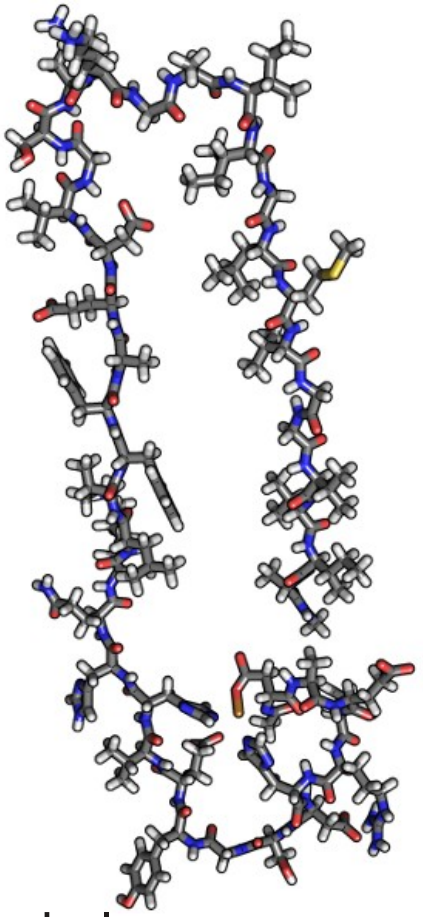
Electronic structure calculations on Thousands of CPU's and GPU's

Emil Briggs, North Carolina State University

1. Outline of real-space Multigrid (RMG)
2. Trends in high performance computing
3. Scalability limitations
4. Hybrid/threaded models
5. Scaling tests
6. Mixed precision
7. GPU acceleration



Collaborators: Wenchang Lu, Miroslav Hodak, Jerzy Bernholc
North Carolina State University

Real-space Multi-Grid (RMG)

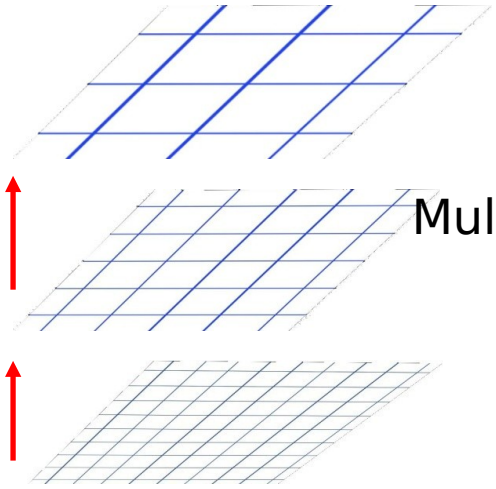
- **Density functional equations solved directly on the grid instead of with plane waves**
- **Multigrid techniques remove instabilities by working on one length scale at a time**
- **Convergence acceleration and automatic preconditioning on all length scales**
- **Non-periodic boundary conditions are as easy as periodic**
- **Compact "Mehrstellen" discretization**

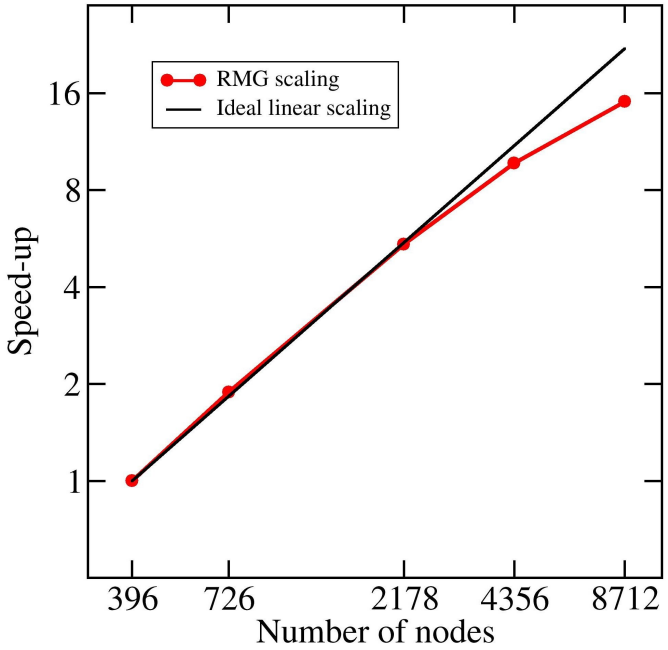
$$A[\phi_i] + B[(V_{eff} + V_{NL})\phi_i] = \varepsilon_i B[S\phi_i]$$

- **Allows for efficient massively parallel implementation (no FFTs)**

See E. L. Briggs, D. J. Sullivan and J. Bernholc Phys. Rev. B 54, 14362 (96).

Ultrasoft pseudopotentials:
M. Hodak, S. Wang, W. Lu and J. Bernholc, PRB 76, 085108 (07)





Compact Real-Space Discretization

Higher accuracy achieved by using more local information.
***Local* nature also important for MPP implementations.**
Both $\nabla^2\psi$ and V are discretized along several grid points in each coordinate direction (we use 3 points per direction) leads to a generalized eigenvalue problem

$$A[\psi_i] + B[V\psi_i] = e_i B[\psi_i] + O(h^4)$$

A* : kinetic energy operator to second order in *h
B* : smoothing operator, *I* to second order in *h
***A* and *B* are components of the *Mehrstellen* discretization.**
Two dimensional stencil form: $12h^2\,A$ $\qquad$ $12B$

-2	-8	-2
-8	40	-8
-2	-8	-2

	1	
1	8	1
	1	

We developed 3D Mehrstellen discretizations for many symmetries of crystal lattices (sc, bcc, fcc, orthorhombic and hexagonal) PRB 54, 14362 (1996) and to be published.

Alternate discretization: Central Finite difference operators
Chelikowsky et. al. Phys. Rev. Lett. 72, 1240 (1994)

Trends in HPC architecture

Multisocket/multicore SMP nodes

- 1999 – Cray T3E 1-socket and core per node
- 2013 – Cray XK6 2-sockets and 32 cores per node
- 2020 - ?

