Direct Self-Consistent Field Computations on GPU Clusters

Guochun Shi,
Volodymyr Kindratenko

National Center for Supercomputing Applications

University of Illinois at Urbana-Champaign

Ivan Ufimtsev,
Todd Martinez

Department of Chemistry

Stanford University

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Presentation Outline

GPU computing

NCSA's Lincoln GPU cluster

SCF theory in Quantum Chemistry

Implementation on a GPU cluster

Kernels for J and K matrices

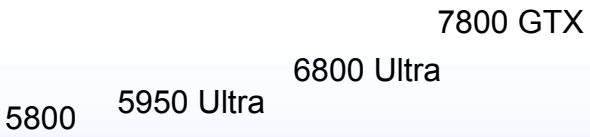
Parallelization strategy for GPU cluster

Performance

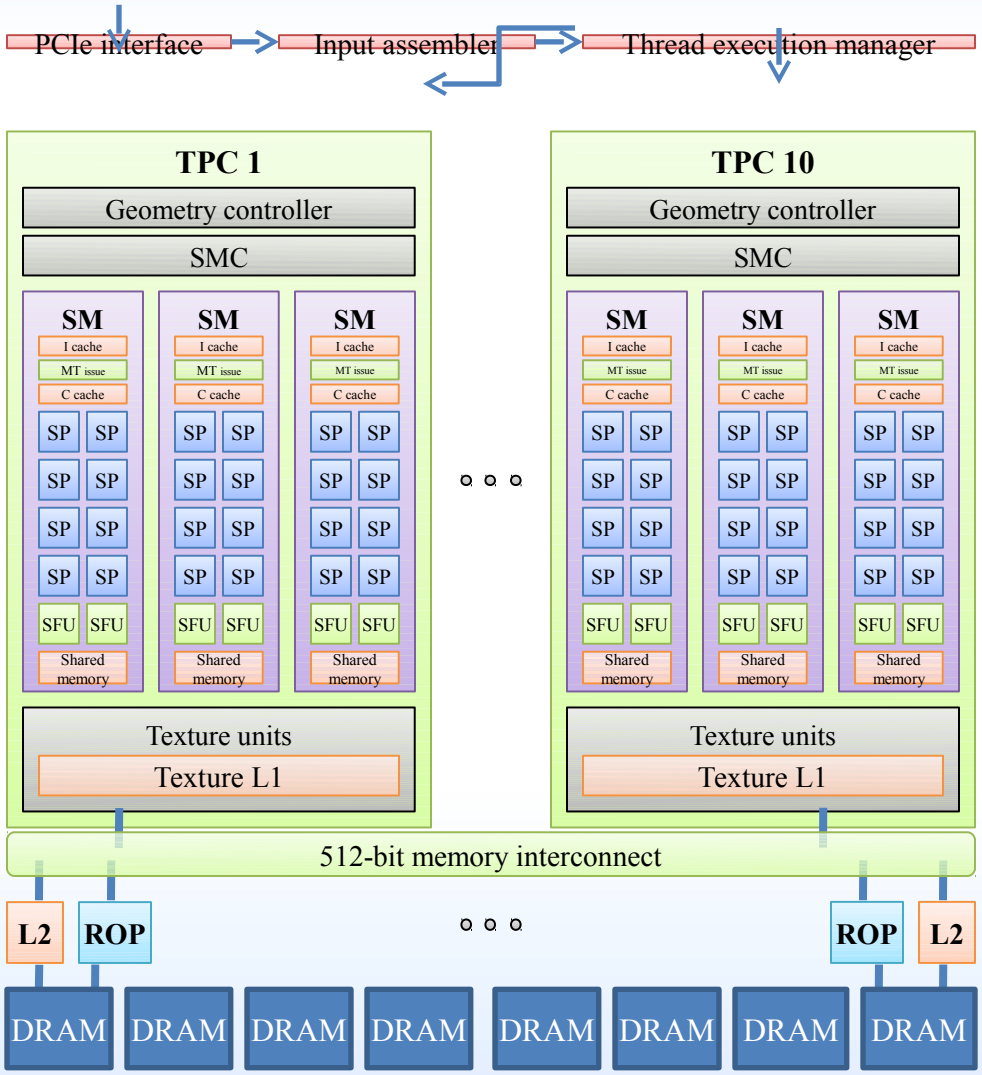
Conclusions and future work

Why GPUs?

