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

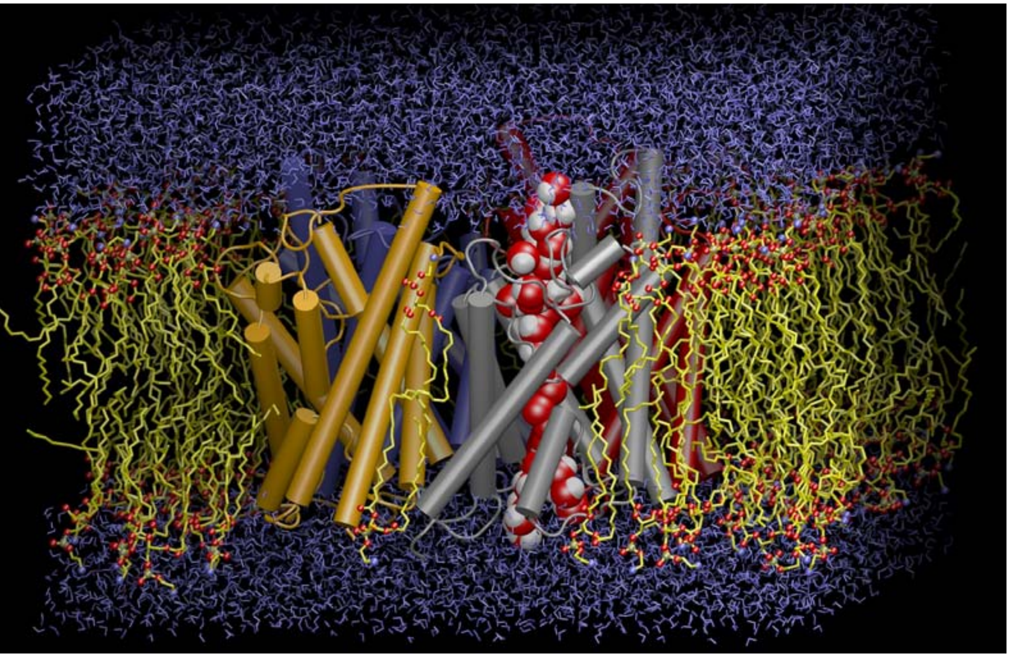
University of Illinois at Urbana-Champaign