High speed interconnect between nodes

- Infiniband
- Myrinet
- Cray Gemini

GPU/Accelerator

- Nvidia Fermi/Tesla/Kepler
- AMD Radeon HD
- Intel Xeon PHI

Multicore CPU's

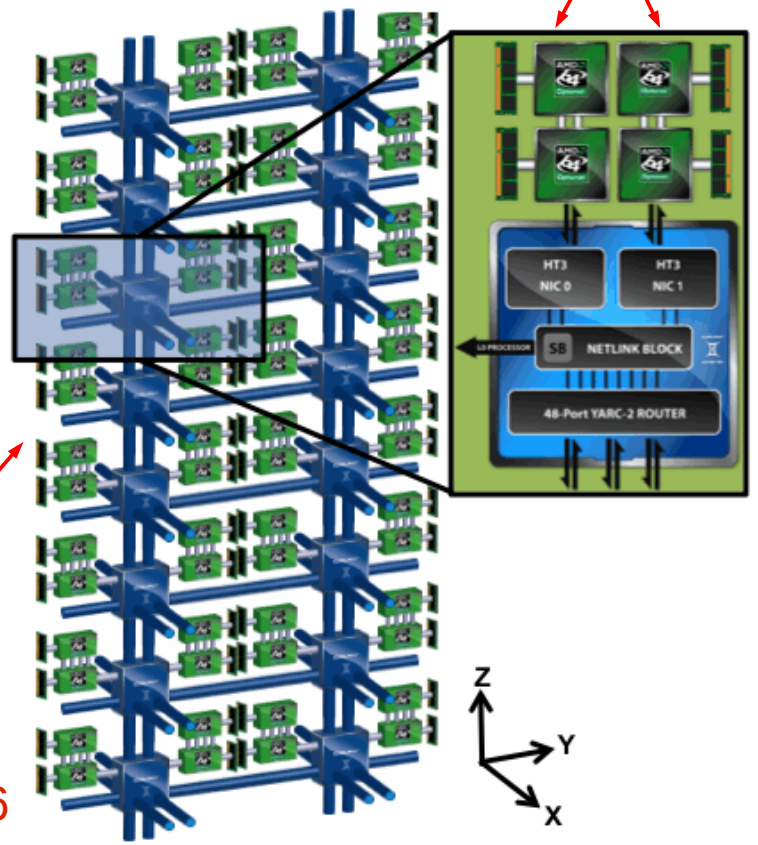


Image courtesy of Cray, Inc.

Schematic of Cray XE6

Challenges for Electronic structure codes

Single core performance growth slowing down

Typical single core performance 2006 vs 2013

2006 -	AMD Opteron	15 SPECfp	
2013 -	Intel Xeon	80 SPECfp	**5.3 times faster**

Typical single socket performance 2006 vs 2013

2006 -	AMD Opteron (*4 cores*)	18 SPECfp_rate	
2013 -	Intel Xeon (*8 cores/ht*)	325 SPECfp_rate	**18.1 times faster**

Core/socket performance difference due to more cores per socket

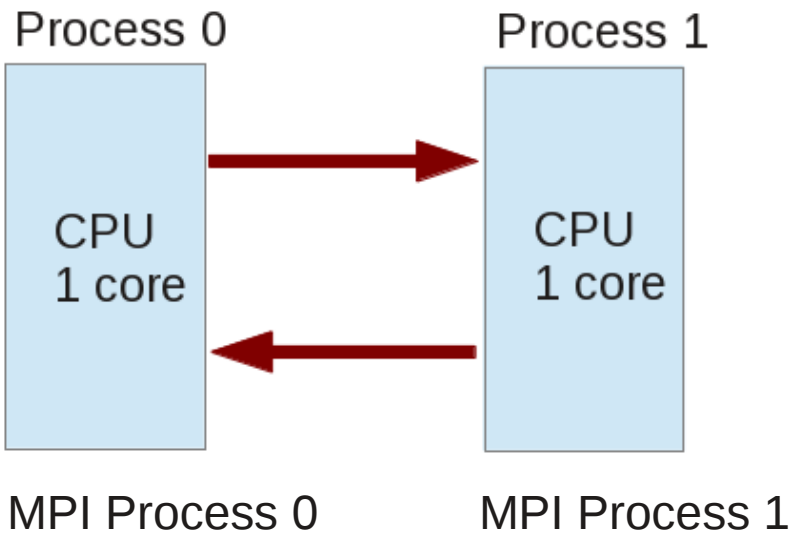
Future performance gains will increasingly come from adding cores and/or accelerators rather than improved single core performance

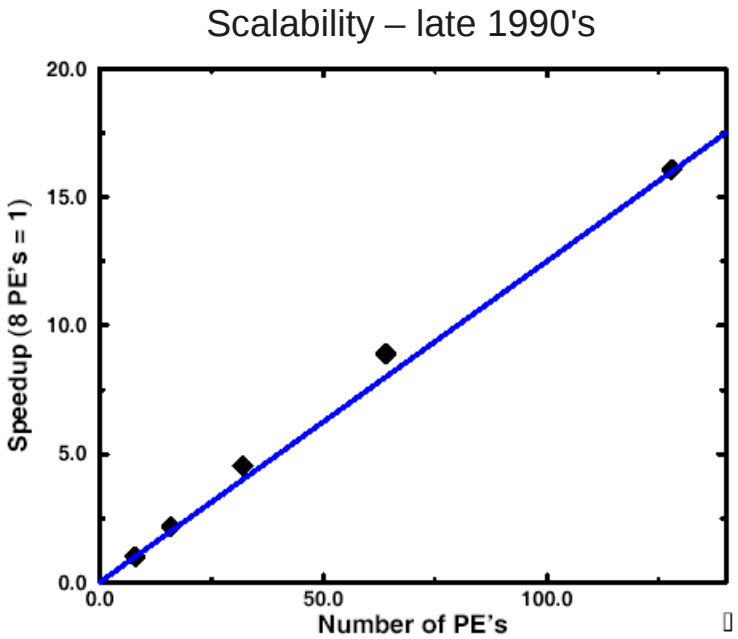
RMG parallelization 1990's

Based on MPI: Message passing interface developed during the early 90s

- Standardized and portable
- Designed for distributed memory operations
- Supports both point to point and collective communications
- Each MPI process ran on a single single network node

MPI_Send(buffer, count, type,
destination, tag, comm)





Scaling of RMG in 1990's

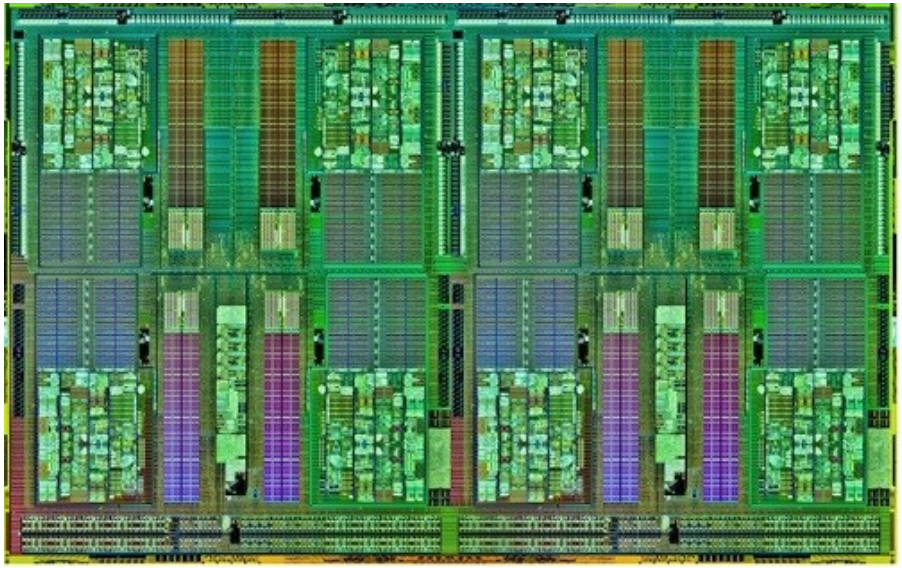
Limitations on scalability

Common programming model circa 2000 was still 1 MPI process/core
And 1 core per network node

Situation today. Core counts getting very large!

Current machines	**400,000** cores
Next generation	**1,000,000** cores

AMD bulldozer chip – 8 cores
Opteron 6xx - 2 chips/multi-chip module
Cray XE6 – 2 Opterons/network node
32 cores per network node!



