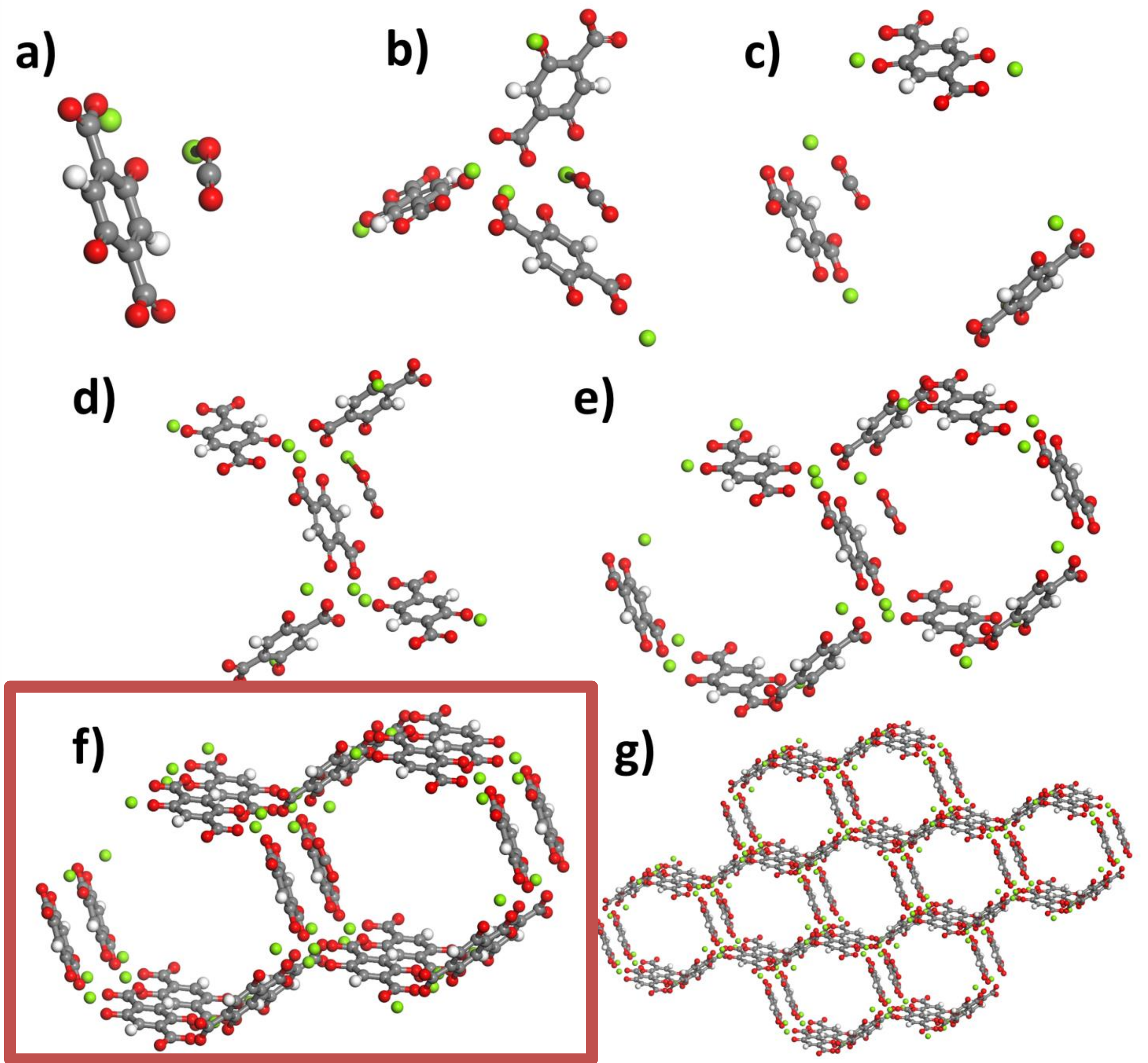
SCIENTIFIC
COMPUTING

Test Systems

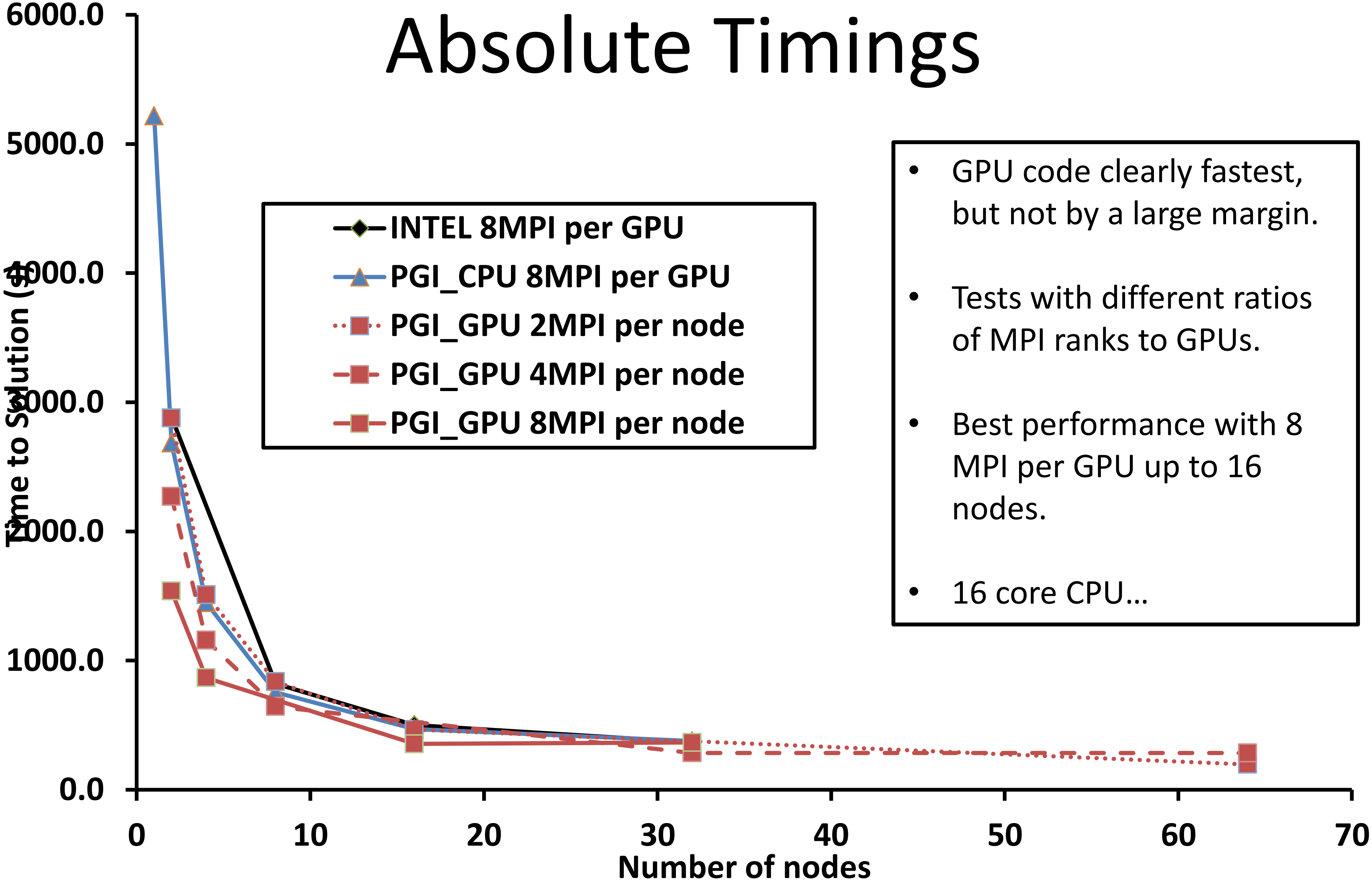
- F: 112 supercell of Mg-MOF74
- 324 atoms

- Production quality calculations:
  - 800 eV KE cutoff
  - 8.0A NGWF radii
  - Grimme D2

