Partitioning and Load Balancing in Parallel Quantum Chemistry

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Workshop on Recent Advances in Parallel and High Performance Computing Techniques and Applications
National University of Singapore, 12–16 January 2015
Large-scale quantum chemistry

- Input: a set of atoms and their coordinates
- Computation: approximately solve Schrödinger’s equation
- Output: wave function and properties of the system
- Many applications: materials science, biochemistry, pharmacology, chemical physics, chemical engineering, energy research, etc.
- Many parallel codes: NWChem, GAMESS, ACES III, MPQC, etc.
Focus and contribution

- Focus of this paper
  - Hartree-Fock self-consistent field (SCF) iteration
  - Highly parallel distributed computation

- GTFock code
  - Fock matrix construction: load balancing, reducing communication
  - Uses density matrix purification rather than eigendecomposition
  - Optimization of integral calculations in Fock matrix construction
  - Heterogeneous Intel Xeon Phi calculation on large clusters

Related work on distributed Fock matrix construction
1) Furlani and King 1995
2) Foster, Tilson, Wagner, Shepard, Harrison, Kendall, Littlefield 1996
4) Alexeev, Kendall, Gordon 2002
5) Takashima, Yamada, Obara, Kitamura, Inabata, Miyakawa, et al. 2002
6) Janssen and Nielsen 2008
7) Ishimura, Kuramoto, Ikuta, Hyodo 2010
8) Liu, Patel, Chow 2014
Basis functions

- Wave functions expressed in terms of molecular orbitals \( \psi_i(r) \)
- Molecular orbitals are linear combinations of basis functions,

\[
\psi_i(r) = \sum_{j=1}^{n} c_{ij} \phi_j(r),
\]

where \( n \) is the number of basis functions, and where the \( c_{ij} \) are to be determined.
Hartree-Fock SCF algorithm

Solve for $C$ in the nonlinear generalized eigenvalue problem:

$$F(C)\, C = S \, C \, \varepsilon,$$

all matrices $F, C, S, \varepsilon \in \mathbb{R}^{n \times n}$

where $F$ is the Fock matrix, and $D = C_{occ}C_{occ}^{T}$ is the density matrix.

1. Guess $D$
2. Compute $H^{\text{core}}$
3. Diagonalize $S = UsU^{T}$
4. Form $X = Us^{1/2}$
5. repeat
6. Construct $F$, which is a function of $D$
7. Form $F' = X^{T}FX$
8. Diagonalize $F' = C'\varepsilon C'^{T}$
9. $C = XC'$
10. Form $D = C_{occ}C_{occ}^{T}$
11. until converged
Hartree-Fock SCF algorithm

Solve for $C$ in the nonlinear generalized eigenvalue problem:

$$F(C) \ C = S \ C \ \varepsilon,$$

all matrices $F, C, S, \varepsilon \in \mathbb{R}^{n \times n}$

where $F$ is the Fock matrix, and $D = C_{occ} C_{occ}^T$ is the density matrix.

1. Guess $D$
2. Compute $H^{\text{core}}$
3. Diagonalize $S = U \sigma U^T$
4. Form $X = U \sigma^{1/2}$

5. repeat
6. Construct $F$, which is a function of $D$
7. Form $F' = X^T F X$
8. Diagonalize $F' = C' \varepsilon C'^T$
9. $C = X C'$
10. Form $D = C_{occ} C_{occ}^T$

11. until converged
Improved scalability of Hartree-Fock calculations

1hsg_38 with 3555 basis functions on Stampede
Improved scalability of Hartree-Fock calculations

![Graph showing improved scalability of Hartree-Fock calculations.