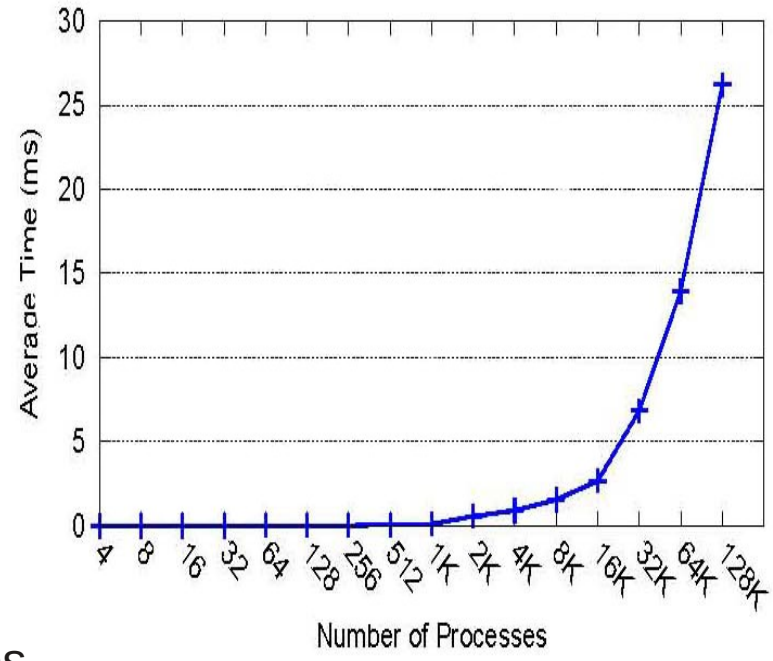
Problems with both MPI specifications and implementations
Vendors and standard bodies working on addressing MPI issues
Applications need to consider the issues as well.

Examples

Scalability issues in the MPI specification

MPI Alltoallv – each process sends data to every other process and also receives data from every other process.
IBM Blue Gene/P results
Actual transfer sizes are 0 bytes!

Poor scaling due to MPI specification!



R. Thakur et, al Proceedings of the 16th European PVM/MPI Users Group Meeting, Springer-Verlag Berlin, 2009

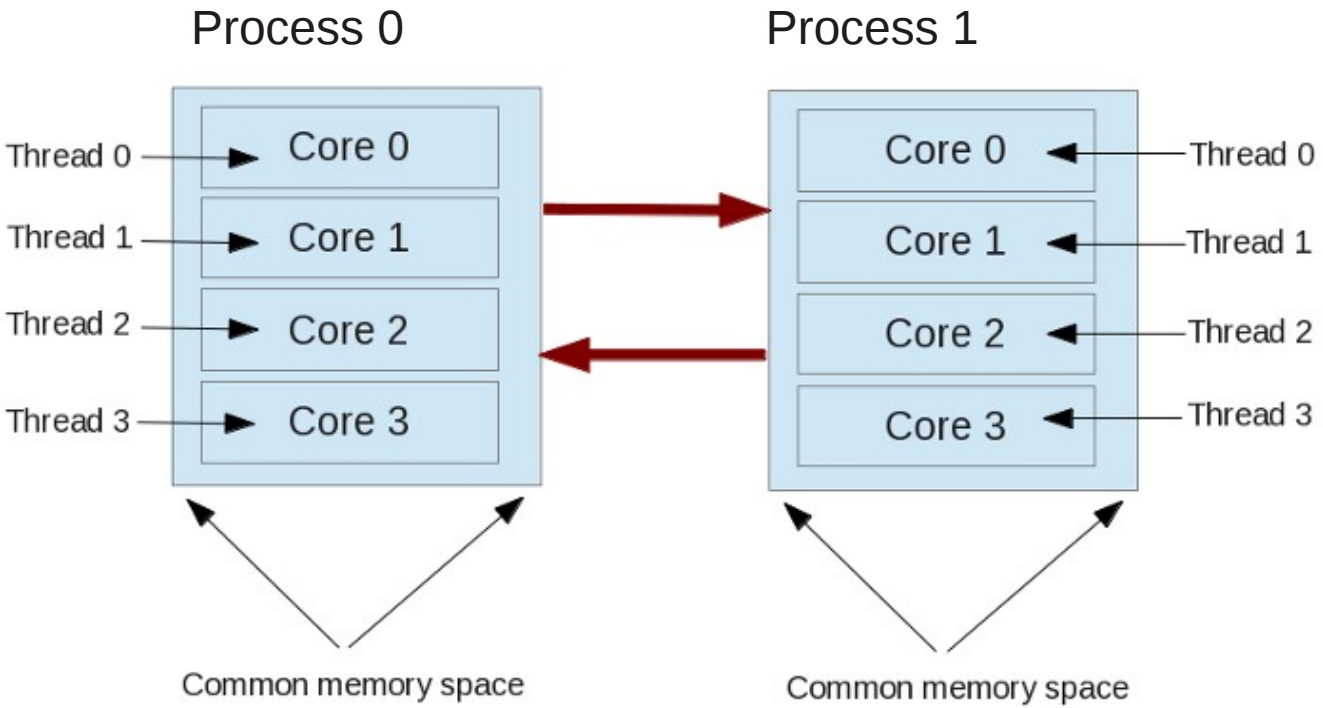
Alltoallv functions take array arguments with dimension equal to the number of MPI processes
Same problem in other MPI functions.

Implementation example: Global reduction operations (e.g. sum data over all processes and distribute the summed result back to all processes).
Tree algorithm implementation scales as Nlog(N) so execution time increases with N.

Hybrid MPI/threads/OpenMP to improve scalability

Use 1 MPI process per node rather than 1 process per core

- Inter node parallelization uses traditional MPI
- Intra node parallelization uses shared memory threads
- Extra cores used via Libraries, Pthreads, Openmp