GPU Performance Trends



NVIDIA Tesla T10 GPU Architecture



T10 architecture

240 streaming processors arranged as 30 streaming multiprocessors

At 1.3 GHz this provides

- 1 TFLOP SP
- 86.4 GFLOP DP

512-bit interface to off-chip GDDR3 memory

- 102 GB/s bandwidth

Intel 64 Tesla Linux Cluster *Lin*

Dell PowerEdge 1955 server

Intel 64 (Harpertown) 2.33 GHz dual socket quad core

16 GB DDR2

Infiniband SDR

Tesla S1070 1U GPU Computing Server

1.3 GHz Tesla T10 processors

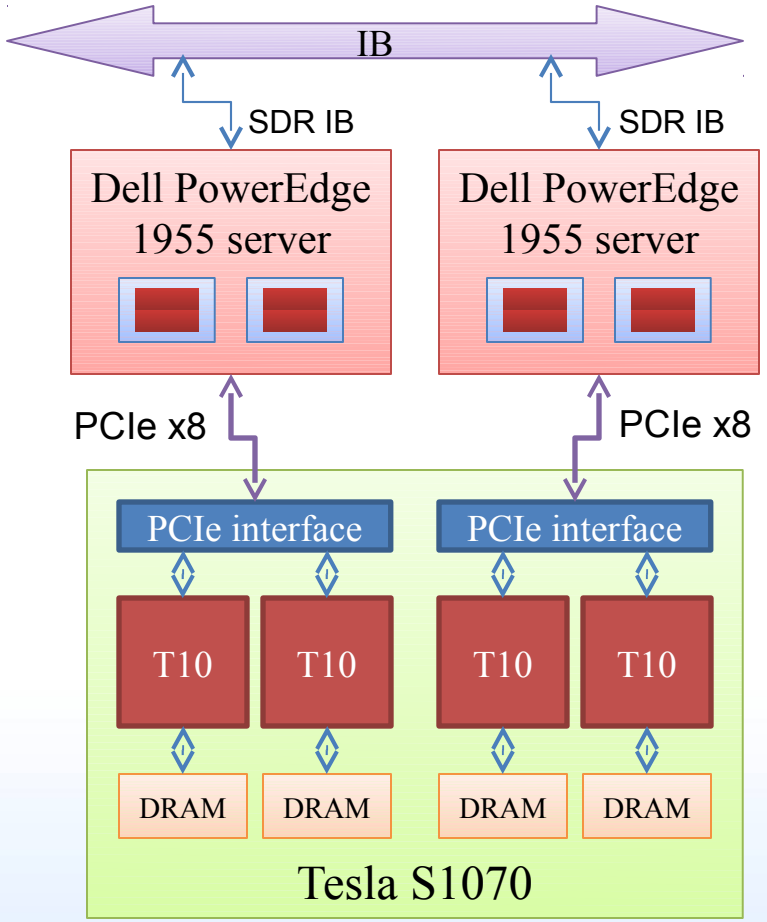
4x4 GB GDDR3 SDRAM

Cluster

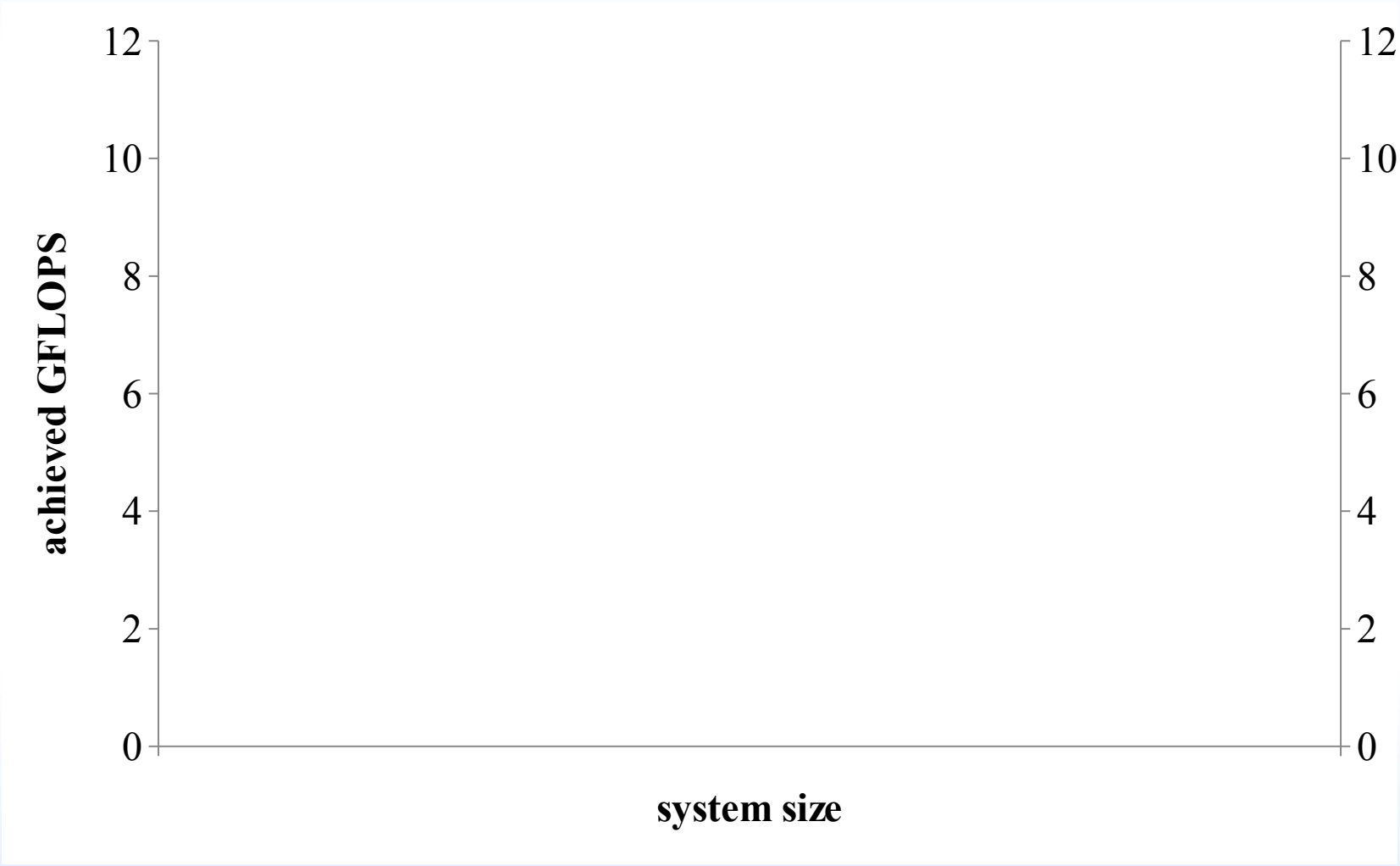
Servers: 192

Accelerator Units: 96

Two Compute Nodes



HPL Benchmark for Lincoln



We used Massimiliano Fatica(nvidia)'s GPU enabled HPL package.

Quantum Chemistry

Why do we need to deal with…

Energy (H = *E* ▪):

Quantifies intra/intermolecular interactions

Drives chemistry, little interesting happens on flat surface

Geometry optimization (R*E* = 0)

Searches for stable atomic arrangements (molecular shapes)

Molecular dynamics (∂2R/ ∂t2 = -1/M ▪ R*E*)

The chemistry itself (at some, sometimes crude, approximation)

Studies system at atomistic time, and length scales

Exact energy is a hard problem

$$\Psi(\mathbf{r_i}) = ?$$

$$E = ?$$

$$\left\{ -\frac{1}{2}\sum_i \left(\frac{\partial^2}{\partial x_i^2} + \frac{\partial^2}{\partial y_i^2} + \frac{\partial^2}{\partial z_i^2} \right) - \sum_{i,A} \frac{Z_A}{|\mathbf{r_i} - \mathbf{R_A}|} + \sum_{i,j} \frac{1}{|\mathbf{r_i} - \mathbf{r_j}|} \right\} \Psi(\mathbf{r_i}) = E\Psi(\mathbf{r_i})$$