- Truncated in terms of time
  - 1 NGWF iteration
  - 10 LNV iterations per NGWF

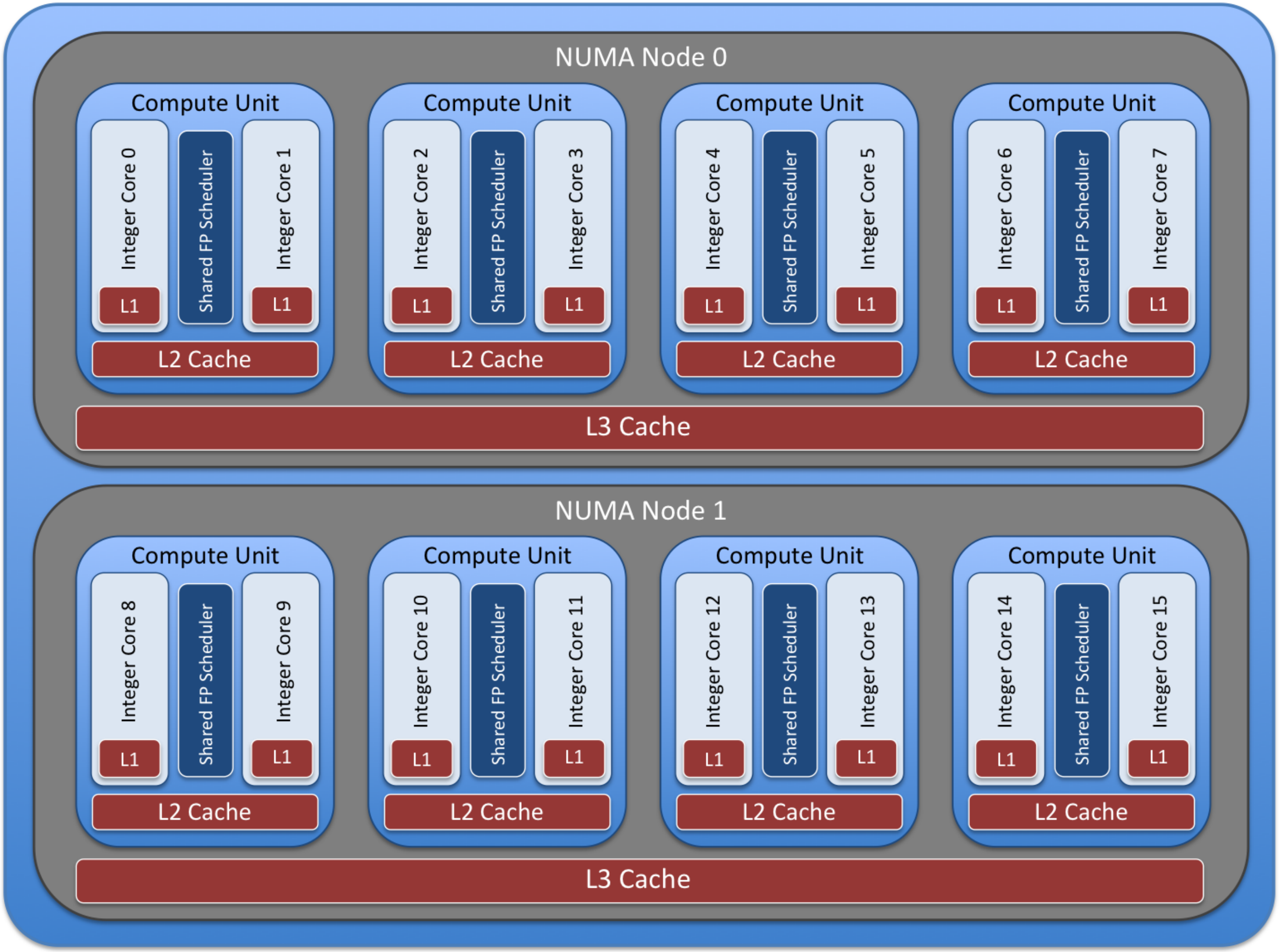


Absolute Timings



- GPU code clearly fastest, but not by a large margin.
- Tests with different ratios of MPI ranks to GPUs.
- Best performance with 8 MPI per GPU up to 16 nodes.
- 16 core CPU…

