GPU-Accelerated Real Space Electronic Structure Theory on HPC Resources

William P. Huhn1, Björn Lange1, Victor Yu1, Seyong Lee2, Mina Yoon3, Volker Blum1
1. Department of Mechanical Engineering and Materials Science, Duke University
2. Computer Science and Math Division, Oak Ridge National Laboratory
3. Center for Nanophase Materials Sciences, Oak Ridge National Laboratory

MolSSI Workshop/ELSI Conference 2018
August 17, 2018, 2:40 - 3:10 EST
## Introduction

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>[TFlop/s]</th>
<th>[TFlop/s]</th>
<th>[kW]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Summit</strong> - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States</td>
<td>2,282,544</td>
<td>122,300.0</td>
<td>187,659.3</td>
<td>8,806</td>
</tr>
<tr>
<td>2</td>
<td><strong>Sunway TaihuLight</strong> - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRPC National Supercomputing Center in Wuxi China</td>
<td>10,649,600</td>
<td>93,014.6</td>
<td>125,435.9</td>
<td>15,371</td>
</tr>
<tr>
<td>3</td>
<td><strong>Sierra</strong> - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/NNSA/LLNL United States</td>
<td>1,572,480</td>
<td>71,610.0</td>
<td>119,193.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><strong>Tianhe-2A</strong> - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China</td>
<td>4,981,760</td>
<td>61,444.5</td>
<td>100,678.7</td>
<td>18,482</td>
</tr>
<tr>
<td>5</td>
<td><strong>Al Bridging Cloud Infrastructure (ABCI)</strong> - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu National Institute of Advanced Industrial Science and Technology (AIST) Japan</td>
<td>391,680</td>
<td>19,880.0</td>
<td>32,576.6</td>
<td>1,649</td>
</tr>
<tr>
<td>6</td>
<td><strong>Piz Daint</strong> - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre [CSCS] Switzerland</td>
<td>361,760</td>
<td>19,590.0</td>
<td>25,326.3</td>
<td>2,272</td>
</tr>
</tbody>
</table>

[https://www.top500.org/list/2018/06/](https://www.top500.org/list/2018/06/), accessed 25 June 2018
## Introduction

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>TFlop/s</th>
<th>TFlop/s</th>
<th>kW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM</td>
<td>2,282,544</td>
<td>122,300.0</td>
<td>187,659.3</td>
<td>8,806</td>
</tr>
<tr>
<td>2</td>
<td>Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRPC</td>
<td>10,649,600</td>
<td>93,014.6</td>
<td>125,435.9</td>
<td>15,371</td>
</tr>
<tr>
<td>3</td>
<td>Sierra - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM</td>
<td>1,572,480</td>
<td>71,610.0</td>
<td>119,193.6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT</td>
<td>4,981,760</td>
<td>61,444.5</td>
<td>100,678.7</td>
<td>18,482</td>
</tr>
<tr>
<td>5</td>
<td>AI Bridging Cloud Infrastructure (ABC) - PRIMERGY CX2550 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, EDR, Fujitsu</td>
<td>391,680</td>
<td>19,880.0</td>
<td>32,576.6</td>
<td>1,649</td>
</tr>
<tr>
<td>6</td>
<td>Piz Daint - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect, NVIDIA Tesla P100, Cray Inc.</td>
<td>361,760</td>
<td>19,590.0</td>
<td>25,326.3</td>
<td>2,272</td>
</tr>
</tbody>
</table>

[https://www.top500.org/list/2018/06/](https://www.top500.org/list/2018/06/), accessed 25 June 2018
Introduction

Supporting general-purpose GPU (GPGPU) acceleration is critical for running electronic structure theory on current leading HPC platforms!
Supporting general-purpose GPU (GPGPU) acceleration is critical for running electronic structure theory on current leading HPC platforms!

Code targeted: FHI-aims

- All-Electron, Full-Potential KS-DFT
- Localized basis sets: Numeric atom-centered orbitals (NAOs)
- Developed by an active, globally-distributed academic community of 100+ developers
Introduction

Supporting general-purpose GPU (GPGPU) acceleration is critical for running electronic structure theory on current leading HPC platforms!