Hartree-Fock approximation is one of the simplest

is an antisymmetrized product of *N* 1-electron orbitals

$$\Psi = A\left[\psi_1(r_1)\psi_2(r_2)...\psi_N(r_N)\right]$$

Expand over predefined basis set

$$\psi_i(r) = \sum_{j=1}^{K} C_{ij}\varphi_j(r)$$

$$\Psi \leftrightarrow C_{ij} = ?$$

Hartree-Fock Self Consistent Field (SCF) procedure

$$\mathbf{F}(\mathbf{C})\mathbf{C} = E\mathbf{S}\mathbf{C}$$

$$\mathbf{F}_{k+1}(\mathbf{C}) = \mathbf{F}(\mathbf{C}_k)$$

$$\mathbf{F}_{k+1}\mathbf{C}_{k+1} = E\mathbf{S}\mathbf{C}_{k+1}$$

Repeat until Ck+1 more or less equals Ck

Hartree-Fock equations

$$\mathbf{F}(\mathbf{C})\mathbf{C} = E\mathbf{S}\mathbf{C}$$

$$F_{ij}(\mathbf{C}) = H_{ij}^{core} + J_{ij}(\mathbf{C}) - \frac{1}{2}K_{ij}(\mathbf{C})$$

$$J_{ij} = \sum_{k,l}[ij \mid kl]P_{kl}(\mathbf{C})$$

$$K_{ij} = \sum_{k,l}[ik \mid jl]P_{kl}(\mathbf{C})$$

$$[ij \mid kl] = \iint \varphi_i(r_1)\varphi_j(r_1)\frac{1}{|r_1 - r_2|}\varphi_k(r_2)\varphi_l(r_2)d\mathbf{r_1}d\mathbf{r_2}$$

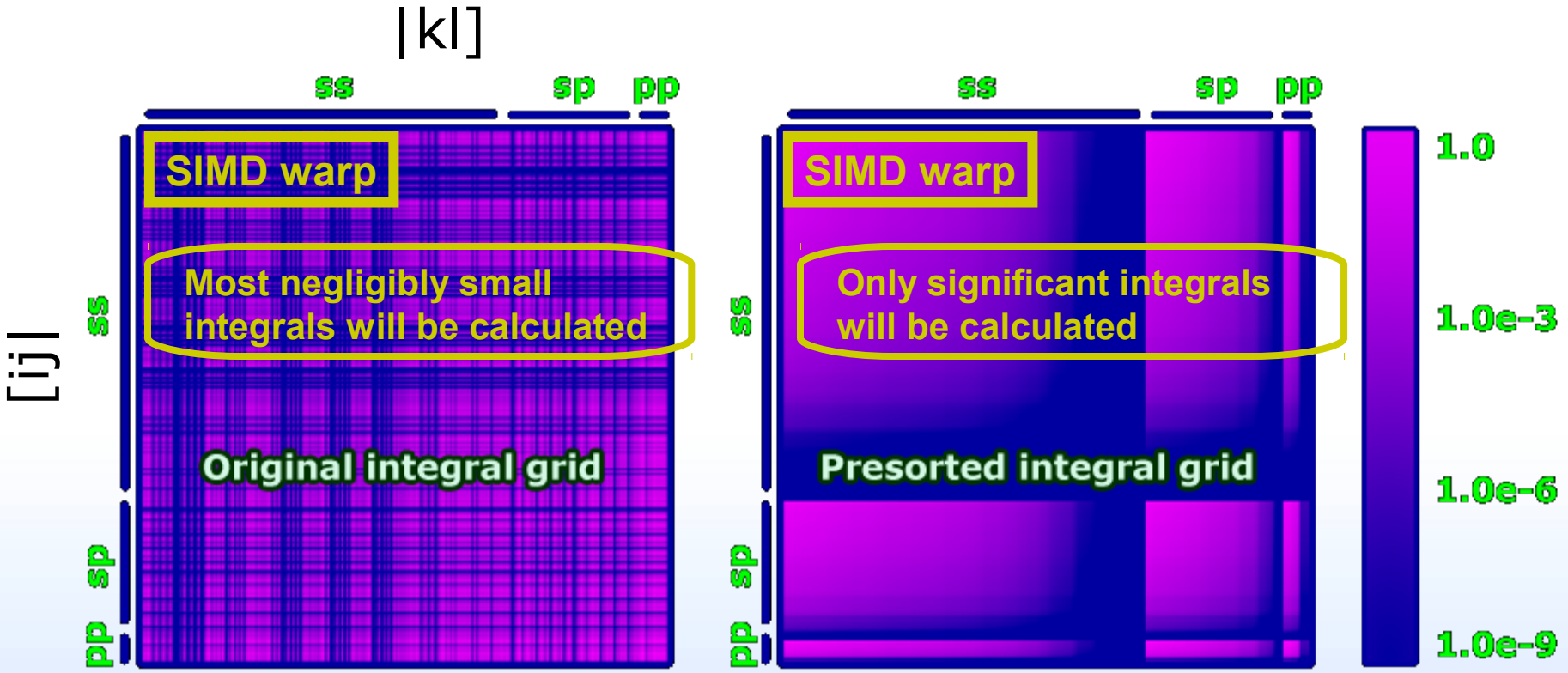
- All matrices are of N N size (N ~ 1,000 … 10,000)
- $N3$ operations to solve HF equations (need to deal with diagonalization)
- $N4$ operations to get **F**

Kernel In GPU

2e integral grid

$$[ij \mid kl] = \iint \varphi_i(r_1)\varphi_j(r_1)\frac{1}{|r_1 - r_2|}\varphi_k(r_2)\varphi_l(r_2)d\mathbf{r_1}d\mathbf{r_2}$$