1hsg_38 with 3555 basis functions on Stampede]
Fock matrix and electron repulsion integrals

\[ F_{ij} = H_{ij}^{\text{core}} + \sum_{kl} D_{kl} \left( 2(ij|kl) - (ik|jl) \right) \]

where \((ij|kl)\) denotes an element of a 4-index electron repulsion integral (ERI) tensor.

\[ (ij|kl) = \int \phi_i(x_1)\phi_j(x_1)r_{12}^{-1}\phi_k(x_2)\phi_l(x_2)dx_1\,dx_2 \]

where \(x_1\) and \(x_2\) are coordinates in \(\mathbb{R}^3\), and \(r_{12} = \|x_1 - x_2\|\).

Symmetry
\[(ij|kl) = (ji|kl) = (ij|lk) = (kl|ij) = \cdots\]

Schwarz Screening
\[(ij|kl) \leq \sqrt{(ij|ij)} \sqrt{(kl|kl)} \leq \tau\]
Numerically nonzero shell quartets

System: C_{24}H_{50} with cc-pVDZ (294 shells, 586 functions)
Coulomb and exchange matrices

\[ F_{ij} = H_{ij}^{\text{core}} + 2 \sum_{kl} D_{kl}(ij|kl) - \sum_{kl} D_{kl}(ik|jl) \]

\[ F = H^{\text{core}} + 2J - K \]

A group of basis functions is called a *shell*. Suppose we have computed the *shell quartet* \((MN|PQ)\), then

\[ J_{MN} = \sum D_{PQ}(MN|PQ) \quad J_{NM} = \sum D_{PQ}(NM|PQ) \]
\[ J_{MN} = \sum D_{QP}(MN|QP) \quad J_{NM} = \sum D_{QP}(NM|QP) \]
\[ J_{PQ} = \sum D_{MN}(PQ|MN) \quad J_{QP} = \sum D_{MN}(QP|MN) \]
\[ J_{PQ} = \sum D_{NM}(PQ|NM) \quad J_{QP} = \sum D_{NM}(QP|NM) \]
\[ K_{NQ} = \sum D_{MP}(MN|PQ) \quad K_{MQ} = \sum D_{NP}(NM|PQ) \]
\[ K_{NP} = \sum D_{MQ}(MN|QP) \quad K_{MP} = \sum D_{NQ}(NM|QP) \]
\[ K_{QN} = \sum D_{PM}(PQ|MN) \quad K_{PN} = \sum D_{QM}(QP|MN) \]
\[ K_{QM} = \sum D_{PN}(PQ|NM) \quad K_{PM} = \sum D_{QN}(QP|NM) \]
Coulomb and exchange matrices

\[ F_{ij} = H_{ij}^{\text{core}} + 2 \sum_{kl} D_{kl}(ij|kl) - \sum_{kl} D_{kl}(ik|jl) \]

\[ F = H^{\text{core}} + 2J - K \]

A group of basis functions is called a shell.
Suppose we have computed the shell quartet \((MN|PQ)\), then

\[ J_{MN} = \sum D_{PQ}(MN|PQ) \]
\[ J_{NM} = \sum D_{PQ}(NM|PQ) \]
\[ J_{MP} = \sum D_{PQ}(MN|QP) \]
\[ J_{NP} = \sum D_{PQ}(NM|QP) \]
\[ J_{QP} = \sum D_{MN}(PQ|MN) \]
\[ J_{QM} = \sum D_{MN}(PQ|NM) \]
\[ J_{PN} = \sum D_{MN}(QP|MN) \]
\[ J_{MP} = \sum D_{MN}(QP|NM) \]
\[ J_{MN} = \sum D_{PQ}(MN|QP) \]
\[ J_{NM} = \sum D_{PQ}(NM|QP) \]
\[ J_{MP} = \sum D_{PQ}(MN|QP) \]
\[ J_{NP} = \sum D_{PQ}(NM|QP) \]
\[ J_{QP} = \sum D_{MN}(PQ|MN) \]
\[ J_{QM} = \sum D_{MN}(PQ|NM) \]
\[ J_{PN} = \sum D_{MN}(QP|MN) \]
\[ J_{PM} = \sum D_{MN}(QP|NM) \]