Code targeted: FHI-aims
- All-Electron, Full-Potential KS-DFT
- Localized basis sets: Numeric atom-centered orbitals (NAOs)
- Developed by an active, globally-distributed academic community of 100+ developers

Δ-Project Places FHI-aims on a Similar Accuracy Level to LAPW

<table>
<thead>
<tr>
<th>Code</th>
<th>Version</th>
<th>Basis</th>
<th>Δ-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIEN2k</td>
<td>13.1</td>
<td>LAPW/APW+lo</td>
<td>0 meV/atom</td>
</tr>
<tr>
<td>FHI-aims</td>
<td>081213</td>
<td>tier2 numerical orbitals</td>
<td>0.2 meV/atom</td>
</tr>
</tbody>
</table>

https://molmod.ugent.be/deltacodesdft
What is Needed for GPGPU-Accelerated KS-DFT?

For (non-hybrid) KS-DFT using real-space operations, the rate-limiting steps are:

Integration of the Hamiltonian Matrix Elements

\[ H_{mn} = \int \varphi_m^*(r) \hat{h} \varphi_n(r) dr \]

Electron Density Calculation

\[ \rho(r) = \sum_{mn} \varphi_m^*(r) D_{mn} \varphi_n(r) \]

Solution (or circumvention) of the KS-DFT Equation

\[ \left[ \hat{t} + \hat{v}_{ES} + \hat{v}_{XC} \right] \psi_m(r) = \epsilon \psi_m(r) \]
What is Needed for GPGPU-Accelerated KS-DFT?

For (non-hybrid) KS-DFT using real-space operations, the rate-limiting steps are:

Integration of the Hamiltonian Matrix Elements

\[ H_{mn} = \int \varphi_m^*(r) \hat{h} \varphi_n(r) \, dr \]

Electron Density Calculation

\[ \rho(r) = \sum_{mn} \varphi_m^*(r) D_{mn} \varphi_n(r) \]

Solution (or circumvention) of the KS-DFT Equation

\[
\left[ \hat{t} + \hat{v}_{ES} + \hat{v}_{XC} \right] \psi_m(r) = \epsilon \psi_m(r)
\]
What is Needed for GPGPU-Accelerated KS-DFT?

For (non-hybrid) KS-DFT using real-space operations, the rate-limiting steps are:

Integration of the Hamiltonian Matrix Elements

\[ H_{mn} = \int \varphi^*_m(r) \hat{h} \varphi_n(r) \, dr \]

Electron Density Calculation

\[ \rho(r) = \sum_{mn} \varphi^*_m(r) D_{mn} \varphi_n(r) \]

Solution (or circumvention) of the KS-DFT Equation

\[ \left[ \hat{t} + \hat{v}_{ES} + \hat{v}_{XC} \right] \psi_m(r) = \epsilon \psi_m(r) \]

Small/Mid-Scale Calculations

\[ O(N_{\text{atom}}) \]

Batch Integration+

GPGPUs

Large-Scale Calculations

ELSI

elsi-interchange.org
Numeric Atom-Centered Orbitals

- **Integration points** distributed on radial grids around atoms
- **Basis elements** localized in space give O(N) real-space operation
- Capture core region “wiggles”; naturally all-electron
- Pre-constructed “tiers” of basis elements suitable for DFT:

<table>
<thead>
<tr>
<th>Basis Set</th>
<th>H</th>
<th>C</th>
<th>Si</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>1s</td>
<td>1s2s2p</td>
<td>1s2s2p3s3p</td>
</tr>
<tr>
<td>Tier 1</td>
<td>+sp</td>
<td>+pds</td>
<td>+dpfs</td>
</tr>
<tr>
<td>Tier 2</td>
<td>+spsd</td>
<td>+fpsgd</td>
<td>+dgps</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Blum et al., Comp. Phys. Comm. 2009
Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points \textit{(batches)} via cut planes iteratively

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (**batches**) via cut planes iteratively.

As system size grows, number of batches grows linearly.

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

As system size grows, number of batches grows linearly

Batches are embarrassingly parallel over tasks

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively

As system size grows, number of batches grows linearly

Batches are embarrassingly parallel over tasks

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively.

As system size grows, number of batches grows linearly.

Batches are embarrassingly parallel over tasks.

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively.

As system size grows, number of batches grows linearly.

Batches are embarrassingly parallel over tasks.

Only basis elements touching a batch contribute to work done for batch.

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively.

As system size grows, number of batches grows linearly.

Batches are embarrassingly parallel over tasks.

Only basis elements touching a batch contribute to work done for batch.

Grid Adapted Cut-Plane Partitioning

Divide integration grid into compact sets of points (batches) via cut planes iteratively.

As system size grows, number of batches grows linearly.