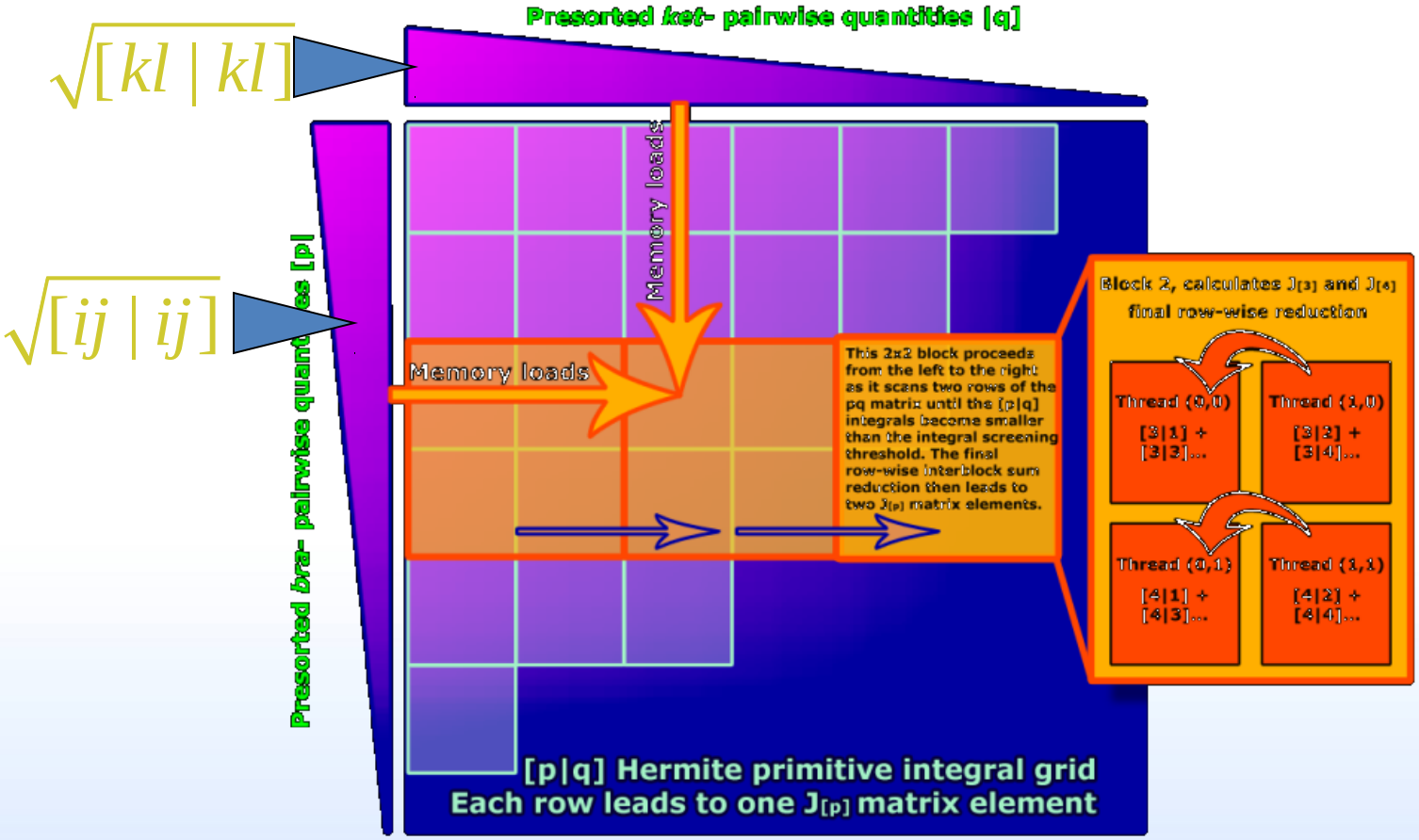
$[ij \mid kl] \le \sqrt{[ij \mid ij]}\sqrt{[kl \mid kl]} \ge 10^{-11}$ leaves only *N*2 out of *N*4 integrals



Kernel in GPU: J-matrix implementation

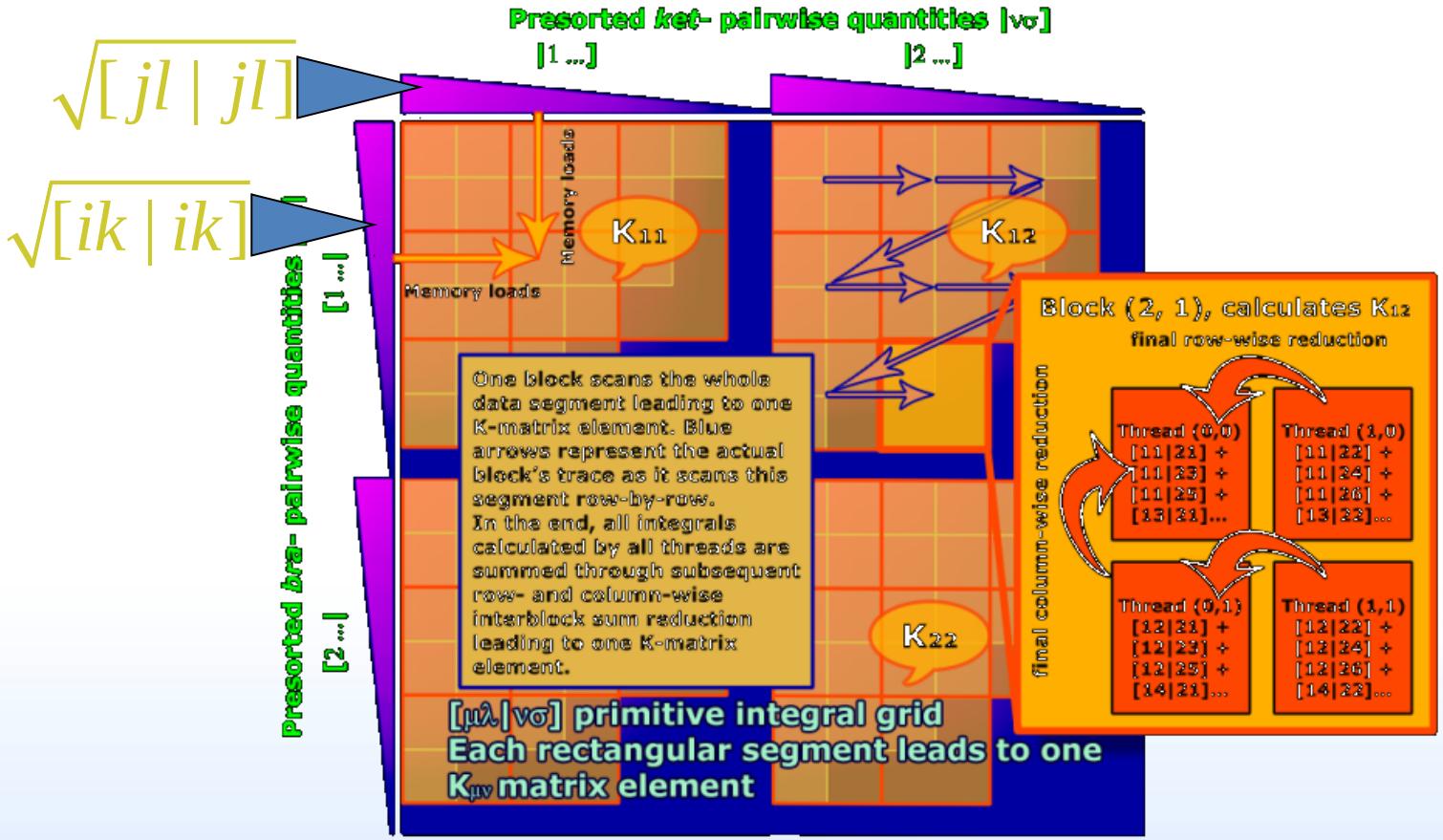
$$J_{ij} = \sum_{k,l} [ij \mid kl] P_{kl}$$

