High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs

John Stone, Jan Saam, David Hardy, Kirby Vandivort, Wen-mei Hwu, Klaus Schulten
University of Illinois at Urbana-Champaign
http://www.ks.uiuc.edu/Research/gpu/

IACAT Accelerator Workshop, January 23, 2009
VMD

- VMD – “Visual Molecular Dynamics”
- Visualization of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry data, particle systems
- User extensible with scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/
Molecular Orbitals

- Visualization of MOs aids in understanding the chemistry of molecular system
- MO spatial distribution is correlated with probability density for an electron
- Algorithms for computing other interesting properties are similar, and can share code
Computing Molecular Orbitals

• Calculation of high resolution MO grids can require tens to hundreds of seconds in existing tools

• Existing tools cache MO grids as much as possible to avoid recomputation:
  – Doesn’t eliminate the wait for initial calculation
  – Can consume a lot of memory…
  – Hampers interactivity
Animating Molecular Orbitals

• Animation of molecular dynamics trajectories is helpful in gaining insight into simulation results
• To do the same for QM or QM/MM simulations one must compute MOs at 10 fps or more
• >100x speedup (GPU) over existing codes will now make this possible!!
MO Data-parallel Decomposition

- Compute a regularly spaced 3-D grid of MO amplitude values in the region surrounding the molecule.
- Each grid point can be computed independently.
- 3-D grid can be decomposed into 2-D slices which can be independently processed on one or more GPUs/CPUs.
CUDA Block/Grid Decomposition

Small 8x8 thread blocks afford large per-thread register count:

Each thread computes 1 MO amplitude.

Grid padding optimizes global mem. perf.

Padding waste

Grid of thread blocks:
MO Kernel for One Grid Point (Simplified C)

```c
for (at=0; at<numatoms; at++) {
    int prim_counter = atom_basis[at];
    calc_distances_to_atom(&atompos[at], &xdist, &ydist, &zdist, &dist2, &xdiv);
    for (contracted_gto=0.0f, shell=0; shell < num_shells_per_atom[at]; shell++) {
        int shell_type = shell_symmetry[shell_counter];
        for (prim=0; prim < num_prim_per_shell[shell_counter];  prim++) {
            float exponent      = basis_array[prim_counter ];
            float contract_coeff = basis_array[prim_counter + 1];
            contracted_gto += contract_coeff * expf(-exponent*dist2);
            prim_counter += 2;
        }
        for (tmpshell=0.0f, j=0, zdp=1.0f; j<=shell_type; j++, zdp*=zdist) {
            int imax = shell_type - j;
            for (i=0, ydp=1.0f, xdp=pow(xdist, imax); i<=imax; i++, ydp*=ydist, xdp*=xdiv)
                tmpshell += wave_f[ifunc++] * xdp * ydp * zdp;
        }
        value += tmpshell * contracted_gto;
        shell_counter++;
    }
} ...
```

Loop over atoms
Loop over shells
Loop over primitives: largest component of runtime, particularly due to expf()
Loop over angular momenta
GPU MO Kernel Design Observations

• Loop unrolling and specialization used to efficiently process angular momenta for the most common shell types
• Low ratio of FLOPS per array reference
• Must achieve very high effective memory bandwidth on references to coefficient arrays in the innermost loops
• Nested loops traverse multiple coefficient arrays of varying length, complicates things significantly
Preprocessing of Atoms, Basis Set, and Wavefunction Coefficients

• Must make effective use of high bandwidth, low-latency GPU on-chip memory, or CPU cache:
  – Overall storage requirement reduced by eliminating any duplicate basis set coefficients
  – Sorting atoms by element type allows re-use of basis set coefficients for subsequent atoms of identical type

• Padding, alignment of arrays guarantees coalesced GPU global memory accesses, CPU SSE loads
Use of GPU On-chip Memory

• If total data less than 64 kB:
  – GPU constant memory can broadcast to all threads
  – No global memory accesses!