Batches are embarrassingly parallel over tasks.

Only basis elements touching a batch contribute to work done for batch.

Divide-And-Conquer on the Real-Space Matrix

Each batch only contributes to a small number of real-space matrix elements…

Each batch only contributes to a small number of real-space matrix elements...
Divide-And-Conquer on the Real-Space Matrix

Each batch only contributes to a small number of real-space matrix elements...

... so for each batch, we only calculate a reduced $H^{\text{batch}}$ on all basis elements touching that batch.

Each batch only contributes to a small number of real-space matrix elements…

… so for each batch, we only calculate a reduced $H^{\text{batch}}$ on all basis elements touching that batch.

Since basis elements are localized, there is an maximum limit on the size of $H^{\text{batch}}$!

---


---

**Divide-And-Conquer on the Real-Space Matrix**

<table>
<thead>
<tr>
<th>Basis Functions Touching Batch</th>
<th>Basis Functions Touching Batch</th>
<th>$H^{\text{batch, max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basis Functions Touching Batch</td>
<td>Basis Functions Touching Batch</td>
<td>Basis Functions Touching Batch</td>
</tr>
</tbody>
</table>
Each batch only contributes to a small number of real-space matrix elements…

... so for each batch, we only calculate a reduced $H_{\text{batch}}$ on all basis elements touching that batch.

Since basis elements are localized, there is a maximum limit on the size of $H_{\text{batch}}$!

Fixed max work done per batch

* $O(N_{\text{atom}})$ number of batches = $O(N_{\text{atom}})$ overall work

Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \phi_m^*(r) \hat{h} \phi_n(r) dr \]

CPU-Only

do i_batch = 1, number_of_batches
  calculate_w( i_batch )
  calculate_psi( i_batch )
  calculate_H_times_psi( i_batch )

  psi(:) = w . psi(:)
  DSYR2K( H_times_psi, psi, H_batch )

  index_H( H_batch, hamiltonian )
end do
Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \varphi_m^*(r) \hat{h} \varphi_n(r) \, dr \]

**CPU-Only**

```plaintext
do i_batch = 1, number_of_batches
  calculate_w( i_batch )
calculate_psi( i_batch )
calculate_H_times_psi( i_batch )

psi(:) = w . psi(:)
DSYR2K( H_times_psi, psi, H_batch )
index_H( H_batch, hamiltonian )
end do
```

Setting up batch quantities (tedious but cheap on CPU)
Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \varphi_m^*(\mathbf{r}) \hat{h} \varphi_n(\mathbf{r}) d\mathbf{r} \]

CPU-Only

```
do i_batch = 1, number_of_batches
   calculate_w( i_batch )
   calculate_psi( i_batch )
   calculate_H_times_psi( i_batch )
   psi(:) = w * psi(:)
   DSYR2K( H_times_psi, psi, H_batch )
   index_H( H_batch, hamiltonian )
end do
```

Setting up batch quantities (tedious but cheap on CPU)

BLAS3 calls and vectorized ops (computationally expensive part)
Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \varphi_m^*(r) \hat{h} \varphi_n(r) \, dr \]

**CPU-Only**

```plaintext
    do i_batch = 1, number_of_batches
        calculate_w( i_batch )
        calculate_psi( i_batch )
        calculate_H_times_psi( i_batch )

        psi(:) = w * psi(:)
        DSYR2K( H_times_psi, psi, H_batch )

        index_H( H_batch, hamiltonian )
    end do
```

- **Setting up batch quantities** (tedious but cheap on CPU)
- **BLAS3 calls and vectorized ops** (computationally expensive part)
- **Indexing from batch matrix back to accumulated matrix**
Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \varphi^*_m(r) \hat{h} \varphi_n(r) \, dr \]

**CPU-Only**

```plaintext
do i_batch = 1, number_of_batches
  calculate_w( i_batch )
calculate_psi( i_batch )
calculate_H_times_psi( i_batch )

psi(:) = w . psi(:)
DSYR2K( H_times_psi, psi, H_batch )

index_H( H_batch, hamiltonian )
end do
```

**GPGPU-Accelerated**

```plaintext
do i_batch = 1, number_of_batches
  calculate_w( i_batch )
calculate_psi( i_batch )
calculate_H_times_psi( i_batch )

  copy_cpu_data_to_gpu( w, psi, H_times_psi )
cublasDDGMM( psi, w, psi )
cublasDSYR2K( H_times_psi, psi, H_batch )

  index_H_on_gpu( H_batch, hamiltonian )
end do

copy_gpu_data_to_cpu( hamiltonian )
```
**Example Real-Space Operation: Hamiltonian Integration**