\[ K_{NQ} = \sum D_{MP}(MN|PQ) \]
\[ K_{MQ} = \sum D_{NP}(NM|PQ) \]
\[ K_{MP} = \sum D_{NP}(MN|QP) \]
\[ K_{NP} = \sum D_{MN}(QP|MN) \]
\[ K_{QM} = \sum D_{NP}(PQ|NM) \]
\[ K_{QP} = \sum D_{MN}(QP|MN) \]
\[ K_{PM} = \sum D_{NP}(QM|NM) \]
\[ K_{PN} = \sum D_{MN}(QP|NM) \]
Distributed computation of the Fock matrix

$F$ and $D$ are distributed; each node stores a part of $F$ and $D$

1. **for unique shell quartets $(MN|PQ)$ do**
2.  **if $(MN|PQ)$ is not screened out then**
3.    Compute shell quartet $(MN|PQ)$
4.    Receive submatrices $D_{MN}, D_{PQ}, D_{NP}, D_{MQ}, D_{NP}, D_{MP}$
5.    Compute submatrices $F_{MN}, F_{PQ}, F_{NP}, F_{MQ}, F_{NP}, F_{MP}$
6.    Send submatrices of $F$ to their owners
7.  **end**
8. **end**

Two basic options for distributed parallelization:

- Static partitioning of the shell quartets, maintaining load balance, and minimizing communication of $D$ and $F$
- Dynamic scheduling of tasks, where a task is a set of shell quartets
Static partitioning problem

Balance the number of non-screened shell quartets in each partition.

Each partition should use a minimal set of $D$ submatrices; these $D$ submatrices can be prefetched for all computations on a node.
Partitioning framework

1D partitioning for $p$ nodes

$$(\mathcal{M}_i, : | :, :) \equiv \{(MN|PQ), \text{ s.t. } M \in \mathcal{M}_i, \text{ for all } N, P, Q\}, \quad i \in \{1, \ldots, p\}$$

2D partitioning for $p_r \times p_c$ nodes

$$(\mathcal{M}_i, \mathcal{N}_j | :, :) \equiv \{(MN|PQ), \text{ s.t. } M \in \mathcal{M}_i, N \in \mathcal{N}_j, \text{ for all } P, Q\}, \quad i \in \{1, \ldots, p_r\}, \quad j \in \{1, \ldots, p_c\}$$

$$(\mathcal{M}_i, : | \mathcal{P}_k:, :) \equiv \{(MN|PQ), \text{ s.t. } M \in \mathcal{M}_i, P \in \mathcal{P}_k, \text{ for all } N, Q\}, \quad i \in \{1, \ldots, p_r\}, \quad j \in \{1, \ldots, p_c\}$$

Two “types” of 2D partitioning.
One type each of 1D, 3D, and 4D partitioning.
Generalized slices through the 4D tensor
Shell quartets, tasks, and partitions...

- Group shell quartets into tasks
- Group tasks into partitions, one for each node
Task matrix partitioning

Even grouping of indices – assumes number of non-screened shell quartets will average out

\[
\begin{bmatrix}
  T_{11} & T_{12} & T_{13} & T_{14} & T_{15} & T_{16} \\
  T_{21} & T_{22} & T_{23} & T_{24} & T_{25} & T_{26} \\
  T_{31} & T_{32} & T_{33} & T_{34} & T_{35} & T_{36} \\
  T_{41} & T_{42} & T_{43} & T_{44} & T_{45} & T_{46} \\
  T_{51} & T_{52} & T_{53} & T_{54} & T_{55} & T_{56} \\
  T_{61} & T_{62} & T_{63} & T_{64} & T_{65} & T_{66}
\end{bmatrix}
\]