$$[ij \mid kl] \le \sqrt{[ij \mid ij]}\sqrt{[kl \mid kl]}$$

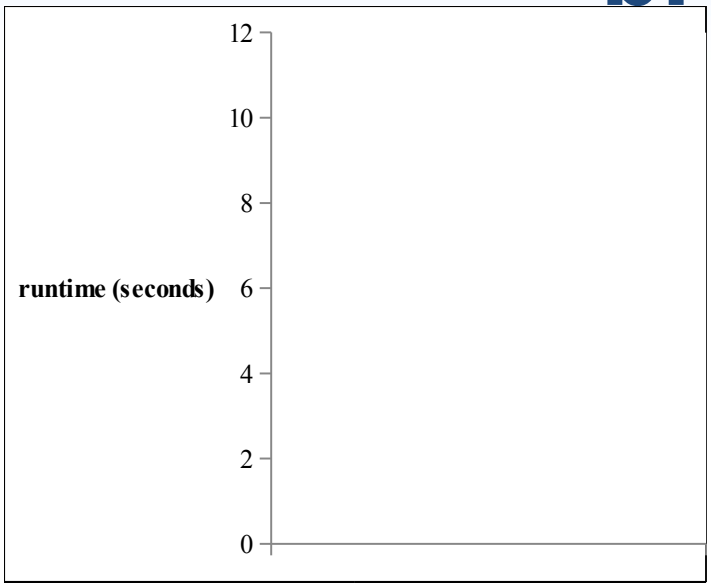


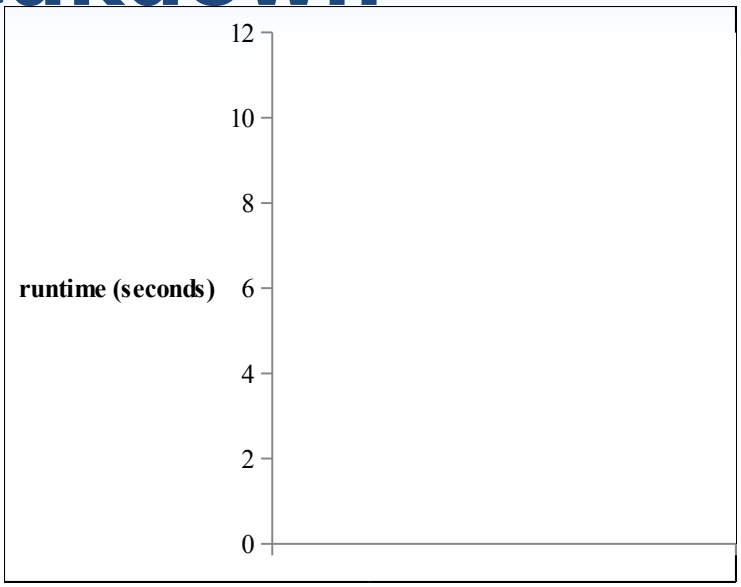
Kernels in GPU: K-matrix implementation

$$K_{ij} = \sum_{k,l} [ik \mid jl] P_{kl}$$



Singe node execution time breakdown





- The J and K matrices computation and Linear Algebra (LA) computation dominate the overall execution time
- Pair quantity computations can be significant

GPU cluster parallelization strategy

Each GPU has a global id

nodeid * num_gpu_per_node + local_gpu_index

J/K matrices work distribution

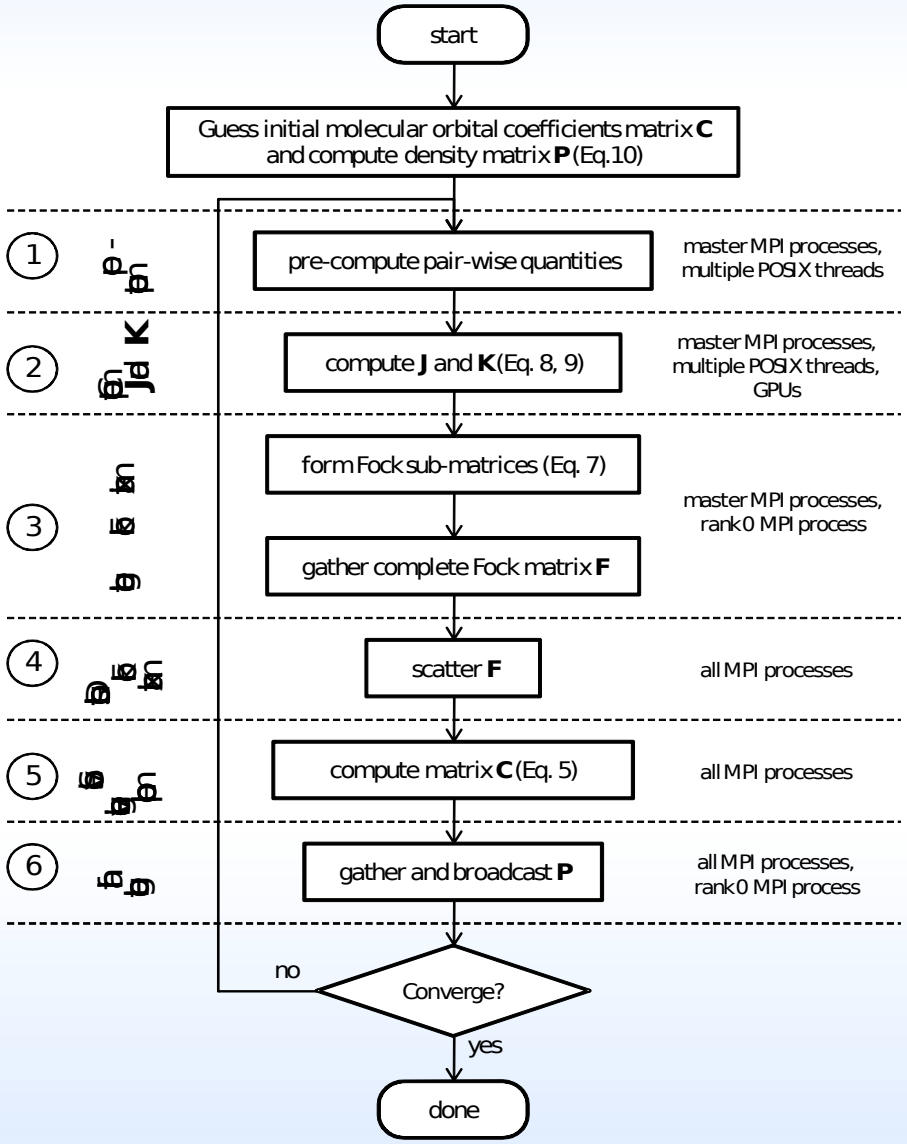
Computations for elements in J and K matrices are not even.