**Hamiltonian Matrix Calculation**

\[ H_{mn} = \int \varphi_m^*(\mathbf{r}) \hat{h} \varphi_n(\mathbf{r}) \, d\mathbf{r} \]

---

**CPU-Only**

```plaintext
do i_batch = 1, number_of_batches
    calculate_w( i_batch )
    calculate_psi( i_batch )
    calculate_H_times_psi( i_batch )
    psi(:) = w . psi(:)
    DSYR2K( H_times_psi, psi, H_batch )
    index_H( H_batch, hamiltonian )
end do
```

**GPGPU-Accelerated**

```plaintext
do i_batch = 1, number_of_batches
    calculate_w( i_batch )
    calculate_psi( i_batch )
    calculate_H_times_psi( i_batch )
    copy_cpu_data_to_gpu( w, psi, H_times_psi )
    cublasDDGMM( psi, w, psi )
    cublasDSYR2K( H_times_psi, psi, H_batch )
    index_H_on_gpu( H_batch, hamiltonian )
end do
```

**BLAS3 calls become cuBLAS calls**

**Compute-time-intensive portion**
Example Real-Space Operation: Hamiltonian Integration

Hamiltonian Matrix Calculation

\[ H_{mn} = \int \varphi_m^*(\mathbf{r}) \hat{h} \varphi_n(\mathbf{r}) d\mathbf{r} \]

**CPU-Only**

```java
do i_batch = 1, number_of_batches
   calculate_w( i_batch )
   calculate_psi( i_batch )
   calculate_H_times_psi( i_batch )

   psi(:) = w * psi(:)
   DSYR2K( H_times_psi, psi, H_batch )

   index_H( H_batch, hamiltonian )
end do
```

- BLAS3 calls become cuBLAS calls
- Compute-time-intensive portion

**GPGPU-Accelerated**

```java
do i_batch = 1, number_of_batches
   calculate_w( i_batch )
   calculate_psi( i_batch )
   calculate_H_times_psi( i_batch )

   copy_cpu_data_to_gpu( w, psi, H_times_psi )
   cublasDDGMM( psi, w, psi )
   cublasDSYR2K( H_times_psi, psi, H_batch )

   index_H_on_gpu( H_batch, hamiltonian )
end do
```

- GPGPU handles indexing of matrix…
- allowing CPU to prepare next batch
- GPGPU-to-CPU transfer occurs once
- Programmer-time-intensive portion
## Systems Used for Benchmarking

Two clusters used:
- **timewarp**: Development cluster of aims group at Duke University
- **PSG**: Benchmarking cluster for Tesla GPGPUs at NVIDIA

Three types of nodes used:

<table>
<thead>
<tr>
<th>Node</th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>timewarp</td>
<td>PSG</td>
<td>PSG</td>
</tr>
<tr>
<td>CPU</td>
<td>2x Intel Xeon E5-2670v2 (20 cores*, Ivy Bridge)</td>
<td>2x Intel Xeon E5-2698v3 (32 cores, Haswell)</td>
<td>2x Intel Xeon Gold 6148 (20 cores, Skylake)</td>
</tr>
<tr>
<td>GPGPU**</td>
<td>1x Quadro GP100 (Pascal)</td>
<td>4x Tesla P100 (Pascal)</td>
<td>4x Tesla V100 (Volta)</td>
</tr>
<tr>
<td>MPI Tasks/GPGPUs</td>
<td>16/1</td>
<td>32/4</td>
<td>20/4</td>
</tr>
<tr>
<td>Compilers/Libraries</td>
<td>ifort 14.0, MKL 11.1.1, IMPI 4.1.3, CUDA 8.0</td>
<td>ifort 17.0, MKL 11.3.3, IMPI 5.0.3, CUDA 9.1</td>
<td>ifort 17.0, MKL 11.3.3, IMPI 5.0.3, CUDA 9.1</td>
</tr>
</tbody>
</table>

* Due to usage of MPS, only 16 MPI tasks used for Ivy Bridge/GP100

** All GPGPUs are PCI-E models

---

MoISSI/ELSI 2018  
August 17, 2018, 2:40 pm EST  
www.williamphuhn.com
A Complete All-Electron Calculation: Si (3x3x3)

We perform a complete all-electron calculation including forces and analytical stress tensor for a 3x3x3 supercell of Si.
A Complete All-Electron Calculation: Si (3x3x3)

We perform a complete all-electron calculation including forces and analytical stress tensor for a 3x3x3 supercell of Si.