Uneven grouping of indices – adjust grouping to balance number of non-screened shell quartets in each partition

\[
\begin{bmatrix}
  T_{11} & T_{12} & T_{13} & T_{14} & T_{15} & T_{16} \\
  T_{21} & T_{22} & T_{23} & T_{24} & T_{25} & T_{26} \\
  T_{31} & T_{32} & T_{33} & T_{34} & T_{35} & T_{36} \\
  T_{41} & T_{42} & T_{43} & T_{44} & T_{45} & T_{46} \\
  T_{51} & T_{52} & T_{53} & T_{54} & T_{55} & T_{56} \\
  T_{61} & T_{62} & T_{63} & T_{64} & T_{65} & T_{66}
\end{bmatrix}
\]
Our choice of a task

A task is the shell quartets \((M : |P :\)\).

Define \(\Phi(M)\) to be the set of indices of shells that have nearby centers to shell \(M\), i.e., shell \(M\) and shells in \(\Phi(M)\) are “close”.

Set of significant shell quartets in \((M : |P :\) is contained in

\[
\{(MN|PQ) \text{ s.t. } N \in \Phi(M), Q \in \Phi(P)\}.
\]

Good upper bound on number of significant shell quartets is \(\eta(M)\eta(P)\), where \(\eta(M)\) is number of elements in \(\Phi(M)\).
Number of non-screened shell quartets in \((M, : | P, :)\)

System: \(C_{24}H_{50}\) with cc-pVDZ (294 shells, 586 basis functions)
“Uneven” grouping of indices

Group the indices $M$ into sets $G_i$ such that the sum of the $\eta(M)$ for each subset is about the same,

$$\sum_{M \in G_i} \eta(M) \approx \eta^*.$$  

Task $(M, : | P, :)$ is assigned to partition $(i, j)$ if $M \in G_i$ and $P \in G_j$.

For partition $(i, j)$, the estimated number of shell quartets is

$$\sum_{M \in G_i} \eta(M) \eta(P) = \sum_{M \in G_i} \eta(M) \times \sum_{P \in G_j} \eta(P) \approx (\eta^*)^2$$

which is approximately balanced.
What communication is required for a task?

For \((M : \mid P : )\), what are the submatrices of \(F\) and \(D\) that are needed?

Submatrices of \(F\) and \(D\) that are needed are contained in

\[ (M, \Phi(M)), \quad (P, \Phi(P)), \quad (\Phi(M), \Phi(P)) \]

i.e., \(F_{MN}, F_{PQ}, F_{NP}, F_{MQ}, F_{NP}, F_{MP}\) are all contained in one of the above sets

These submatrices of \(D\) are *prefetched* before the task is started. Local buffer stores submatrices of \(F\) during computation. Finally, submatrices of \(F\) are used to update the distributed/global \(F\).
Shell reordering – reorder for locality using a space filling curve

Elements of $D$ required

for $(100, : | 200, :)$
1 task

for $(100 : 109, : | 200 : 209, :)$
100 tasks

100 tasks require only $10 \times$ communication of single task

System: C$_2$H$_5$ with cc-pVDZ (294 shells, 586 functions)
## 64 node partitioning

<table>
<thead>
<tr>
<th>Protein with 617 basis functions</th>
<th>shell quartet balance</th>
<th>communication balance</th>
<th>ave comm/node (submatrices)</th>
<th>ave comm/node (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(M_i, N_j; :, :)$</td>
<td>8x8</td>
<td>3.81</td>
<td>40731.3</td>
<td>633.6</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>2.01</td>
<td>42789.4</td>
</tr>
<tr>
<td>$(M_i, N_j</td>
<td>P_k, :)$</td>
<td>4x4x4</td>
<td>7.71</td>
<td>18093.6</td>
</tr>
<tr>
<td>$(M_i, N_j</td>
<td>P_k, Q_l)$</td>
<td>4x4x2x2</td>
<td>11.38</td>
<td>16629.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alkane with 1930 basis functions</th>
<th>shell quartet balance</th>
<th>communication balance</th>
<th>ave comm/node (submatrices)</th>
<th>ave comm/node (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(M_i, N_j; :, :)$</td>
<td>8x8</td>
<td>6.37</td>
<td>82484.9</td>
<td>1283.1</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>1.07</td>
<td>84447.5</td>
</tr>
<tr>
<td>$(M_i, N_j</td>
<td>P_k, :)$</td>
<td>4x4x4</td>
<td>4.26</td>
<td>40707.0</td>
</tr>
<tr>
<td>$(M_i, N_j</td>
<td>P_k, Q_l)$</td>
<td>4x4x2x2</td>
<td>8.01</td>
<td>37533.6</td>
</tr>
</tbody>
</table>
### 64 node partitioning