AMD Opteron™ 6274 (Interlagos) CPU



NUMA Node 0

Compute Unit
Integer Core 0
L1
Shared FP Scheduler
Integer Core 1
L1
L2 Cache

Compute Unit
Integer Core 2
L1
Shared FP Scheduler
Integer Core 3
L1
L2 Cache

Compute Unit
Integer Core 4
L1
Shared FP Scheduler
Integer Core 5
L1
L2 Cache

Compute Unit
Integer Core 6
L1
Shared FP Scheduler
Integer Core 7
L1
L2 Cache

L3 Cache

NUMA Node 1

Compute Unit
Integer Core 8
L1
Shared FP Scheduler
Integer Core 9
L1
L2 Cache

Compute Unit
Integer Core 10
L1
Shared FP Scheduler
Integer Core 11
L1
L2 Cache

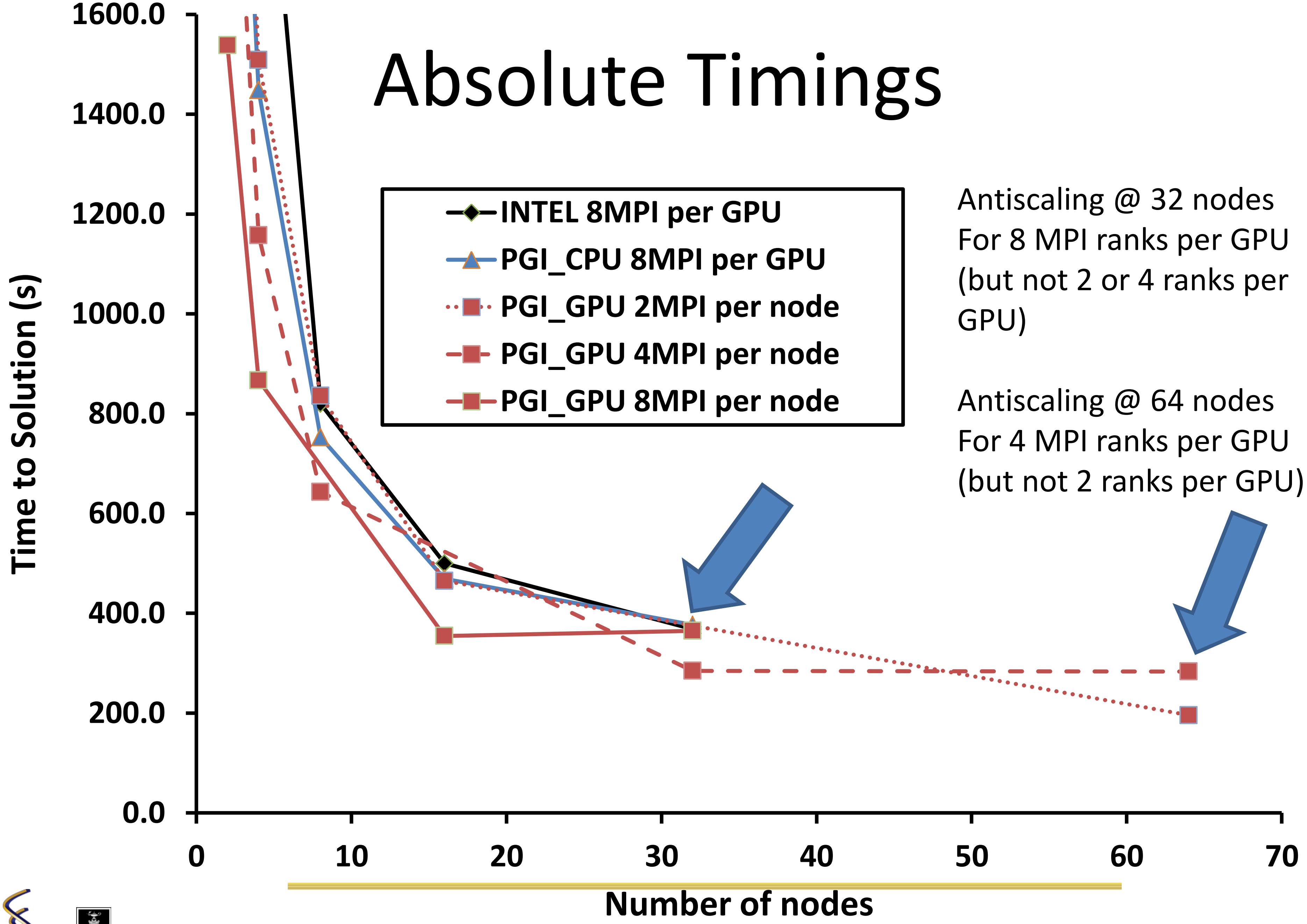
Compute Unit
Integer Core 12
L1
Shared FP Scheduler
Integer Core 13
L1
L2 Cache