This calculation is representative of a single iteration of a geometric optimization calculation, containing:

- Initialization (not GPGPU accelerated)
- 12 “Normal” SCF Iterations
- 1 SCF + Forces + Analytical Stress Tensor Iteration
A Complete All-Electron Calculation: Si (3x3x3)

We perform a complete all-electron calculation including forces and analytical stress tensor for a 3x3x3 supercell of Si

This calculation is representative of a single iteration of a geometric optimization calculation, containing
• Initialization (not GPGPU accelerated)
• 12 “Normal” SCF Iterations
• 1 SCF + Forces + Analytical Stress Tensor Iteration

Computational Details:
• PBE functional
• Tight integration settings
• Tight basis sets
• SCF k-grid: 1x1x1
• Load balancing for GPGPU, CSR for CPU
Total Time for **Entire Calculation**: Si (3x3x3)

![Graph showing speed-up in total time for different processor-GPU combinations.](image-url)

- **Ivy Bridge/GP100**
  - 16 Cores Ivy Bridge
  - 1 Pascal GPGPU
  - Speed-up: 3.8x

- **Haswell/P100**
  - 32 Cores Haswell
  - 4 Pascal GPGPU
  - Speed-up: 3.7x

- **Skylake/V100**
  - 20 Cores Skylake
  - 4 Volta GPGPU
  - Speed-up: 4.0x
Total Time for **Entire Calculation**: Si (3x3x3)

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPGPU</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ivy Bridge/GP100</td>
<td>16 Cores Ivy Bridge, 1 Pascal GPGPU</td>
<td>3.8x</td>
</tr>
<tr>
<td>Haswell/P100</td>
<td>32 Cores Haswell, 4 Pascal GPGPU</td>
<td>3.7x</td>
</tr>
<tr>
<td>Skylake/V100</td>
<td>20 Cores Skylake, 4 Volta GPGPU</td>
<td>4.0x</td>
</tr>
</tbody>
</table>

Do these results extend to other functionals?
Total Time for **Entire Calculation**: Si (3x3x3)

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPGPU</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ivy Bridge/GP100</td>
<td>16 Cores Ivy Bridge</td>
<td>1 Pascal GPGPU</td>
</tr>
<tr>
<td>Haswell/P100</td>
<td>32 Cores Haswell</td>
<td>4 Pascal GPGPU</td>
</tr>
<tr>
<td>Skylake/V100</td>
<td>20 Cores Skylake</td>
<td>4 Volta GPGPU</td>
</tr>
</tbody>
</table>
Total Time for **Entire Calculation**: Si (3x3x3)

Total Time to Solution (s)

<table>
<thead>
<tr>
<th>System</th>
<th>CPU Speed-up</th>
<th>GPGPU Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ivy Bridge/GP100</td>
<td>2.9x</td>
<td></td>
</tr>
<tr>
<td>Haswell/P100</td>
<td>3.2x</td>
<td></td>
</tr>
<tr>
<td>Skylake/V100</td>
<td>3.5x</td>
<td></td>
</tr>
</tbody>
</table>

Yes, but less speedup for LDA due to less linear algebra
### Speedups for Diamond Si (3x3x3)

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.9x</td>
<td>2.5x</td>
<td>3.1x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>4.4x</td>
<td>5.8x</td>
<td>5.7x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>5.2x</td>
<td>6.7x</td>
<td>7.8x</td>
</tr>
<tr>
<td>Entire Calculation</td>
<td>3.8x</td>
<td>3.7x</td>
<td>4.0x</td>
</tr>
</tbody>
</table>
Back to PBE…

Speedups for Diamond Si (3x3x3)

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.9x</td>
<td>2.5x</td>
<td>3.1x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>4.4x</td>
<td>5.8x</td>
<td>5.7x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>5.2x</td>
<td>6.7x</td>
<td>7.8x</td>
</tr>
<tr>
<td>Entire Calculation</td>
<td>3.8x</td>
<td>3.7x</td>
<td>4.0x</td>
</tr>
</tbody>
</table>

Do these results generalize to other materials?
To systematically verify the performance of the GPGPU code, we use the 103 compound material test set from Huhn and Blum:

3x3x3 supercells of the primitive cell for a given material was used, yielding unit cell sizes ranging from 27, 54, and 108 atoms.