#### Protein with 617 basis functions

<table>
<thead>
<tr>
<th>Submatrices</th>
<th>Shell Quartet Balance</th>
<th>Communication Balance</th>
<th>Ave Comm/Node (Submatrices)</th>
<th>Ave Comm/Node (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(M_i, \mathcal{N}_j : : :)$</td>
<td>8x8</td>
<td>3.81</td>
<td>1.36</td>
<td>40731.3</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>2.01</td>
<td>1.57</td>
</tr>
<tr>
<td>$(M_i, \mathcal{N}_j</td>
<td>P_k, :)$</td>
<td>4x4x4</td>
<td>7.71</td>
<td>4.71</td>
</tr>
<tr>
<td>$(M_i, \mathcal{N}_j</td>
<td>P_k, Q_l)$</td>
<td>4x4x2x2</td>
<td>11.38</td>
<td>4.98</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>1.15</td>
<td>1.37</td>
</tr>
</tbody>
</table>

#### Alkane with 1930 basis functions

<table>
<thead>
<tr>
<th>Submatrices</th>
<th>Shell Quartet Balance</th>
<th>Communication Balance</th>
<th>Ave Comm/Node (Submatrices)</th>
<th>Ave Comm/Node (kB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(M_i, \mathcal{N}_j : : :)$</td>
<td>8x8</td>
<td>6.37</td>
<td>3.09</td>
<td>82484.9</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>1.07</td>
<td>1.25</td>
</tr>
<tr>
<td>$(M_i, \mathcal{N}_j</td>
<td>P_k, :)$</td>
<td>4x4x4</td>
<td>4.26</td>
<td>4.79</td>
</tr>
<tr>
<td>$(M_i, \mathcal{N}_j</td>
<td>P_k, Q_l)$</td>
<td>4x4x2x2</td>
<td>8.01</td>
<td>7.32</td>
</tr>
<tr>
<td>$(M_i, :</td>
<td>P_k, :)$</td>
<td>8x8</td>
<td>1.03</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Work stealing scheduler

Initial static partitioning is not perfect
Work stealing is used to polish the load balance

► Each node has its own task queue
► Checking another node for work involves locking its queue
► We use a hierarchical scheme: $p_r$ process groups of $p_c$ nodes

Related work
1) Large-scale distributed work stealing: Dinan et al. 2009
2) Application to DFT: Nikodem et al. 2014
Hierarchical work stealing dynamic scheduling

1. On a node in process group $PG_k$ do
2. while any process group has work, i.e., any $W_i = 1$ do
3.   Select victim process group $PG_i$ with $W_i = 1$ closest to $PG_k$
4.   $C \leftarrow \emptyset$
5.   repeat
6.     Randomly select a node $n$ from $PG_i$
7.     if $n \notin C$ then
8.       while the task queue of $n$ has tasks do
9.         steal half of the tasks from $n$
10.    end
11.   $C \leftarrow C \cup n$
12. end
13. until $|C| = p_c$
14. $W_i \leftarrow 0$
15. end
Number of steals in distributed Fock matrix construction