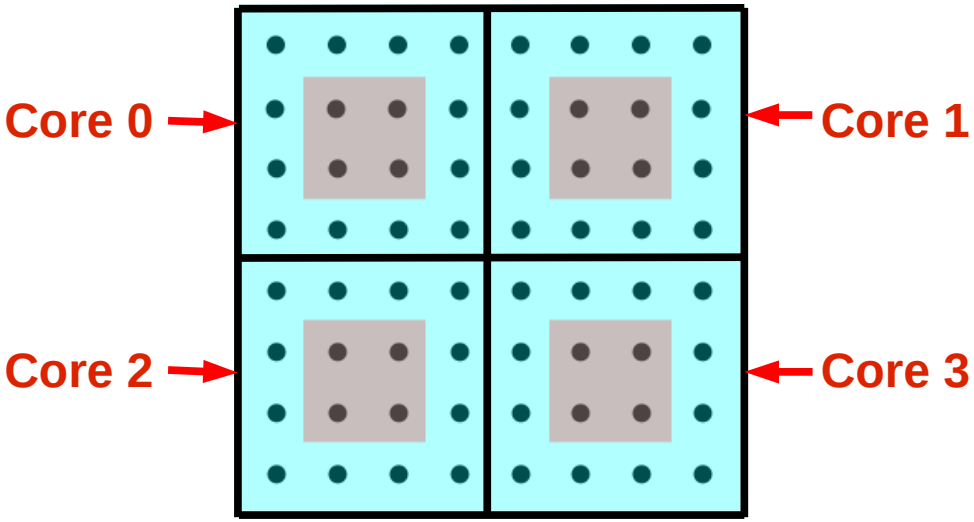
Hybrid model benefits/drawbacks

Benefits

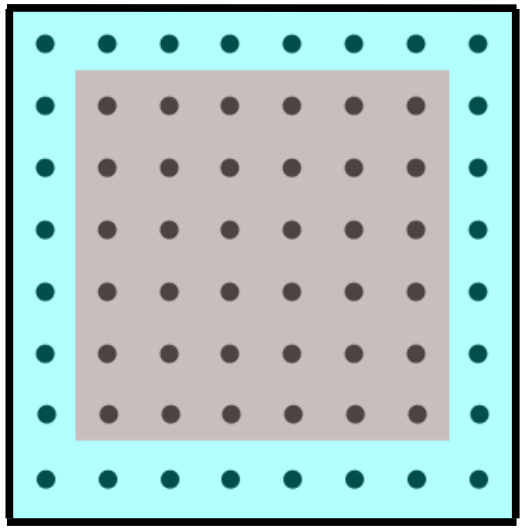
Order of magnitude reduction in MPI process count
Corresponding increase in real space domain size per node
Surface to volume ratio reduced for finite differencing

Cray XK7 = 16 cores/node Cray XE6 = 32 cores/node

Standard model 1 MPI process/core



Hybrid model 1 MPI process/node



Node 0

Drawbacks

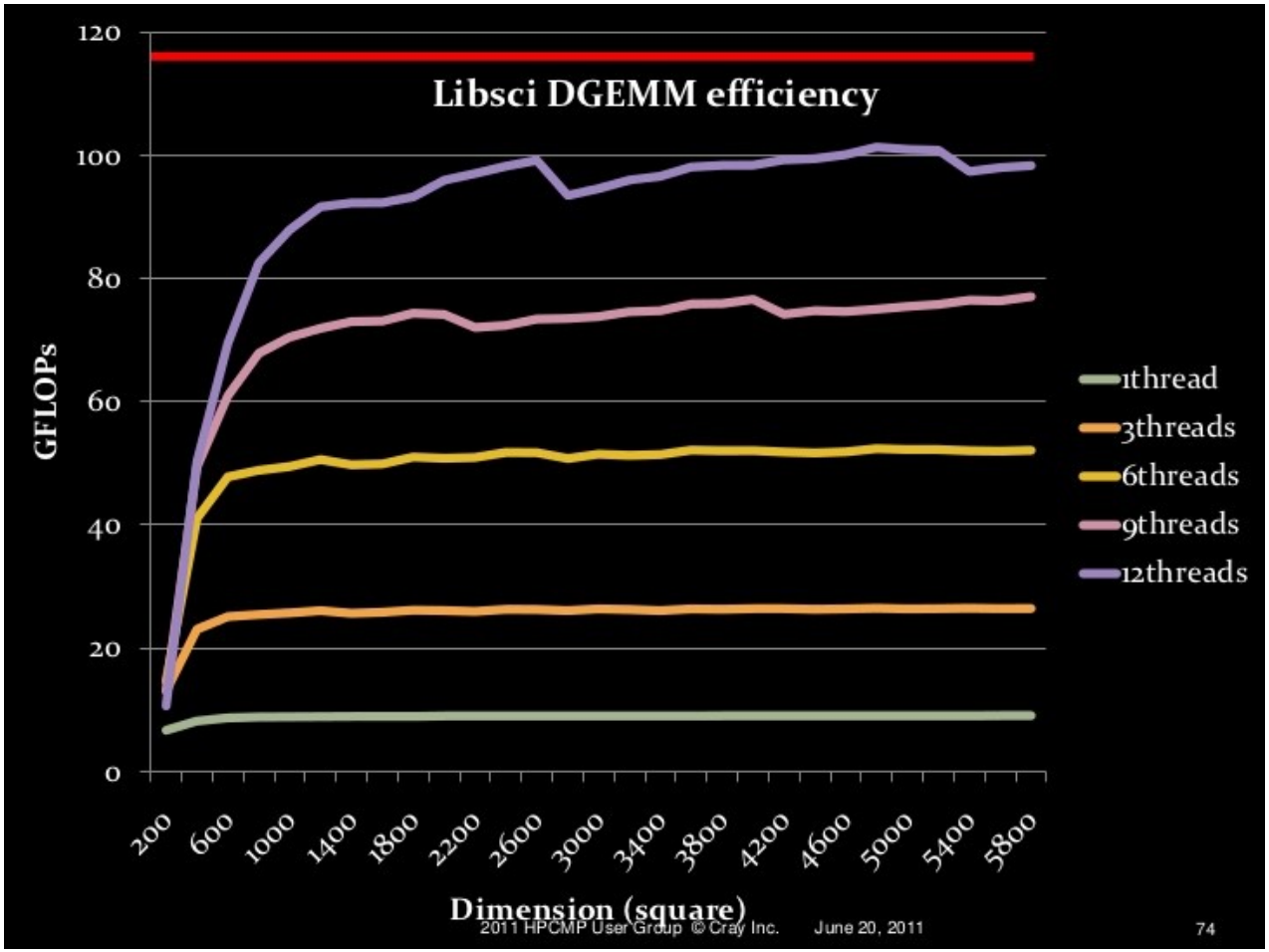
Additional programming complexity required to utilize all cores on a node
Greater overhead (potentially worse performance for small problems)

Vendor libraries

Vendor supplied BLAS, LAPACK, etc. Easy to use (link and runtime control)
Large performance gains possible with little work

DGEMM BLAS3 function performs matrix multiply
C = A x B

Tests performed on Cray XT5 with 12 core CPU's



Open MP

Shared memory multiprocessing Implemented on top of a compute platforms native threading model
Excellent implementations available for a wide range of platforms
Source code modifications via pragmas and directives
Pragma provides additional information to compiler outside of the standard Language definition

```
#pragma parallel for private(istate,sp,t1)
    for (idx = 0; idx < nbasis; idx++)
    {
       A[idx] = B[idx] + C[idx];
    }  /*end for idx */
```

Particularly easy to use for loop level parallelism
Not as well suited for other types of parallelism