Computational Details:
- PBE functional
- Tight integration settings
- Tight basis sets
- SCF k-grid: 1x1x1
- Load balancing was used (critical!)

<table>
<thead>
<tr>
<th>Family</th>
<th>No. materials</th>
<th>Materials</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elementals</td>
<td>45</td>
<td>Be, C [GRA], Ne, Mg, Al, Si, Ca, Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Ge, Sr, Y, Zr, Nb, Mo, Te, Ru, Rh, Pd, Ag, Cd, Sn, Xe, Ba, Lu, Hf, Ta, W, Re, Os, Ir, Pt, Au, Ti, Pb, Bi, Po</td>
</tr>
<tr>
<td>Compound semiconductors</td>
<td>37</td>
<td>C [DIA], MgO, AlN [WUR], AlN [ZB], SiC, BP, AIP, MgS, ZnO, ZnS [WUR], ZnS [ZB], GaN [WUR], GaN [ZB], GaP, AlAs, BAs, GaAs, MgSe, ZnSe, CdS [WUR], CdS [ZB], CdSe [WUR], CdSe [ZB], InN, InP, InAs, AlSb, GaSb, InSb, ZnTe, CdTe, HgS, HgSe, HgTe, PbS, PbSe, PbTe</td>
</tr>
<tr>
<td>Alkali halides</td>
<td>21</td>
<td>LiF, NaF, LiCl, NaCl, KF, KCl, LiBr, NaBr, KBr, RbF, RbCl, RbBr, LiI, NaI, KI, RbI, CsF, CsCl [CSCL], CsCl [RS], CsBr, CsI</td>
</tr>
</tbody>
</table>
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

= \times
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

# basis fns × # basis fns = # basis fns × # points × # points × # basis fns
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

\[
\text{DGEMM} \approx (\text{# basis fn})^2 \times (\text{# points})
\]
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

\[ \# \text{ basis fns} \times \# \text{ basis fns} = \# \text{ basis fns} \times \# \text{ points} \times \# \text{ points} \times \# \text{ basis fns} \]

DGEMM \( \approx (\# \text{ basis fn})^2 \times (\# \text{ points}) \)

If we did one DGEMM per batch…
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

\[
\text{# basis fns} \times \text{# basis fns} \times \text{# basis fns} = \text{# basis fns} \times \text{# points} \times \text{# points} \times \text{# basis fns}
\]

\[\text{DGEMM} \approx (\text{# basis fn})^2 \times (\text{# points})\]

If we did one DGEMM per batch…

\[\text{# ops} \approx \text{# batches} \times <\text{# basis fns}>^2 \times <\text{# points}>\]

where \(<\ldots>\) denotes average over batches
Quantities Used in 103 Compound Benchmark

(Crude Estimate!) of Problem Size

\[
\text{# basis fns} \times \text{# basis fns} = \text{# basis fns} \times \text{# points} \times \text{# points}
\]

\[
\text{DGEMM} \approx (\text{# basis fn})^2 \times (\text{# points})
\]

If we did one DGEMM per batch...

\[
\# \text{ ops} \approx \# \text{ batches} \times <\text{# basis fns}^2 \times \text{# points}>
\]

where \(<\ldots\>) denotes average over batches

Speedup

To compute speedups, we use timings for all calculations:

\[
T_{CPU} = \sum_{\text{materials}} t_{CPU}
\]

\[
T_{GPGPU} = \sum_{\text{materials}} t_{GPGPU}
\]

\[
\text{Speedup} = \frac{T_{CPU}}{T_{GPGPU}}
\]

This is commonly reported as \(\#x\), where 1.0x corresponds to no speedup
SCF Iteration for 103 Compounds: Skylake/V100

**Hamiltonian Integration**
- 3.6x

**Density**
- 3.2x

**Hartree Multipole Summation**
- N/A

**Total Time for Iteration**
- 2.4x

**Speed-up**
- CPU
- GPGPU
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

- Time (s)
- # 10^13 ops
- Density

3.6x

Hartree Multipole Summation

- Time (s)
- # 10^13 ops
- Total Time for Iteration

N/A

Each data point corresponds to one of the 103 materials

3.2x

CPU

GPGPU

2.4x

Speed-up
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.6x

Hartree Multipole Summation

N/A

Density

3.2x

Total Time for Iteration

2.4x

AIN

CPU
GPGPU
Speed-up

**MolSSI/ELSI 2018**
August 17, 2018, 2:40 pm EST

www.williamphuhn.com
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.6x

Dense matters!