<table>
<thead>
<tr>
<th>Nodes</th>
<th>1hsg_28</th>
<th>1hsg_38</th>
<th>1hsg_45</th>
<th>1hsg_90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>36</td>
<td>34</td>
<td>18</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>64</td>
<td>62</td>
<td>32</td>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>144</td>
<td>134</td>
<td>87</td>
<td>65</td>
<td>42</td>
</tr>
<tr>
<td>225</td>
<td>269</td>
<td>160</td>
<td>127</td>
<td>109</td>
</tr>
<tr>
<td>529</td>
<td>2444</td>
<td>380</td>
<td>366</td>
<td>349</td>
</tr>
<tr>
<td>1024</td>
<td>-</td>
<td>1738</td>
<td>1310</td>
<td>797</td>
</tr>
</tbody>
</table>

The test systems 1hsg_28, 1hsg_38, 1hsg_45, 1hsg_90 have 1159, 3555, 5065, 11163 basis functions, respectively.
### Fock matrix construction load balance ratio

<table>
<thead>
<tr>
<th>Nodes</th>
<th>static</th>
<th>hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>9</td>
<td>1.077</td>
<td>1.049</td>
</tr>
<tr>
<td>36</td>
<td>1.183</td>
<td>1.048</td>
</tr>
<tr>
<td>64</td>
<td>1.293</td>
<td>1.046</td>
</tr>
<tr>
<td>144</td>
<td>1.358</td>
<td>1.051</td>
</tr>
<tr>
<td>225</td>
<td>1.439</td>
<td>1.046</td>
</tr>
<tr>
<td>529</td>
<td>1.457</td>
<td>1.047</td>
</tr>
</tbody>
</table>

*1hsg_28 protein-ligand system (122 atoms and 1159 basis functions)*
Computation time and parallel overhead, $C_{144}H_{290}$

3466 basis functions
Density matrix purification

Alternative to diagonalization: \( D = C_{occ} C_{occ}^T \) developed for linear scaling \( O(N) \) methods.

If \( F = U \Lambda_F U^T \), then \( D = U \Lambda_D U^T \), where

\[
(\Lambda_D)_{ii} \begin{cases} 
1 & i \text{ is an occupied orbital} \\
0 & i \text{ is an unoccupied orbital}
\end{cases}
\]

Related to computing the matrix sign function.

McWeeny iteration: \( D_{k+1} = 3D_k^2 - 2D_k^3 \) starting with \( D_0 \) (appropriately scaled and shifted \( F \))

Based on matrix multiplication: more parallelism than eigendecomposition.
Canonical purification (Palser-Manolopoulos 1998) or trace-correcting purification (Niklasson 2002) to avoid needing chemical potential

1 Set $D_0$ appropriately, depending on number of occupied orbitals
2 for $k = 0, 1, \ldots$ until convergence do
3 $c_k = \frac{\text{trace}(D_k^2 - D_k^3)}{\text{trace}(D_k - D_k^2)}$
4 if $c_k \leq 1/2$ then
5 $D_{k+1} = \left( (1 - 2c_k)D_k + (1 + c_k)D_k^2 - D_k^3 \right) / (1 - c_k)$
6 else
7 $D_{k+1} = \left( (1 + c_k)D_k^2 - D_k^3 \right) / c_k$
8 end
9 end

Distributed 3D algorithm for matrix multiply (e.g., Dekel-Nassimi-Sahni 1981, Agarwal-Balle-Gustavson-Joshi-Palkar 1995)
Optimization of integral calculations

- Optimized the ERD library (Flocke-Lotrich 2008) from ACES III
- ERD uses Rys quadrature (compute Rys quadrature roots and weights, compute intermediate 2D integrals via recurrence relations, compute ERIs from the 2D integrals)