• For large models, shared memory can be used as a program-managed cache:
  – Load tiles of coefficient data on-demand
  – Key to performance is to pull tile loading checks outside of the performance-critical inner loops, tiles must be large enough to service whole loop passes
  – Only 27% slower than hardware caching provided by constant memory (GT200)
Traversing of Atom Type, Basis Set, Shell Type, and Wavefunction Coefficients

- Loop iterations always access same or consecutive array elements:
  - Yields good constant memory cache performance
  - Increase shared memory tile reuse
Array tile loaded in GPU shared memory. Tile is a multiple of coalescing block size.

Surrounding data, unreferenced by next batch of loop iterations

64-Byte memory coalescing block boundaries

Full tile padding

Coefficient array in GPU global memory
VMD MO Performance Results for $C_{60}$

<table>
<thead>
<tr>
<th>Kernel</th>
<th>cores/GPUs</th>
<th>Runtime (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Q6600, gcc-cephes</td>
<td>1</td>
<td>200.22</td>
<td>0.23</td>
</tr>
<tr>
<td>Intel Q6600, gcc-cephes</td>
<td>4</td>
<td>51.52</td>
<td>0.90</td>
</tr>
<tr>
<td>Intel Q6600, icc-sse-cephes</td>
<td>1</td>
<td>46.58</td>
<td>1.00</td>
</tr>
<tr>
<td>Intel Q6600, icc-sse-approx</td>
<td>1</td>
<td>14.82</td>
<td>3.14</td>
</tr>
<tr>
<td>Intel Q6600, icc-approx</td>
<td>4</td>
<td>13.13</td>
<td>3.55</td>
</tr>
<tr>
<td>Intel Q6600, icc-sse-cephes</td>
<td>4</td>
<td>11.74</td>
<td>3.97</td>
</tr>
<tr>
<td>Intel Q6600, gcc-approx</td>
<td>4</td>
<td>10.21</td>
<td>4.56</td>
</tr>
<tr>
<td>Intel Q6600, icc-sse-approx</td>
<td>4</td>
<td>3.76</td>
<td>12.38</td>
</tr>
<tr>
<td>CUDA 8800 GTX (G80), tiled-shared-approx</td>
<td>1</td>
<td>1.05</td>
<td>44.36</td>
</tr>
<tr>
<td>CUDA 8800 GTX (G80), tiled-shared</td>
<td>1</td>
<td>0.89</td>
<td>51.98</td>
</tr>
<tr>
<td>CUDA 8800 GTX (G80), const-cache-approx</td>
<td>1</td>
<td>0.63</td>
<td>73.93</td>
</tr>
<tr>
<td>CUDA 8800 GTX (G80), const-cache</td>
<td>1</td>
<td>0.57</td>
<td>81.72</td>
</tr>
<tr>
<td>CUDA GTX 280 (GT200), tiled-shared-approx</td>
<td>1</td>
<td>0.54</td>
<td>85.62</td>
</tr>
<tr>
<td>CUDA GTX 280 (GT200), tiled-shared</td>
<td>1</td>
<td>0.46</td>
<td>100.38</td>
</tr>
<tr>
<td>CUDA GTX 280 (GT200), const-cache-approx</td>
<td>1</td>
<td>0.41</td>
<td>113.61</td>
</tr>
<tr>
<td>CUDA GTX 280 (GT200), const-cache</td>
<td>1</td>
<td>0.37</td>
<td>125.89</td>
</tr>
</tbody>
</table>
Performance Evaluation Test Cases

<table>
<thead>
<tr>
<th>system</th>
<th>atoms</th>
<th>basis set</th>
<th>basis functions (unique)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C60-a</td>
<td>60</td>
<td>STO-3G</td>
<td>300 (5)</td>
</tr>
<tr>
<td>C60-b</td>
<td>60</td>
<td>6-31Gd</td>
<td>900 (15)</td>
</tr>
<tr>
<td>Thr-a</td>
<td>17</td>
<td>STO-3G</td>
<td>49 (16)</td>
</tr>
<tr>
<td>Thr-b</td>
<td>17</td>
<td>6-31+Gd</td>
<td>170 (59)</td>
</tr>
<tr>
<td>Kr-a</td>
<td>1</td>
<td>STO-3G</td>
<td>19 (19)</td>
</tr>
<tr>
<td>Kr-b</td>
<td>1</td>
<td>cc-pVQZ</td>
<td>84 (84)</td>
</tr>
</tbody>
</table>