Pthreads

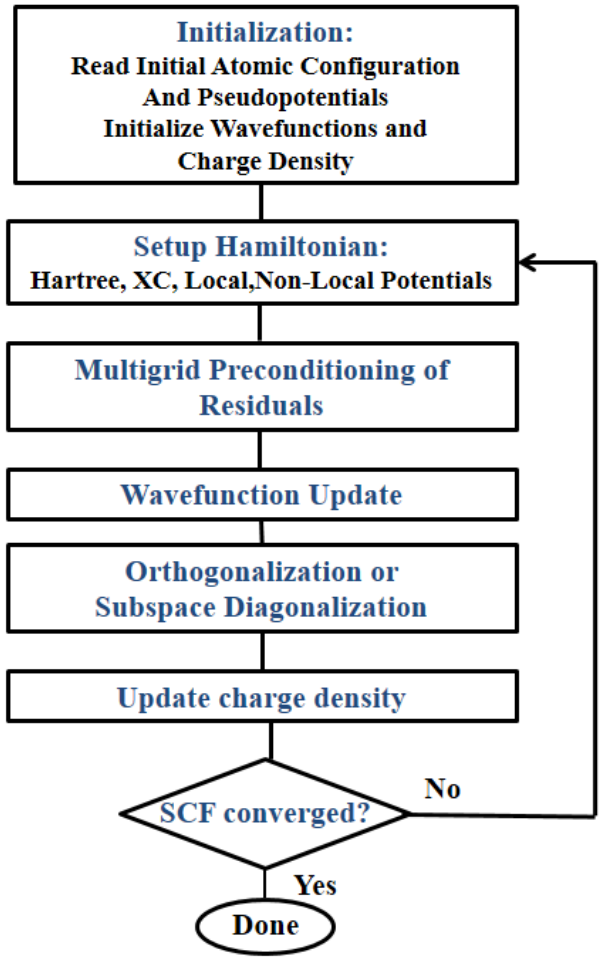
Available for Unix/linux platforms. Provides compete flexibility and control. Careful analysis of algorithms required. A higher level of programming expertise is necessary. More extensive source code modifications than OpenMP.

```
void thread_function(void *arg)
{
 ........
}

// Explicitly create thread which executes thread_function
ret = pthread_create(&thread, &thread_attrs,
                     (void *)thread_function, (void *)arg);
```

Best suited for task based parallelization schemes where complex synchronization and locking between tasks may be required. (e.g. each separate tab or window in a web browser could use a different thread.

RMG Implementation



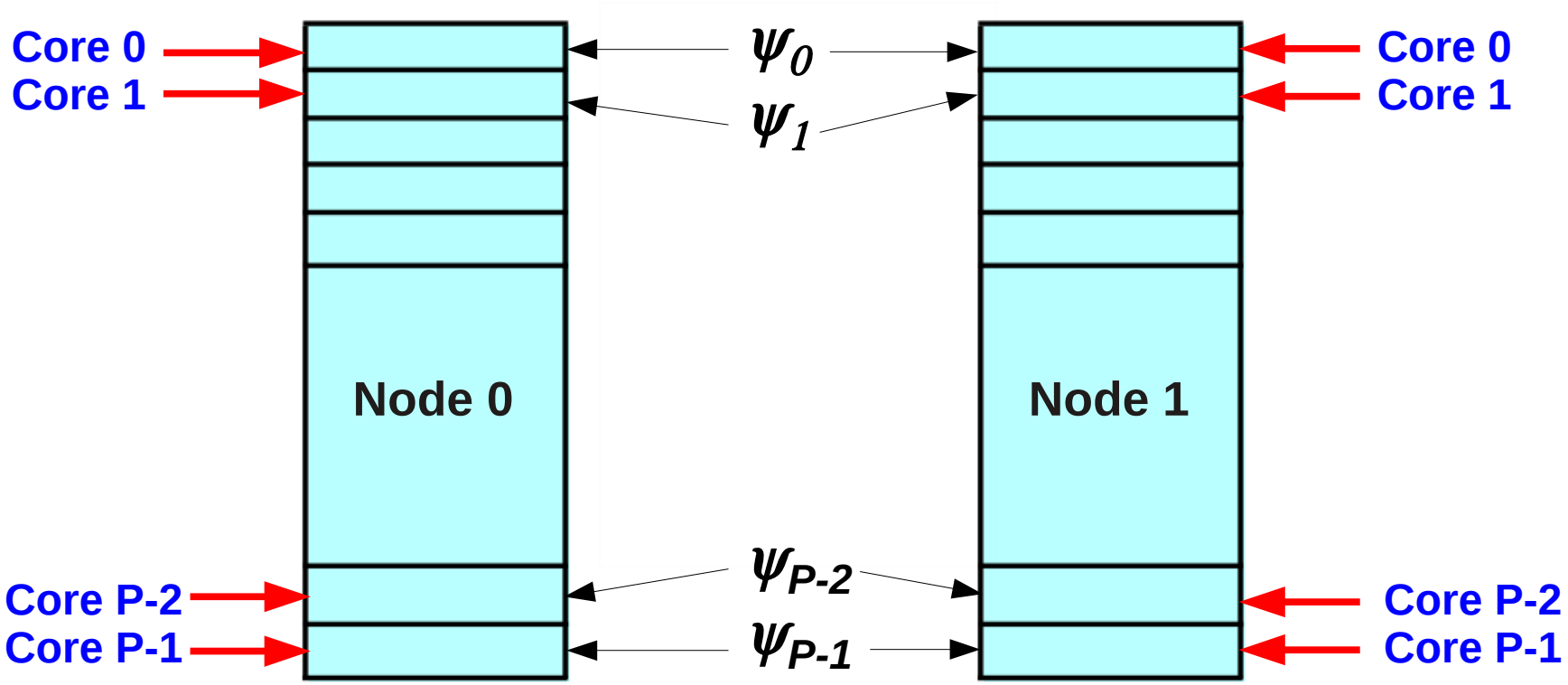
RMG uses MPI for inter-node parallelization. Vendor libraries, OpenMP and Pthreads are used for intra-node. Workload divides naturally into two separate types.

Multigrid preconditioning:
Electronic orbitals may be processed independently.
Nearest neighbor inter-node communications dominate.
Large numbers of small messages. Latency important.
Little opportunity to use parallel library routines.

Orthogonalization and/or subspace diagonalization:
Electronic orbitals must be processed together.
Large global inter-node communications dominate.
Easy to use library routines
(BLAS3, LAPACK, SCALAPACK).

Pthreads for multigrid preconditioner

Consider a typical electronic structure problem with N orbitals
The computer system used for solution has P processing cores per node
Orbitals may be processed independently and N>>P
Natural parallelization method is to assign each orbital to a single core
OpenMP or Pthreads should work equally well

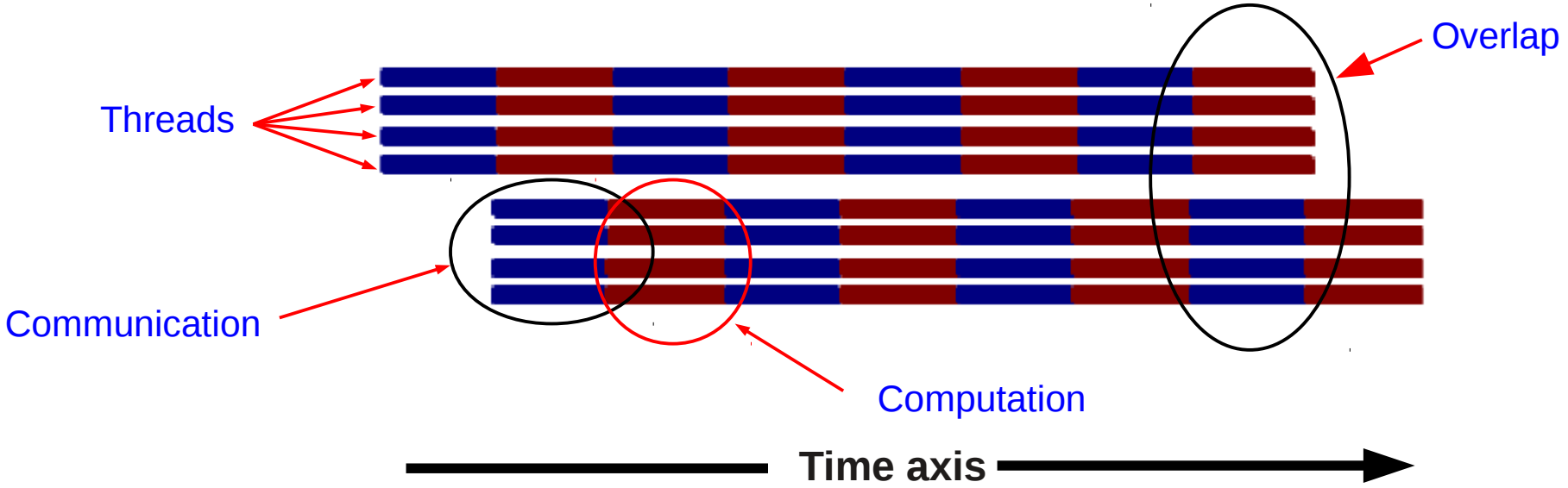


While each thread can operate independently is this the best approach?