**Figure:** Breakup of runtime for the original ERD code.
Optimization of integral calculations

- loop restructuring, especially around branches
- code annotations to help compiler auto-vectorization
- hand optimization with intrinsics in several places
- other low-level optimizations: use of unsigned indices, changed interfaces to reduce register pressure
- avoid divisions and square roots in integral primitive screening
- use more efficient 5th degree Chebyshev polynomial approximation for Boys function
## ERI calculation performance improvement factor

<table>
<thead>
<tr>
<th>Molecule</th>
<th>Dual Ivy Bridge</th>
<th>Intel Xeon Phi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specific</td>
<td>Generic</td>
</tr>
<tr>
<td>alkane_1202</td>
<td>2.31</td>
<td>2.33</td>
</tr>
<tr>
<td>dna_19mer</td>
<td>2.26</td>
<td>2.32</td>
</tr>
<tr>
<td>graphene_936</td>
<td>2.11</td>
<td>2.24</td>
</tr>
<tr>
<td>1hsg_100</td>
<td>2.46</td>
<td>2.42</td>
</tr>
</tbody>
</table>
Heterogeneous Fock matrix construction

How to utilize accelerators on the compute nodes? (How to balance the work, and how to reduce PCIe communication)

- Use both CPUs and Intel Xeon Phi coprocessors for Fock matrix construction
- Must also form the partial Fock matrix sums on the accelerator to avoid communicating integrals across PCIe bus
- Use work-stealing dynamic scheduling between CPUs and coprocessors
- Dedicated CPU core is used for scheduling
- Use a combination of atomic operations and local copies when threads sum into the Fock matrix
Timings for different problem sizes

(a) 1159 basis functions

(b) 3555 basis functions

(c) 5065 basis functions

(d) 11163 basis functions
Strong scalability

(a) 1159 basis functions
(b) 3555 basis functions
(c) 5065 basis functions
(d) 11163 basis functions
Scaling with number of basis functions on 1 and 529 nodes

(a) 1 node

(b) 529 nodes
Tianhe-2 system

- 16000 nodes; we used up to 8100 nodes
- Each node: two Intel Ivy Bridge E5-2692 processors (12 cores each at 2.2 GHz)
- Each node: three Intel Xeon Phi coprocessors (57 core; 31S1P)
- Memory on each node is 64 GB DRAM and 8 GB on each Intel Xeon Phi card.
- TH Express-2 interconnect
Some 1HSG test models (cc-pVDZ)

<table>
<thead>
<tr>
<th>Model</th>
<th>Atoms</th>
<th>Shells</th>
<th>Basis Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1hsg_80</td>
<td>1035</td>
<td>4576</td>
<td>9584</td>
</tr>
<tr>
<td>1hsg_140</td>
<td>2145</td>
<td>9497</td>
<td>19903</td>
</tr>
<tr>
<td>1hsg_180</td>
<td>2938</td>
<td>13054</td>
<td>27394</td>
</tr>
</tbody>
</table>