Compute Unit
Integer Core 14
L1
Shared FP Scheduler
Integer Core 15
L1
L2 Cache

L3 Cache

SCIENTIFIC
COMPUTING

Absolute Timings



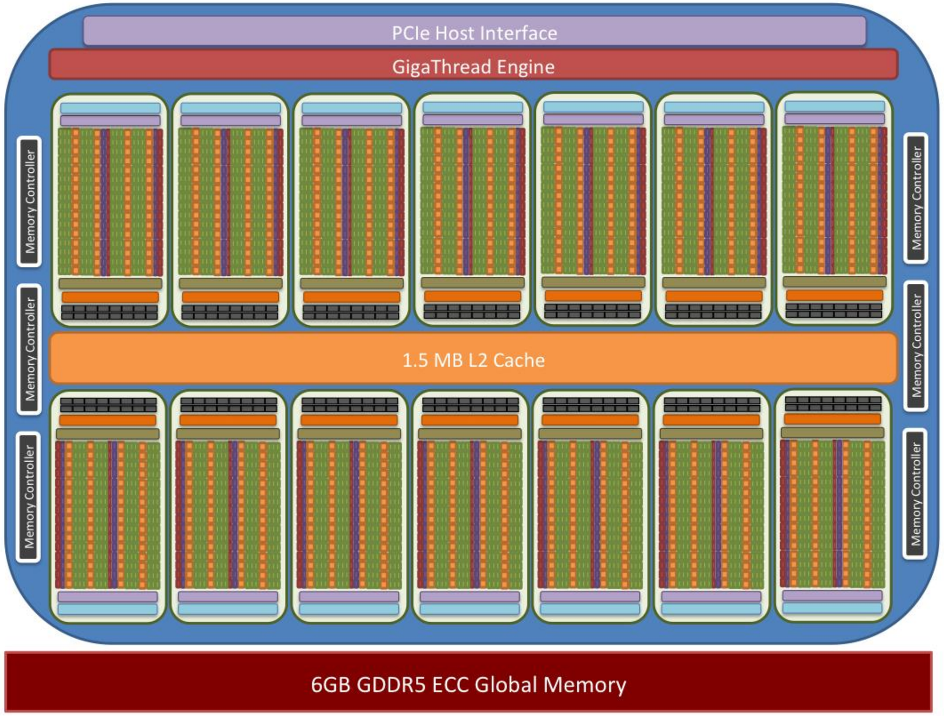
Antiscaling @ 32 nodes
For 8 MPI ranks per GPU
(but not 2 or 4 ranks per GPU)

Antiscaling @ 64 nodes
For 4 MPI ranks per GPU
(but not 2 ranks per GPU)