Hiding latency

Finite difference communications are nearest neighbor
Message sizes are small and reducing/hiding latency is critical to performance

Combine small messages from multiple threads
Overlap communications **(blue)** and computations **(red)** using blocks of threads



Conceptually simple, implementation is non-trivial
Careful synchronization and locking between threads/tasks required
Greater flexibility and control of pthreads made it the better choice

Orthogonalization/Diagonalization

Computational tasks

1. Generate matrix element contributions from each node
2. Sum contributions from all nodes
3. Diagonalize resultant matrix (N^3 scaling) dominant for large problems

Hybrid model improvements for Step 1 using threaded DGEMM
Step 2 uses MPI_Allreduce - improvements mainly dependent on vendors
Various library packages available for Step 3 (LAPACK, SCALAPACK, MAGMA, others) performance improvements from threads/GPU

Hybrid model and memory constraints

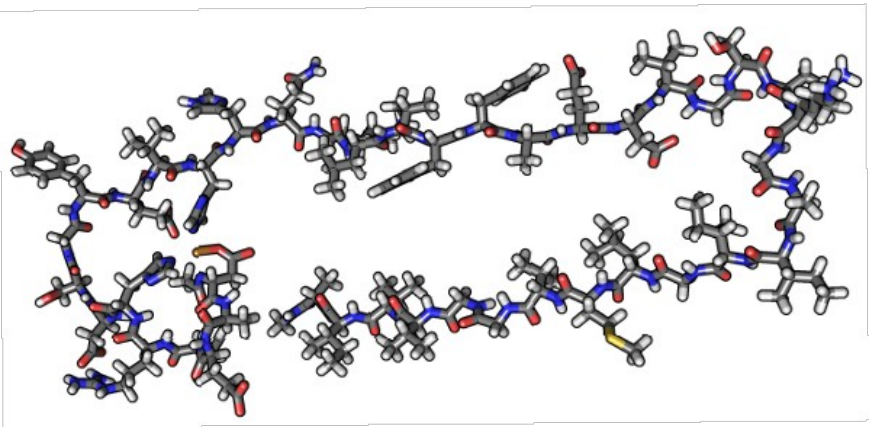
Consider Cray XE6 with 32 cores and 32 Gbytes/RAM node
Using 1 MPI process per core means only 1 Gbyte RAM/process
Not enough for large problems! Memory overhead and data structures needed by each MPI process leaves little available free memory

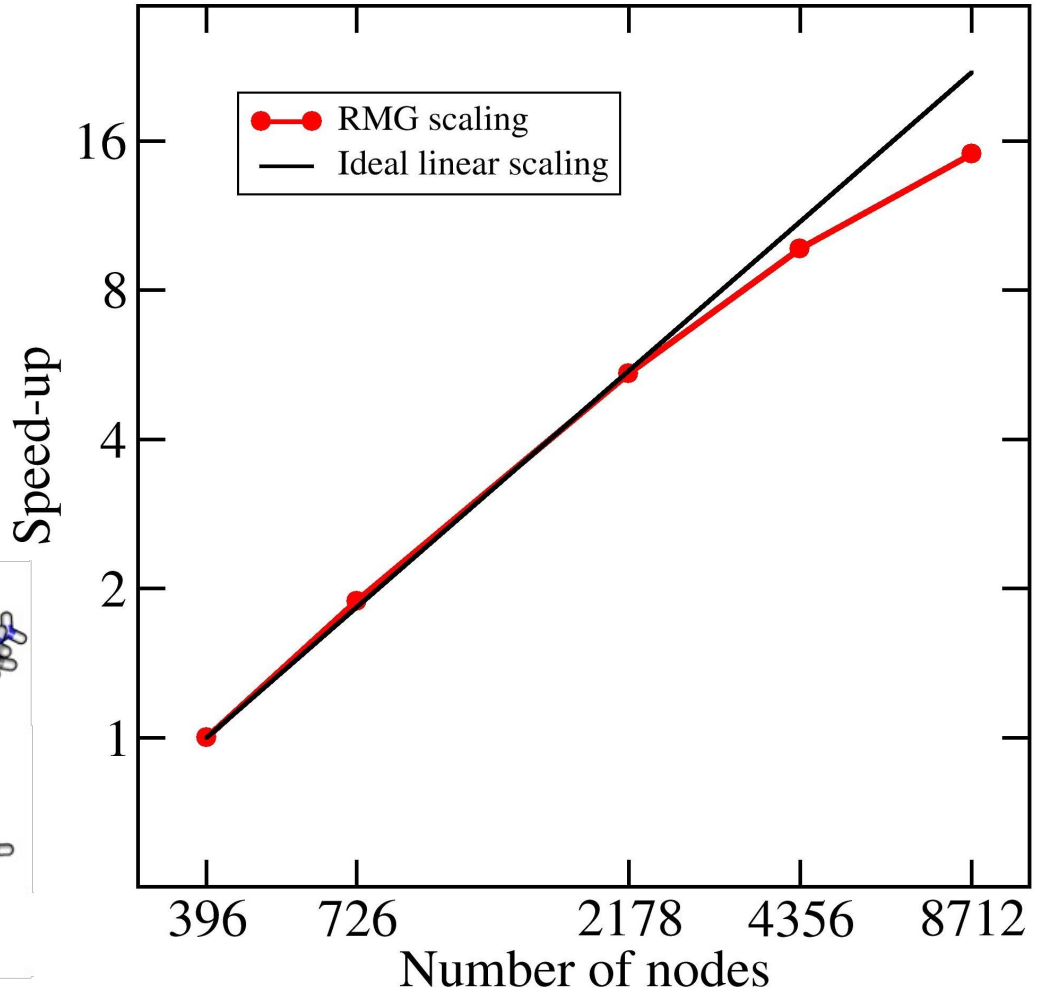
	Overhead process	Free RAM node
1 MPI process/core	0.5 Gbyte	16.0 Gbytes
1 MPI process/node	1.0 GByte	31.0 Gbytes

Scaling test

Test problem: Gas phase Amyloid Beta 1-42 protein
Test system: Cray XK7
1 node = 16 Opeteron cores + 1 Nvidia K20x GPU accelerator
Strong scaling

Largest run used **139,392** CPU cores and **8712** GPU's.