E. Chow: Georgia Tech
## Protein-ligand binding energies for 1HSG

<table>
<thead>
<tr>
<th>$R$ (Å)</th>
<th>Protein-Ligand Energy (hartree)</th>
<th>Protein Energy (hartree)</th>
<th>HF Binding Energy (kcal/mol)</th>
<th>HF-D Binding Energy (kcal/mol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>-2907.05690</td>
<td>-944.67312</td>
<td>31.0</td>
<td>41.7</td>
</tr>
<tr>
<td>3.5</td>
<td>-5251.23018</td>
<td>-3288.81152</td>
<td>52.8</td>
<td>92.7</td>
</tr>
<tr>
<td>4.0</td>
<td>-10693.96902</td>
<td>-8731.57905</td>
<td>34.8</td>
<td>141.9</td>
</tr>
<tr>
<td>4.5</td>
<td>-12286.88136</td>
<td>-10324.48287</td>
<td>40.2</td>
<td>156.3</td>
</tr>
<tr>
<td>5.0</td>
<td>-13486.15403</td>
<td>-11523.75545</td>
<td>40.2</td>
<td>158.1</td>
</tr>
<tr>
<td>5.5</td>
<td>-14204.57382</td>
<td>-12242.17624</td>
<td>39.6</td>
<td>158.1</td>
</tr>
<tr>
<td>6.0</td>
<td>-16008.70680</td>
<td>-14046.31026</td>
<td>39.0</td>
<td>158.5</td>
</tr>
<tr>
<td>6.5</td>
<td>-16956.98340</td>
<td>-14994.58875</td>
<td>37.8</td>
<td>157.8</td>
</tr>
<tr>
<td>7.0</td>
<td>-17784.32979</td>
<td>-15821.93654</td>
<td>36.9</td>
<td>157.5</td>
</tr>
<tr>
<td>8.0</td>
<td>-24177.59972</td>
<td>-22215.20726</td>
<td>36.4</td>
<td>158.4</td>
</tr>
<tr>
<td>9.0</td>
<td>-28027.38358</td>
<td>-26065.00035</td>
<td>30.6</td>
<td>153.1</td>
</tr>
<tr>
<td>10.0</td>
<td>-33055.02296</td>
<td>-31092.64187</td>
<td>29.3</td>
<td>152.1</td>
</tr>
<tr>
<td>12.0</td>
<td>-39807.13278</td>
<td>-37844.74900</td>
<td>31.0</td>
<td>154.1</td>
</tr>
<tr>
<td>14.0</td>
<td>-50116.62443</td>
<td>-48154.24041</td>
<td>31.1</td>
<td>154.3</td>
</tr>
<tr>
<td>16.0</td>
<td>-61609.08891</td>
<td>-59646.70206</td>
<td>32.9</td>
<td>156.2</td>
</tr>
<tr>
<td>18.0</td>
<td>-70047.24752</td>
<td>-68084.85578</td>
<td>35.9</td>
<td>159.3</td>
</tr>
</tbody>
</table>
SCF timings and speedup for 1hsg_80 problem on Stampede

![Graph showing SCF timings and speedup](image)

- **SCF Timings:**
  - `Time (s)` is plotted against `Number of nodes` for `Total`, `Purif`, and `Total w/accel`.
  - The x-axis represents the number of nodes ranging from 64 to 1024.
  - The y-axis represents time in seconds ranging from $10^{-4}$ to $10^{4}$.

- **Speedup:**
  - **Strong speedup (relative to 64 nodes):**
    - `Fock` and `Total` are plotted against `Number of nodes` ranging from 64 to 1024.
    - The y-axis represents strong speedup relative to 64 nodes.

E. Chow: Georgia Tech
SCF timings and speedup for 1hsg_180 problem on Tianhe-2

![Graph showing SCF timings and speedup](image)

- **27394 basis functions**
- **Tianhe-2: 24 CPU cores per node**
- **Relative parallel efficiency at 8100 nodes: 73.5%**
Conclusions

- Reduced communication and improved load balance in Fock matrix construction
- Density matrix purification is recommended for highly parallel computations
- Integral calculations: research and development needed for more efficient SIMD computation
- Developed a heterogeneous scheduler for CPU-MIC computations
- Large-scale runs on 8100 nodes of Tianhe-2 for 2938 atoms (27394 basis functions)
- Released open-source GTFock software which is being integrated into PSI4 and NWChem quantum chemistry packages
Collaborators

Georgia Tech
Xing Liu
Marat Dukhan
Aftab Patel
David Sherrill
Trent Parker

Intel
Sanchit Misra
Mikhail Smelyanskiy
Jeff R. Hammond
Pradeep Dubey

Acknowledgments

Intel Parallel Computing Center grant
National University of Defense Technology, China
National Supercomputing Center in Guangzhou, China
NSF SI2-SSI grant ACI-1147843
NSF XSEDE grant TG-CCR140016