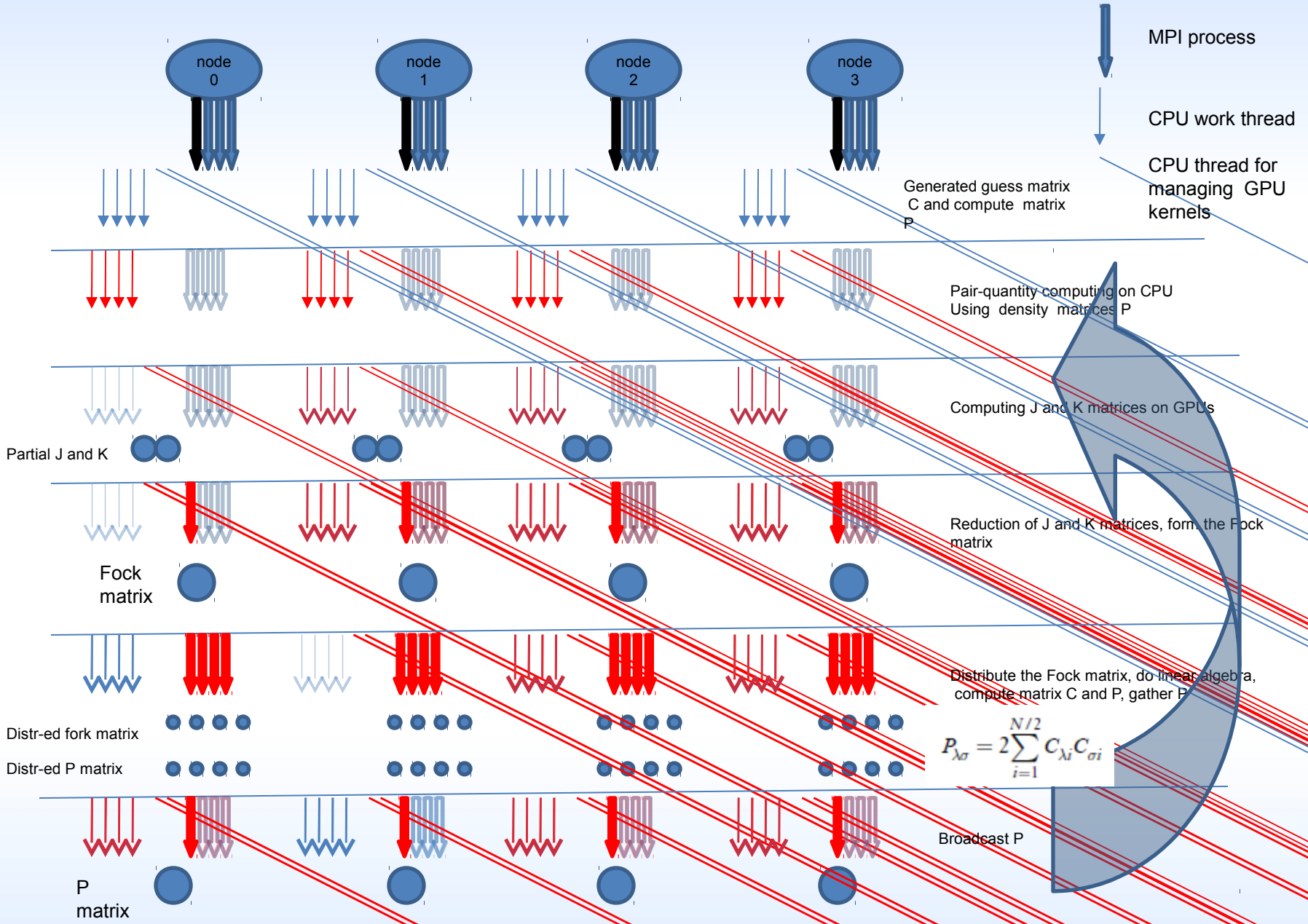
Sort pre-computed pair quantities and choose every one element in N to compute for each GPU

LA using intel MKL

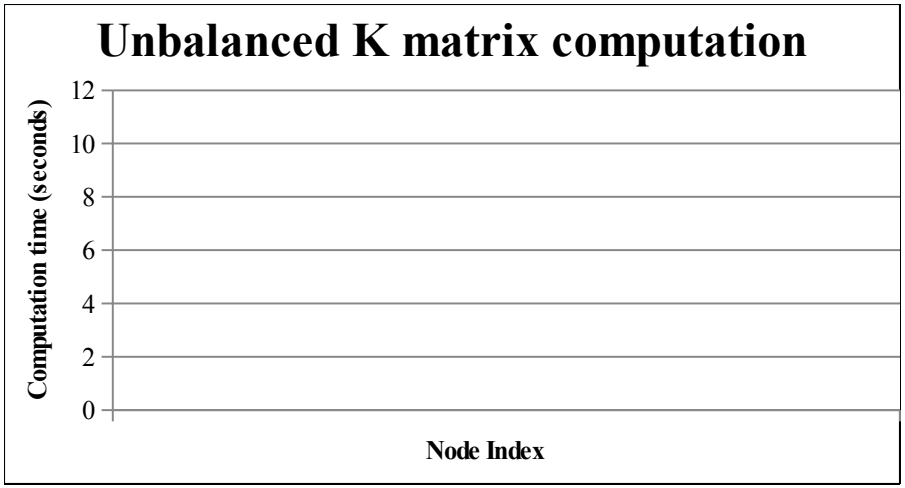
Parallelization strategy (II)

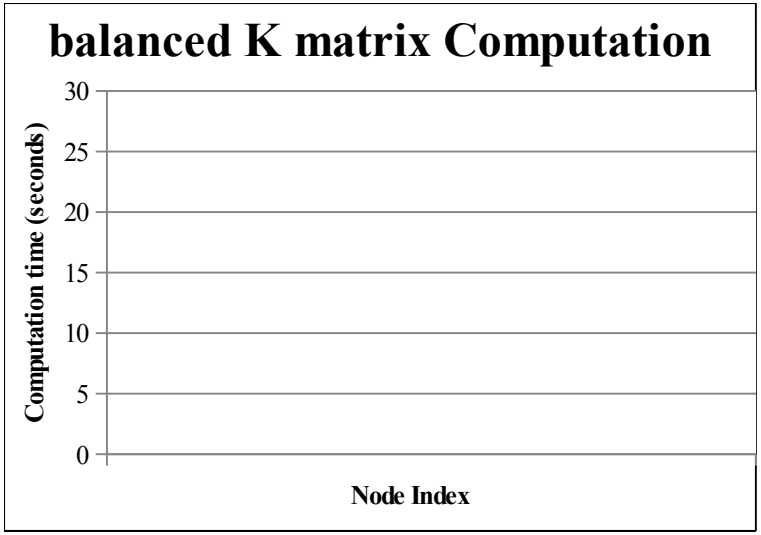
- Start as MPI program, each node has as many MPI processes as CPU cores
- One MPI process per node is designated as “master”
- The master MPI processes create threads for controlling GPUs as well as CPU work threads
- MPI processes/GPU management threads/CPU work threads are awaken or put to sleep as needed