Hartree Multipole Summation

N/A

All-electron does not imply lower atomic number = faster!

Local basis set matters!

Total Time for Iteration

3.2x

3.4x

CPU

GPGPU

Speed-up

AllN

AllN

www.williamphuhn.com
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

Density

3.6x

Hartree Multipole Summation

N/A

Total Time for Iteration

2.4x

CPU

GPGPU

Speed-up
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration
- Time (s) vs. # 10^13 ops
- Speed-up: 3.6x

Hartree Multipole Summation
- Time (s) vs. # 10^13 ops
- Speed-up: N/A

Density
- Time (s) vs. # 10^13 ops
- Speed-up: 3.2x

Total Time for Iteration
- Time (s) vs. # 10^13 ops
- Speed-up: 2.4x

Hamiltonian, Density, Total Time agree with DGEMM estimate

CPU

GPGPU
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.6x

Hartree Multipole Summation

N/A

Not GPGPU accelerated

Density

3.2x

Total Time for Iteration

2.4x

Hamiltonian, Density, Total Time agree with DGEMM estimate

CPU

GPGPU

Speed-up

Not GPGPU accelerated
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration
- Time (s)
- # 10^13 ops

Hartree Multipole Summation
- Time (s)
- # 10^13 ops

Density
- 3.6x

Total Time for Iteration
- Time (s)
- # 10^13 ops

- 3.2x
- 2.4x

CPU GPGPU Speed-up

3.6x
N/A
3.2x
2.4x
SCF Iteration for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.6x

Hartree Multipole Summation

N/A

Total Time for Iteration

2.4x

What if we calculate forces?

CPU

GPGPU

Speed-up

3.2x

0.0 0.2 0.4 0.6 0.8 1.0
# 10^13 ops

0 5 10 15 20 25 30 35 40
Time (s)

0.0 0.2 0.4 0.6 0.8 1.0
# 10^13 ops

0 5 10 15 20 25 30 35 40
Time (s)

0.0 0.2 0.4 0.6 0.8 1.0
# 10^13 ops

0 5 10 15 20 25 30 35 40
Time (s)

0.0 0.2 0.4 0.6 0.8 1.0
# 10^13 ops

0 5 10 15 20 25 30 35 40
Time (s)
SCF Iteration + Forces for 103 Compounds: Skylake/V100

Hamiltonian Integration

Density + Forces

Hartree Multipole Summation

Total Time for Iteration

3.8x

6.7x

N/A

4.6x

CPU

GPGPU

Speed-up

www.williamphuhn.com
SCF Iteration + Forces for 103 Compounds: Skylake/V100

**Hamiltonian Integration**
- 3.8x

**Density + Forces**
- 6.7x

**Hartree Multipole Summation**
- N/A

**Total Time for Iteration**
- 4.6x

More dense serial linear algebra, better speedups.

CPU GPGPU Speed-up
SCF Iteration + Forces for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.8x

Hartree Multipole Summation

N/A

Density + Forces

6.7x

Total Time for Iteration

4.6x

Now let’s add analytical stress…

CGR

GPGPU

Speed-up

www.williamphuhn.com
SCF Iteration + Forces + Stress for 103 Compounds: Skylake/V100

Hamiltonian Integration

3.7x

Hartree Multipole Summation

N/A

Density + Forces + Stress

9.0x

Total Time for Iteration

6.6x

CPU

GPGPU

Speed-up
## TL; DR

### Speedups for Diamond Si (3x3x3)

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.9x</td>
<td>2.5x</td>
<td>3.1x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>4.4x</td>
<td>5.8x</td>
<td>5.7x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>5.2x</td>
<td>6.7x</td>
<td>7.8x</td>
</tr>
<tr>
<td>Entire Calculation</td>
<td>3.8x</td>
<td>3.7x</td>
<td>4.0x</td>
</tr>
</tbody>
</table>

### Speedups for 103 Compound Test Set

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.4x</td>
<td>2.1x</td>
<td>2.4x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>3.9x</td>
<td>4.5x</td>
<td>4.6x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>4.5x</td>
<td>6.6x</td>
<td>6.6x</td>
</tr>
</tbody>
</table>
## TL; DR