Several test cases were used to evaluate MO calculation performance over a range of problem sizes and varying degrees of basis set complexity.
Performance Evaluation: Molekel, MacMolPlt, and VMD

<table>
<thead>
<tr>
<th>Program/Kernel</th>
<th>cores</th>
<th>C60-a</th>
<th>C60-b</th>
<th>Thr-a</th>
<th>Thr-b</th>
<th>Kr-a</th>
<th>Kr-b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molekel CPU</td>
<td>1</td>
<td>39</td>
<td>25</td>
<td>175</td>
<td>108</td>
<td>617</td>
<td>138</td>
</tr>
<tr>
<td>MacMolPlt CPU</td>
<td>4</td>
<td>97</td>
<td>66</td>
<td>361</td>
<td>265</td>
<td>2668</td>
<td>632</td>
</tr>
<tr>
<td>VMD gcc-cephes</td>
<td>4</td>
<td>126</td>
<td>100</td>
<td>518</td>
<td>374</td>
<td>2655</td>
<td>892</td>
</tr>
<tr>
<td>VMD icc-sse-cephes</td>
<td>4</td>
<td>658</td>
<td>429</td>
<td>2428</td>
<td>1366</td>
<td>10684</td>
<td>2968</td>
</tr>
<tr>
<td>VMD gcc-approx</td>
<td>4</td>
<td>841</td>
<td>501</td>
<td>2641</td>
<td>1828</td>
<td>11055</td>
<td>4060</td>
</tr>
<tr>
<td>VMD icc-sse-approx</td>
<td>4</td>
<td>2314</td>
<td>1336</td>
<td>8829</td>
<td>5319</td>
<td>33818</td>
<td>9631</td>
</tr>
<tr>
<td>VMD CUDA 8800 GTX</td>
<td>1</td>
<td>14166</td>
<td>8565</td>
<td>45015</td>
<td>32614</td>
<td>104576</td>
<td>61358</td>
</tr>
<tr>
<td>VMD CUDA GTX 280</td>
<td>1</td>
<td>21540</td>
<td>13338</td>
<td>62277</td>
<td>45498</td>
<td>119167</td>
<td>78884</td>
</tr>
</tbody>
</table>

Units: $10^3$ grid points/sec
Larger numbers indicate higher performance.
VMD MO Performance Summary

- Multi-core CPU (full-precision SSE) algorithm outperforms other tools by factor of 4.0x to 10x on the same number of cores
- CPU SSE exp() approximation improves CPU performance by another factor of 3x
- Single-GPU MO algorithm outperforms other tools by factor of 120x to 220x
Future Work

• Runtime generation of MO kernel code using new CUDA 2.1 / OpenCL APIs (eliminate basis set loop/branching)
• Use multi-pass computation and spatial decomposition and distance-based cutoff to truncate extremely rapidly decaying exponentials
• Tuning of Multi-GPU implementation to workaround small kernel launch delays that adversely impact animation speed
• Move subsequent MO volume gradient and isosurface computations entirely to GPU
Future Multi-GPU Optimizations

• Developing software framework to improve multi-GPU acceleration:
  – NUMA-aware GPU/CPU allocation
  – Host “thread pools” to maintain active connection to GPUs for low-latency kernel launches (e.g. MO animation)
  – Collaborating with NCSA staff on related issues for GPU wrapper library
## GPU Kernel Performance, Jan 2009