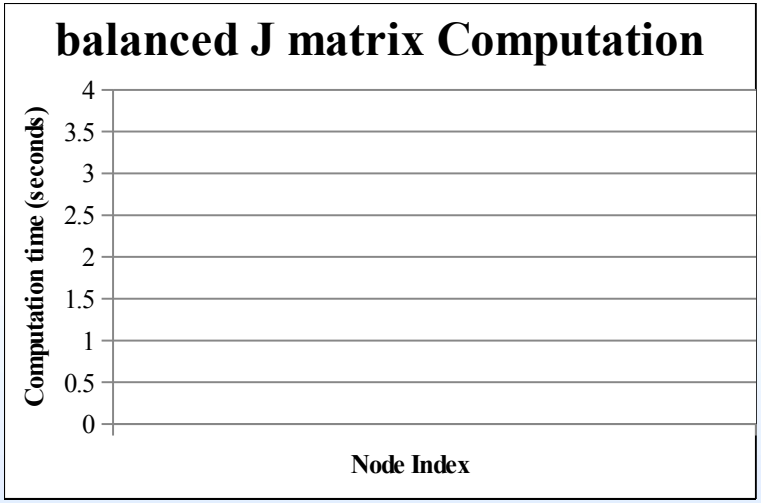

node 0
node 1
node 2
node 3
MPI process
CPU work thread
CPU thread for managing GPU kernels
Generated guess matrix C and compute matrix P
Pair-quantity computing on CPU Using density matrices P
Computing J and K matrices on GPUs
Partial J and K
Reduction of J and K matrices, form the Fock matrix
Fock matrix
Distribute the Fock matrix, do linear algebra, compute matrix C and P, gather P
Distr-ed fork matrix
Distr-ed P matrix
$P_{\lambda\sigma} = 2\sum_{i=1}^{N/2} C_{\lambda i} C_{\sigma i}$
Broadcast P
P matrix

Performance: load balancing



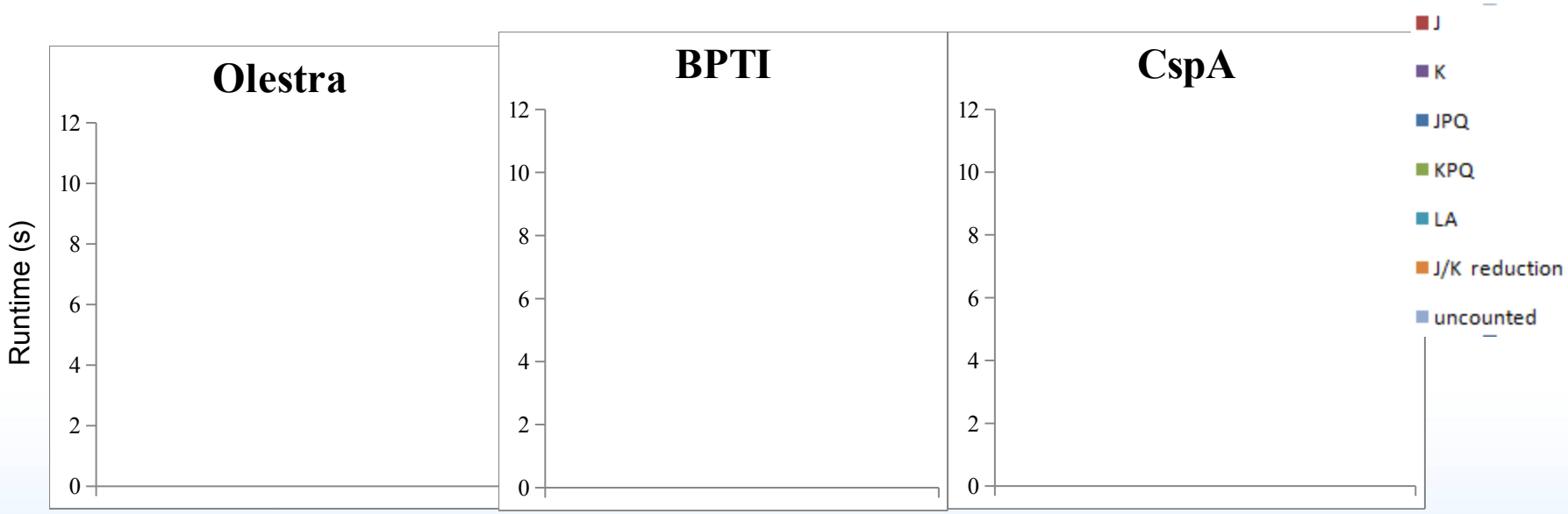


- Sorting for pair quantity computations and work selection strategy makes the computation on GPUs well balanced, reducing performance degradation



Performance

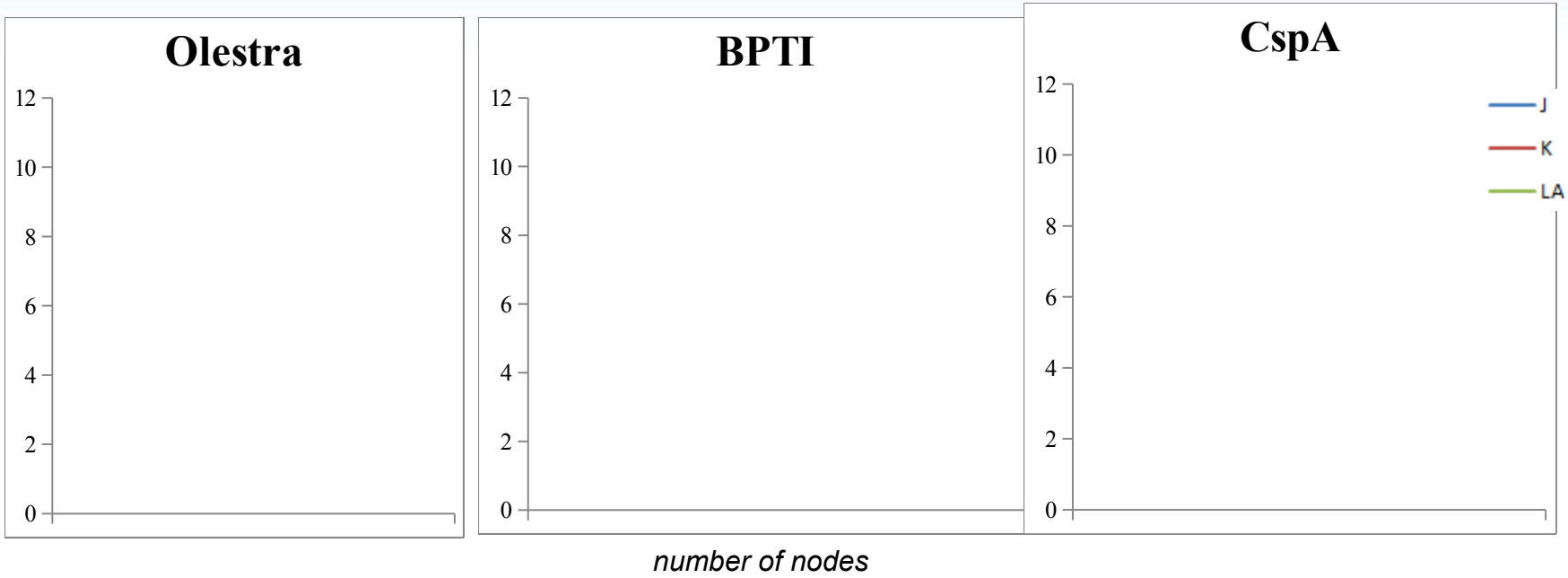
	Atoms	Electrons	Orbitals	S shells	P shells
Olestra	453	1366	2131	1081	350
BPTI	875	3400	4893	2202	897
CspA	1732	6290	8753	4220	1511