### Speedups for Diamond Si (3x3x3)

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.9x</td>
<td>2.5x</td>
<td>3.1x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>4.4x</td>
<td>5.8x</td>
<td>5.7x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>5.2x</td>
<td>6.7x</td>
<td>7.8x</td>
</tr>
<tr>
<td>Entire Calculation</td>
<td>3.8x</td>
<td>3.7x</td>
<td>4.0x</td>
</tr>
</tbody>
</table>

### Speedups for 103 Compound Test Set

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.4x</td>
<td>2.1x</td>
<td>2.4x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>3.9x</td>
<td>4.5x</td>
<td>4.6x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>4.5x</td>
<td>6.6x</td>
<td>6.6x</td>
</tr>
</tbody>
</table>
## TL; DR

### Speedups for Diamond Si (3x3x3)

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.9x</td>
<td>2.5x</td>
<td>3.1x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>4.4x</td>
<td>5.8x</td>
<td>5.7x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>5.2x</td>
<td>6.7x</td>
<td>7.8x</td>
</tr>
<tr>
<td>Entire Calculation</td>
<td>3.8x</td>
<td>3.7x</td>
<td>4.0x</td>
</tr>
</tbody>
</table>

**But does it scale?**

---

### Speedups for 103 Compound Test Set

<table>
<thead>
<tr>
<th></th>
<th>Ivy Bridge/GP100</th>
<th>Haswell/P100</th>
<th>Skylake/V100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCF Iteration</td>
<td>2.4x</td>
<td>2.1x</td>
<td>2.4x</td>
</tr>
<tr>
<td>SCF Iteration + Forces</td>
<td>3.9x</td>
<td>4.5x</td>
<td>4.6x</td>
</tr>
<tr>
<td>SCF Iteration + Forces + AS</td>
<td>4.5x</td>
<td>6.6x</td>
<td>6.6x</td>
</tr>
</tbody>
</table>
Strong Scaling for GPGPU Calculations: 375 Atom Bi$_2$Se$_3$ Slab

- Bi$_2$Se$_3$ Slab
  - 5x5x1 Supercell
  - 375 atoms

- 25 Å slab, 32 Å vacuum

- FHI-aims, tight settings (47.6 basis functions/atom)

- PBE functional

---

**Node**

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Titan Cray XK7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>1x AMD Opteron 6274</td>
</tr>
<tr>
<td></td>
<td>(16 cores, Bulldozer)</td>
</tr>
<tr>
<td><strong>GPGPU</strong></td>
<td>1x Tesla K20X</td>
</tr>
<tr>
<td></td>
<td>(Kepler)</td>
</tr>
<tr>
<td><strong>MPI Tasks/GPGPU</strong></td>
<td>16/1</td>
</tr>
<tr>
<td><strong>Compilers/Libraries</strong></td>
<td>PGI 17.9.0, Cray LibSci 16.11.1, Cray MPICH 7.6.3, CUDA 7.5</td>
</tr>
</tbody>
</table>
SCF Iteration + Forces + Stress on Titan: 375 Atom Bi$_2$Se$_3$ Slab

Bi$_2$Se$_3$ Slab
5x5x1 Supercell
375 atoms
FHI-aims, tight settings, PBE
47.6 basis functions per atom (17850 total)

Total Time for Iteration: 2.6x - 3.1x
Density + Stress: 3.6x - 4.0x
Hamiltonian: 2.9x - 3.5x
KS Solution: N/A
Hartree Sum: N/A
SCF Iteration + Forces + Stress on Titan: 375 Atom Bi$_2$Se$_3$ Slab

- Bi$_2$Se$_3$ Slab
- 5x5x1 Supercell
- 375 atoms
- FHI-aims, tight settings, PBE

47.6 basis functions per atom (17850 total)

Batch integration scales ideally, both for CPU and GPGPU runs

Total Time for Iteration: 2.6x - 3.1x
Density + Stress: 3.6x - 4.0x
Hamiltonian: 2.9x - 3.5x
KS Solution: N/A
Hartree Sum: N/A
Conclusion and Acknowledgements

All-electron real-space DFT can be effectively GPGPU-accelerated, with ideal scaling on HPC resources, using domain decomposition.

For realistic (non-hybrid) calculations with geometry relaxation, can expect $\approx 3x-4x$ speedup with FHI-aims on modern architectures.

GPGPU acceleration of real-space operations shown in this talk available on mainline FHI-aims git repo, production-ready.

Work supported by the LDRD Program of ORNL managed by UT-Battle, LLC, for the U.S. DOE and by the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro GP100 and Titan V used for local development, as well as access to their PSG cluster.