Each node has 16 cores

Mixed precision

Single precision can improve performance
- Effective doubling of cache size and local memory bandwidth
- Effective doubling of network communication bandwidth
- Little impact on network latency

Initial tests in single precision
- Much faster but accuracy not sufficient for many cases
- Some problems exhibited numerical instability or failure to converge

Alternative approach using mixed precision
- Multigrid preconditioner implemented in single precision
- Orthogonalization and diagonalization performed in double precision
- (Mixed precision via Cray IRT failed)
  - Cray IRT (Iterative refinement toolkit) consists of mixed precision solvers that use 32 bit factorizations to improve performance.

Accuracy and convergence tests

Test system: C60 molecule

Mixed precision total energy: -340.98136653**3** Hartree

Double precision total energy: -340.98136651**4** Hartree

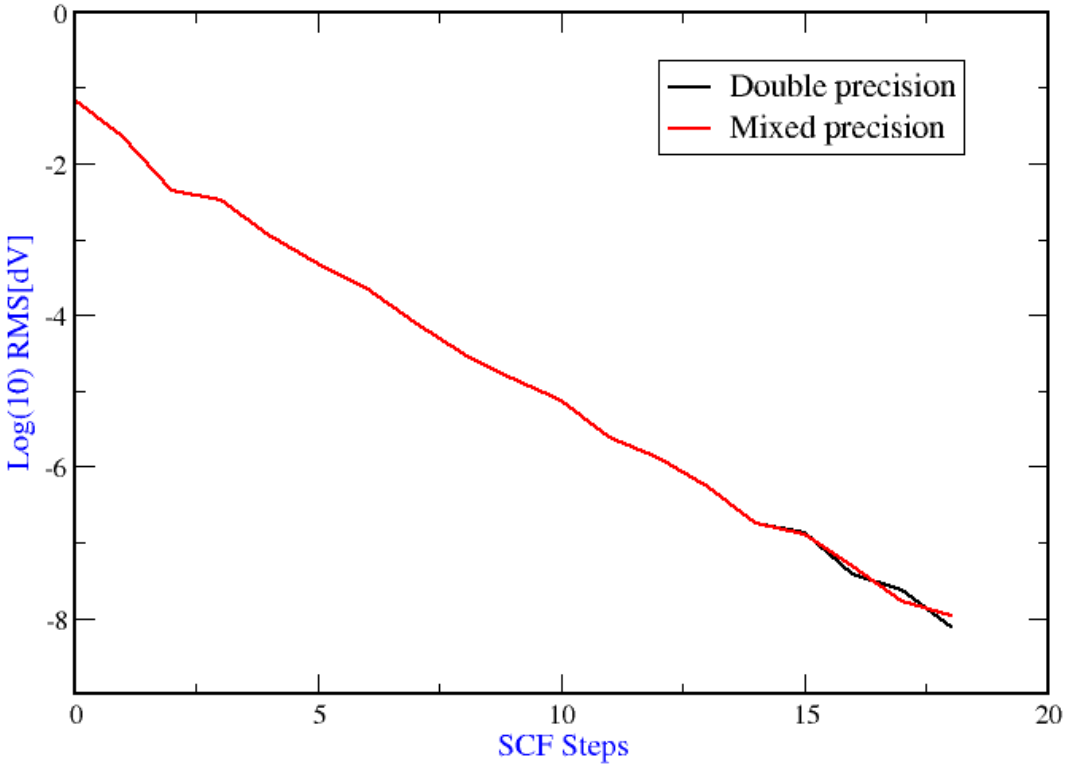
Mixed precision

6.72 secs/SCF step

Double precision

8.12 secs/SCF step

Speedup = **1.2**



GPU/Accelerators

Top 500 list (November 2012): 62 machines with accelerators

Titan at ORNL Cray XK7: 27.1 peak PFLOPS and 17.59 Linpack PFLOPS

18,688 CPU's with **299,008** cores	2.8 DP PFLOPS
18,688 Nvidia K20x GPU's	24.3 DP PFLOPS

Blue Waters at NCSA Cray XE6/XK7: 11.61 peak PFLOPS

24,576 CPU's with **393,216** cores
3,072 Nvidia K20x GPU's

GPU characteristics

Highly parallel
Low clock speed
Large memory bandwidth

Example architecture Nvidia K20x

7.1 billion transistors
732 MHz clock speed
15 SMX units
2688 cores
6 memory controllers

3.7 TFLOP SP
1.3 TFLOP DP
250 GB/sec memory Bandwidth

PCI Express System Interface

Programming Models: CPU vs GPU

GPU programming very different from CPU

CPU:

- High clock speed, small number of powerful execution units.
- Memory latency hidden by caches and out of order execution.
- Good single-threaded performance.

GPU:

- Low clock speed, large number of weaker execution units.
- Memory latency hidden by high thread counts.
- Poor single-threaded performance.

Most HPC codes have components that only run well on CPU's

- Mixed CPU/GPU model required
- Data transfer issues from CPU to GPU (PCI bus latency)

Hints:

- Avoid writing GPU code as much as possible.
- Use vendor supplied libraries.
- Data transfer issues still require careful consideration

GPU/CPU/Network data transfers

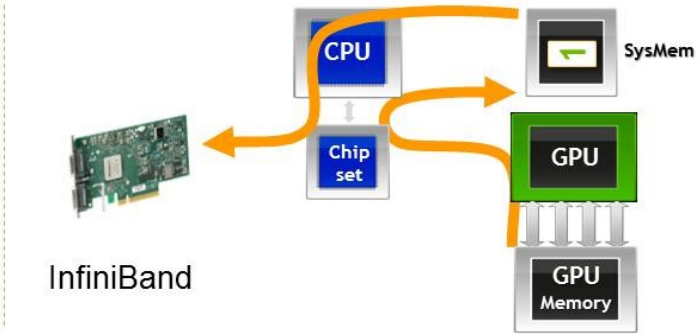
High performance GPU's use a separate memory space from the CPU's
Data transfers between GPU and GPU-RAM peak bandwidth 250GB/s, latency 200ns
Data transfers between CPU and SystemRAM peak bandwidth 25-100 GB/s, latency 50ns
Data transfers across PCI Express v2.x bus peak bandwidth 8GB/s, latency 200ns
Computational tasks suitable for GPU – long running, high ratio FLOPS/mem