# of nodes

Using 321g basis set

Scalability of J, K and LA

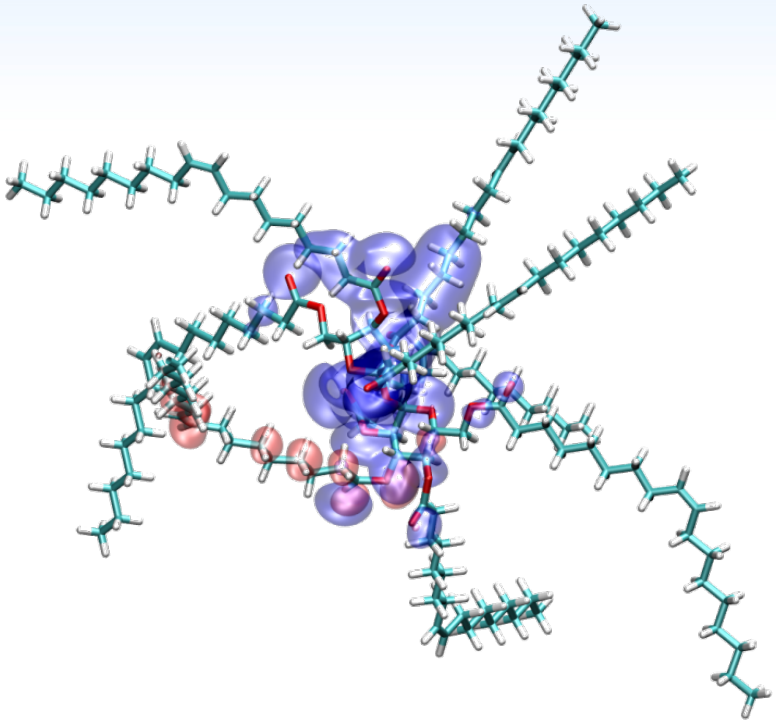


- J and K matrices computation can scale well to 128 nodes
- Linear Algebra scales only up to 16 nodes even for CsPA molecule

Performance: Linear Algebra breakdown

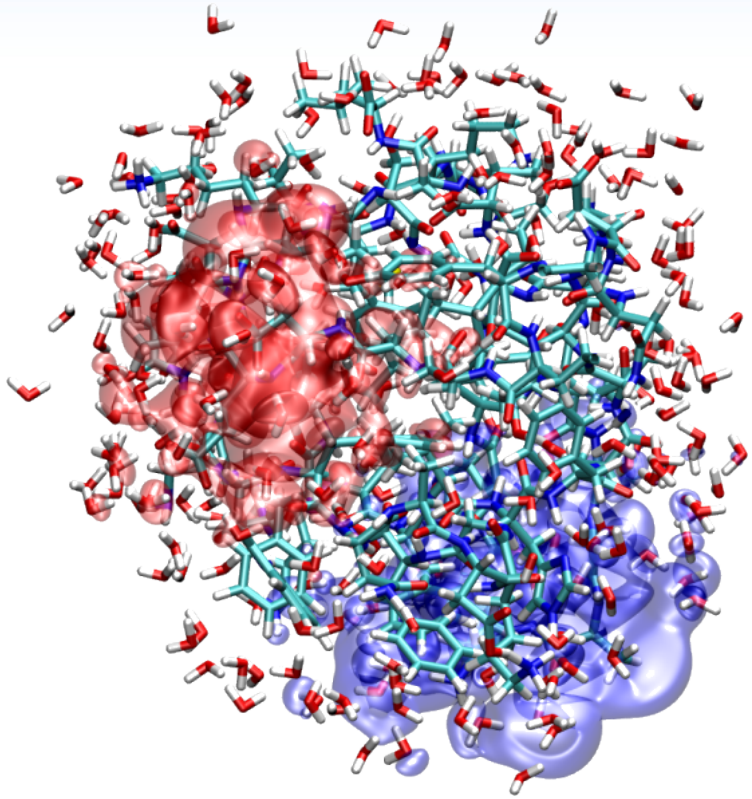
- Diagonization scales the worst, dgemm is also important
- A fast, scalable GPU based SCALAPACK is needed
  - Magma from UTK?
  - Cula?

Results: Olestra molecule



Olestra molecule consisting of 453 atoms (a small example model used of testing the developed software) can be computed by the state-of-the-art quantum chemistry software package GAMESS running on an Intel Pentium D 3 GHz processor in over 12,408 seconds whereas our 8-node GPU cluster implementation performs the same computation in just over 5 seconds, a 2,452× speedup.

Example: CspA molecule



For larger models, one SCF iteration for Cold shock protein A (CspA) molecule consisting of 1,732 atoms can be done in 88 seconds on a 16 node GPU cluster.

Conclusions and future work

GPU computing brings Quantum Chemistry computing to a new level

Parallelization enables computing of large molecules in shorter time

J and K matrices show good scalability

Linear Algebra can only scale up to 16 nodes

Linear Algebra becomes a major bottleneck

A linear algebra package using GPUs with good scalability is needed

- Matrix multiplication and eigenvalue solver

Only S and P orbitals are supported at this moment

Acknowledgement

This work was supported by the National Science Foundation grant CHE-06-26354.