GeForce 8800GTX w/ CUDA 2.0  
http://www.ks.uiuc.edu/Research/gpu/  

<table>
<thead>
<tr>
<th>Calculation / Algorithm</th>
<th>Algorithm class</th>
<th>Speedup vs. Intel QX6700 CPU core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluorescence microphotolysis</td>
<td>Iterative matrix / stencil</td>
<td>12x</td>
</tr>
<tr>
<td>Pairlist calculation</td>
<td>Particle pair distance test</td>
<td>10-11x</td>
</tr>
<tr>
<td>Pairlist update</td>
<td>Particle pair distance test</td>
<td>5-15x</td>
</tr>
<tr>
<td>Molecular dynamics non-bonded force calc.</td>
<td>N-body cutoff force calculations</td>
<td>10x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20x (w/ pairlist)</td>
</tr>
<tr>
<td>Electron density approximation</td>
<td>Particle-grid w/ cutoff</td>
<td>15-23x</td>
</tr>
<tr>
<td>Full multilevel summation electrostatic calc.</td>
<td>Particle-grid, grid-grid w/ cutoff</td>
<td>20x</td>
</tr>
<tr>
<td>Direct Coulomb summation</td>
<td>Particle-grid</td>
<td>44x</td>
</tr>
<tr>
<td>Molecular orbital calculation</td>
<td>Particle-grid</td>
<td>80x</td>
</tr>
</tbody>
</table>
Acknowledgements

• Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign

• David Kirk and the CUDA team at NVIDIA

• NIH support: P41-RR05969
Publications

http://www.ks.uiuc.edu/Research/gpu/


Extra Slides…
Algorithm 1 Calculate an MO value $\Psi_{\nu}(\mathbf{r})$ at a lattice point $\mathbf{r}$ for given wavefunction and basis set coefficient arrays.

1: $\Psi_{\nu} \leftarrow 0.0$
2: ifunc $\leftarrow 0$ {index array of wavefunction coefficients}
3: shell_counter $\leftarrow 0$ {index array of shell numbers}
4: for $n = 1$ to $N$ do {loop over atoms}
5: \begin{align*} (x, y, z) &\leftarrow \mathbf{r} - \mathbf{r}_n \quad \{ \mathbf{r}_n \text{ is position of atom } n \} \\
R^2 &\leftarrow x^2 + y^2 + z^2 \\
\end{align*} 
6: prim_counter $\leftarrow \text{atom_basis}[n]$ {index arrays of basis set data}
7: for $l = 0$ to $\text{num_shells_per_atom}[n] - 1$ do {loop over shells}
8: \begin{align*} \phi_{\text{CGTO}} &\leftarrow 0.0 \\
9: \text{for } p = 0 \text{ to } \text{num_prim_per_shell}[\text{shell_counter}] - 1 &\text{ do } \{ \text{loop over primitives} \} \\
10: c'_p &\leftarrow \text{basis_c}[\text{prim_counter}], \quad \zeta_p &\leftarrow \text{basis_zeta}[\text{prim_counter}] \\
11: \phi_{\text{CGTO}} &\leftarrow \phi_{\text{CGTO}} + c'_p * \exp(-\zeta_p * R^2) \\
12: \text{prim_counter} &\leftarrow \text{prim_counter} + 1 \\
13: \end{align*} 
14: end for
15: for all $i$ such that $0 \leq i \leq \text{shell_type}[\text{shell_counter}]$ do {loop over angular momenta}
16: \begin{align*} j_{\text{max}} &\leftarrow \text{shell_type}[\text{shell_counter}] - i \\
17: \text{for all } j \text{ such that } 0 \leq j \leq j_{\text{max}} &\text{ do} \\
18: k &\leftarrow j_{\text{max}} - j \\
19: c' &\leftarrow \text{wavefunction}[\text{ifunc}] \\
20: \Psi_{\nu} &\leftarrow \Psi_{\nu} + c' * \phi_{\text{CGTO}} * x^i * y^j * z^k \\
21: \text{ifunc} &\leftarrow \text{ifunc} + 1 \\
22: \text{end for} \\
23: \text{end for} \\
24: \text{shell_counter} &\leftarrow \text{shell_counter} + 1 \\
25: \end{align*} 
26: end for
27: return $\Psi_{\nu}$