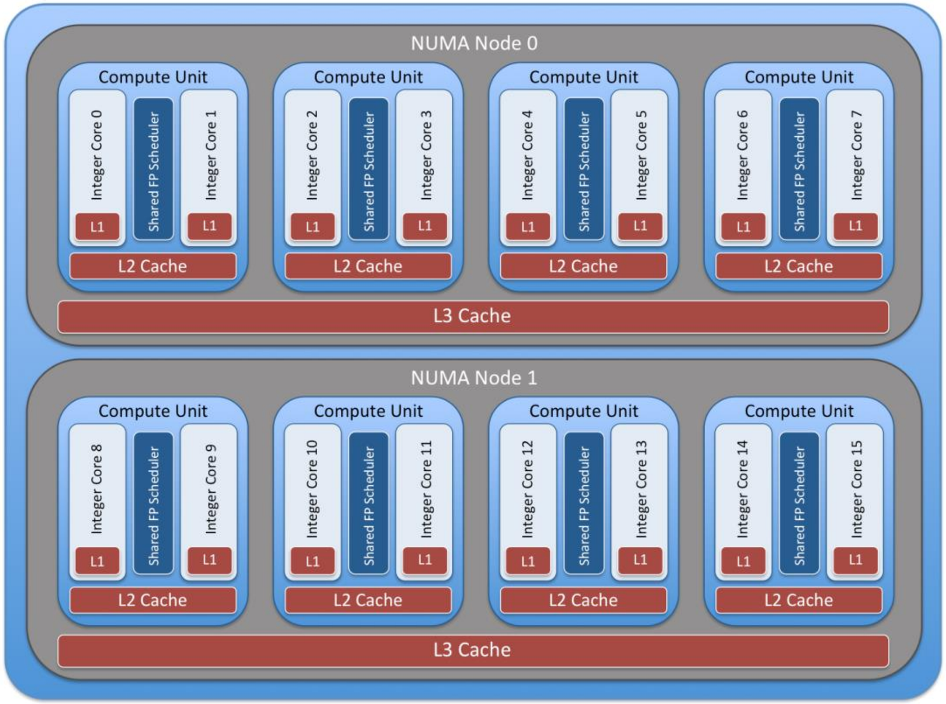
Charging on TITAN: The TITAN hour

- One node hour = 30 TITAN hours

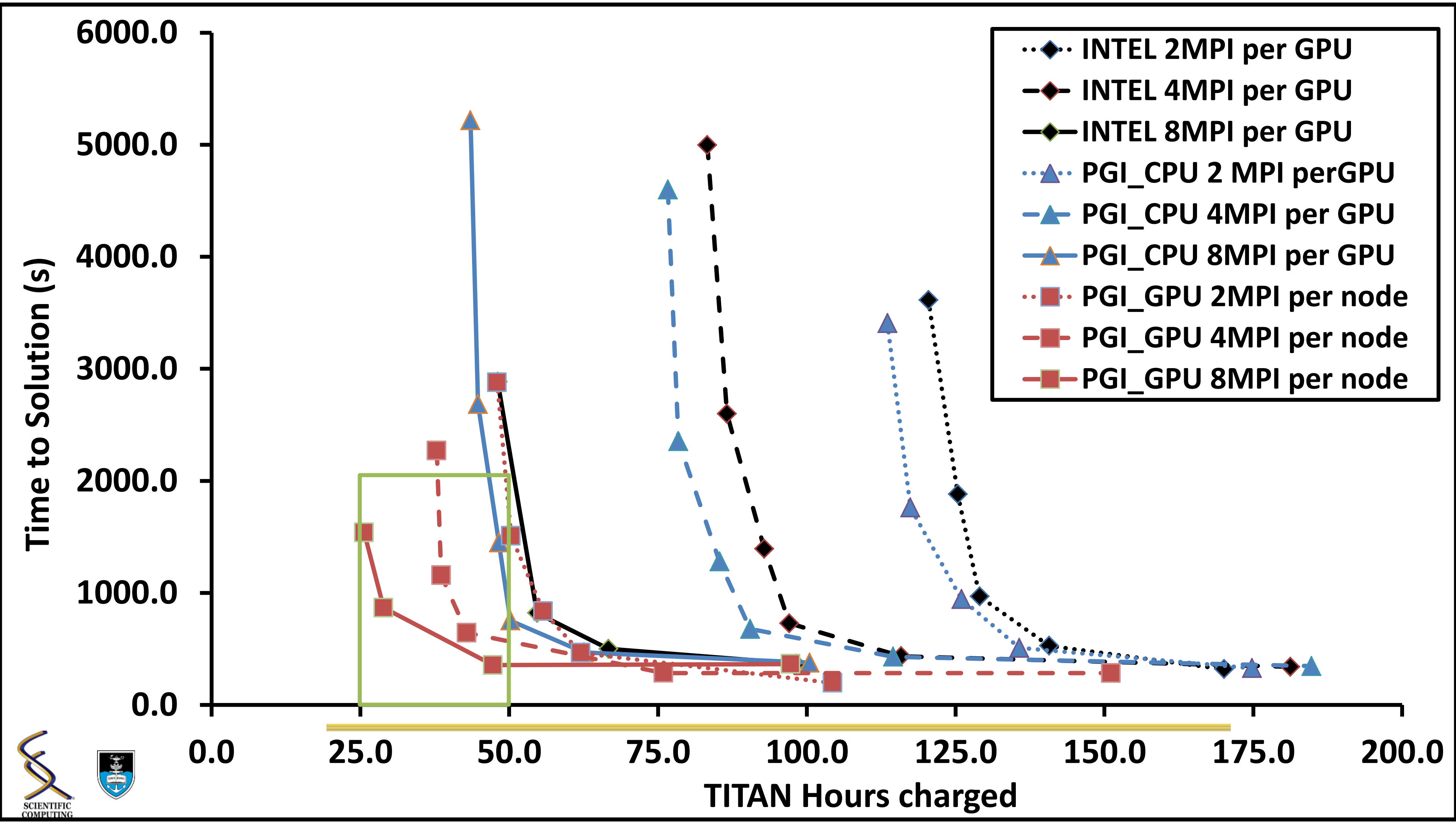
- 16 CPU compute units (SP units!)
- 14 GPU compute units (Streaming multiprocessors)