Further issue: GPU data transfers between network nodes have to traverse CPU

With GPUDirect

Data only copied twice

Sharing pinned system memory makes sysmem-to-sysmem copy unnecessary

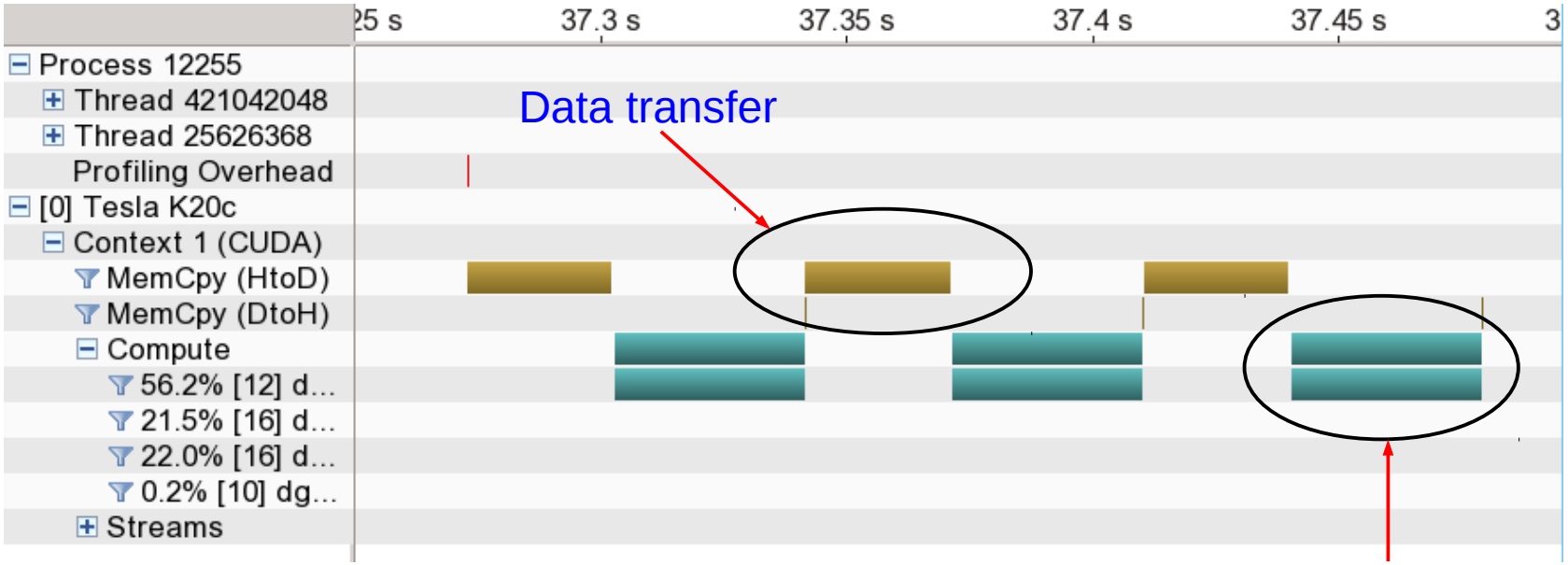


CPU/GPU case 1

Test system: C60 molecule with 200 total electronic orbitals

Compute matrix elements $\langle\psi|A|\psi\rangle$ using GPU dgemm

M=200,N=200,K=110,592 GPU dgemm = 294 GFLOPS

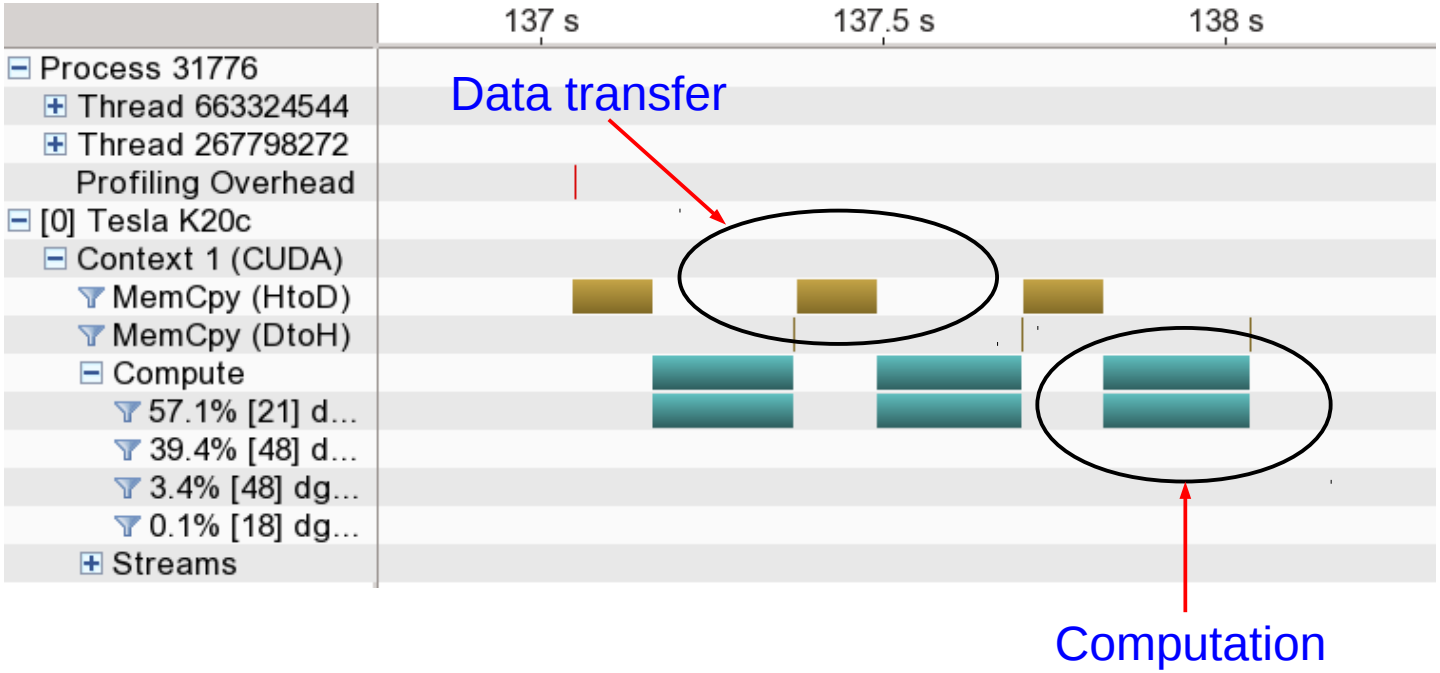


Transfer overhead comparable to computation time

GPU utilization low 22% of peak (peak=1.3 TFLOP)

CPU/GPU case 2

Test system: C60 molecule with 800 total electronic orbitals
Compute matrix elements $\langle\psi|A|\psi\rangle$ using GPU dgemm
M=800,N=800,K=110,592 GPU DGEMM 781 GFLOPS



Transfer fraction smaller and computational speed 2.7x faster
GPU utilization at 60% of peak (peak = 1.3 TFLOP)
GPU's are best suited for bigger problems! N^3 scaling of diagonalization when going from 200 orbitals to 800.

GPU performance improvements

Small test case: C60 molecule in vacuum

60 atoms: 200 electronic orbitals

CPU only calculation
Xeon workstation:12 cores
No GPU's
10.32 seconds/SCF step

CPU/GPU calculation
Xeon workstation:12 cores
1 Nvidia K20 GPU
6.72 seconds/SCF step

Speedup of approximately 1.53

Large test case: Solvated amyloid beta protein fragment

3337 atoms: 4672 electronic orbitals

CPU only calculation
2904 nodes: 92,928 Opteron cores
No GPU's
76 seconds/SCF step

CPU/GPU calculation
2904 nodes: 46,464 Opteron cores
2904 Nvidia K20x GPU's
25 seconds/SCF step

Speedup of approximately 3.02

Summary

HPC performance coming more and more from multicore and GPU.
Different programming methods required to fully utilize the hardware.

Hybrid threaded model used to reduce the number of MPI process's.
Marked improvement in scalability for large problems.

Mixed precision methods can provide substantial performance gains
Not all problems are suitable for mixed precision. Careful tests needed.

GPU accelerators can provide order of magnitude gains on some tasks.
Programming is difficult compared to CPU's. Vendor libraries best choice.
Data transfer issues between CPU/GPU/Network critical.