AMD Opteron™ 6274 (Interlagos) CPU

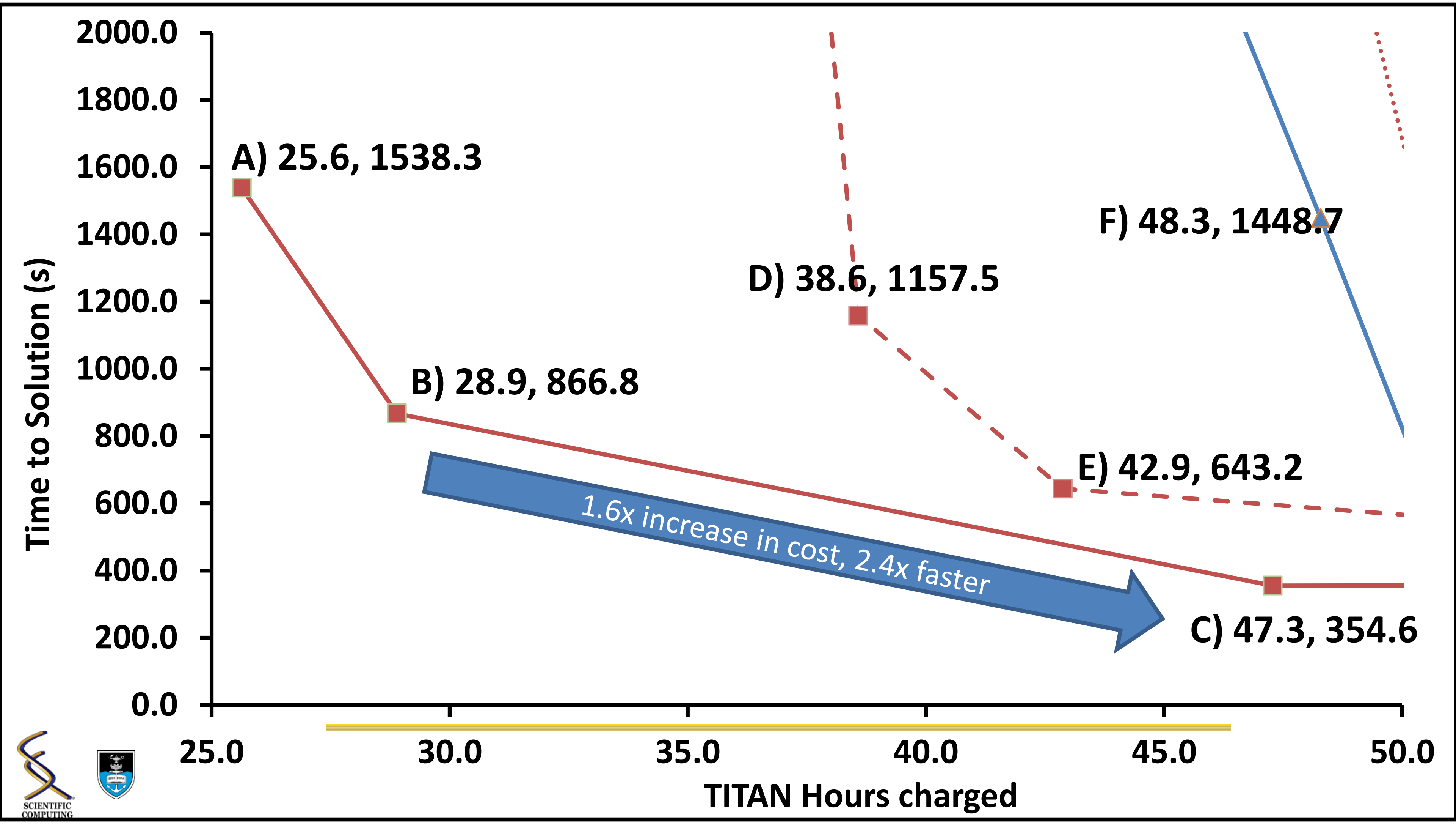


Cost vs Time

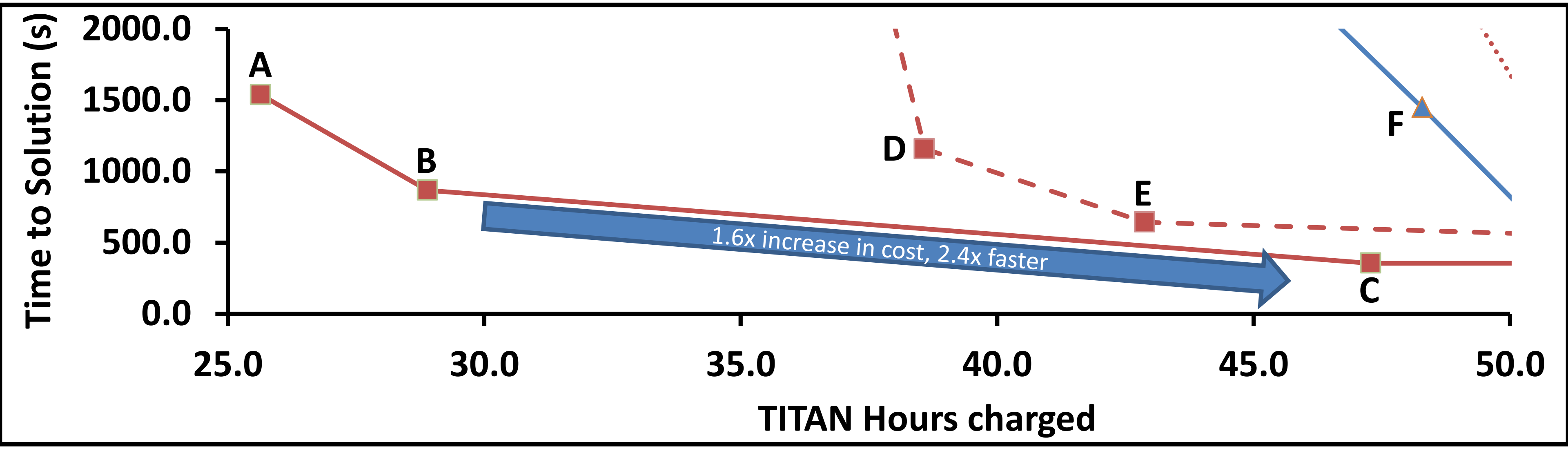


6000.0
5000.0
4000.0
3000.0
2000.0
1000.0
0.0
Time to Solution (s)
0.0
25.0
50.0
75.0
100.0
125.0
150.0
175.0
200.0
TITAN Hours charged
INTEL 2MPI per GPU
INTEL 4MPI per GPU
INTEL 8MPI per GPU
PGI_CPU 2 MPI perGPU
PGI_CPU 4MPI per GPU
PGI_CPU 8MPI per GPU
PGI_GPU 2MPI per node
PGI_GPU 4MPI per node
PGI_GPU 8MPI per node
SCIENTIFIC COMPUTING

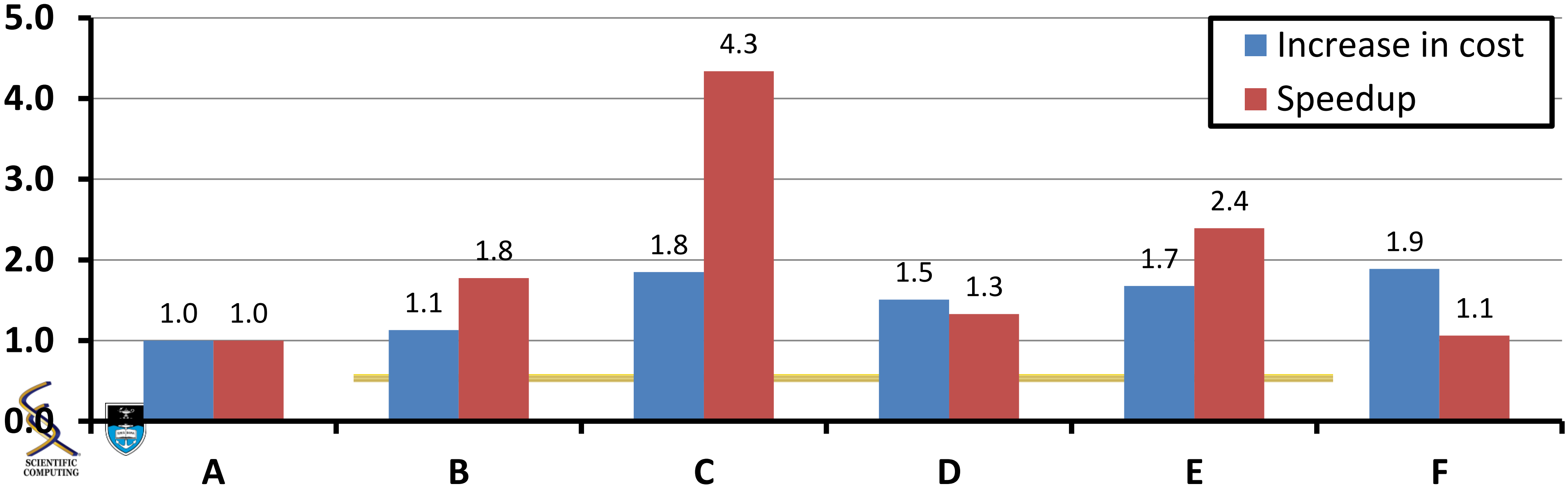
Cost vs Time


2000.0
1800.0
1600.0
1400.0
1200.0
1000.0
800.0
600.0
400.0
200.0
0.0
Time to Solution (s)
A) 25.6, 1538.3
B) 28.9, 866.8
C) 47.3, 354.6
D) 38.6, 1157.5
E) 42.9, 643.2
F) 48.3, 1448.7
1.6x increase in cost, 2.4x faster
25.0
30.0
35.0
40.0
45.0
50.0
TITAN Hours charged

Cost vs Time



1.6x increase in cost, 2.4x faster



Conclusions

- GPU implementation is faster than pure-MPI implementation, similar speed to MPI-OpenMP.
- GPU accelerated implementation is fastest on TITAN, in terms of both time to solution and cost.
- Parallel efficiency of accelerated code is lower than MPI only code:
  - Influence of MPI ranks per node is much lower for GPU accelerated code
  - Interaction between MPI communications and host-device transfers ?

Future Work

- Profiling to confirm MPI comms/data transfer issue
- Extend TITAN benchmark study:
  - Larger systems (122, 222 supercells).
  - Compare against identical calculations on other machines.
- Extension of current GPU accelerated code:
  - Sparse matrix operations
  - MPI communication from/to device
- Combination of OpenACC and OpenMP codes (increased number of “cores” per atom)