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

IACAT Accelerator Workshop, January 23, 2009

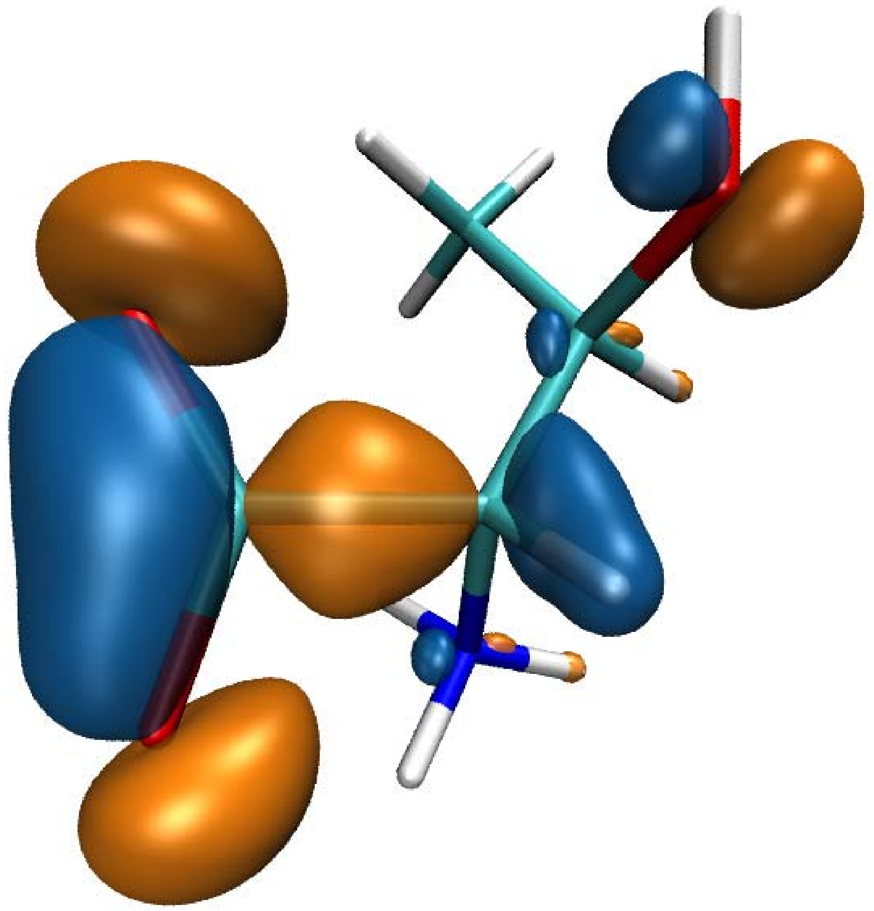
VMD

- 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/



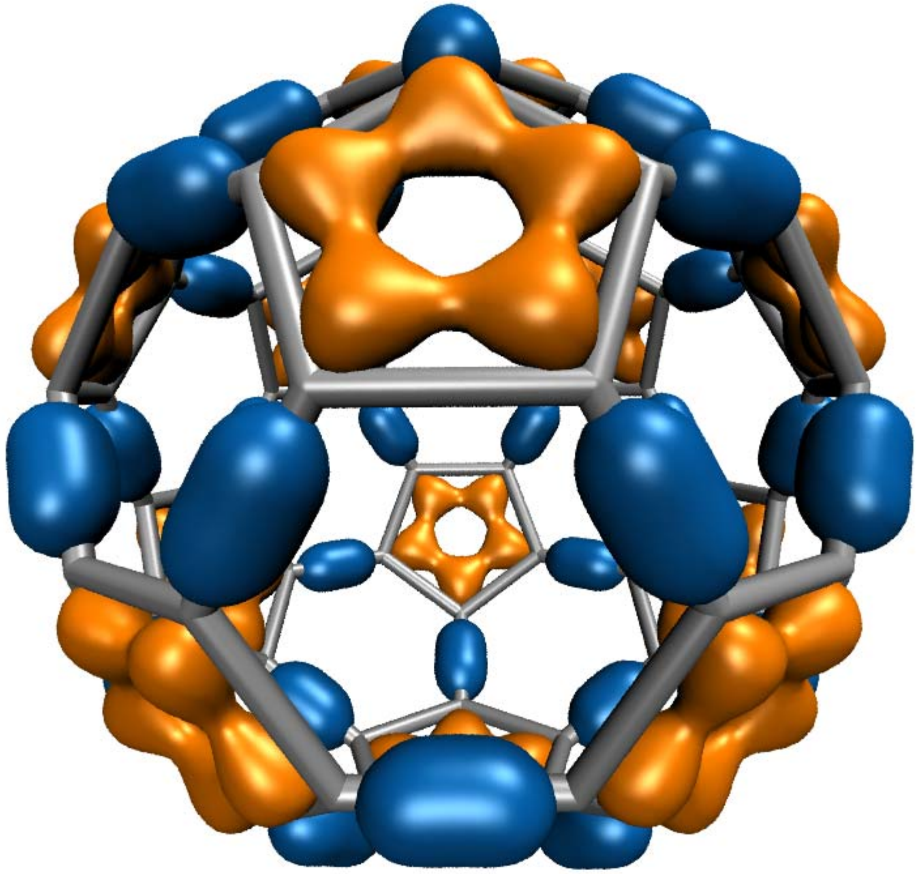
Molecular Orbitals

- 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



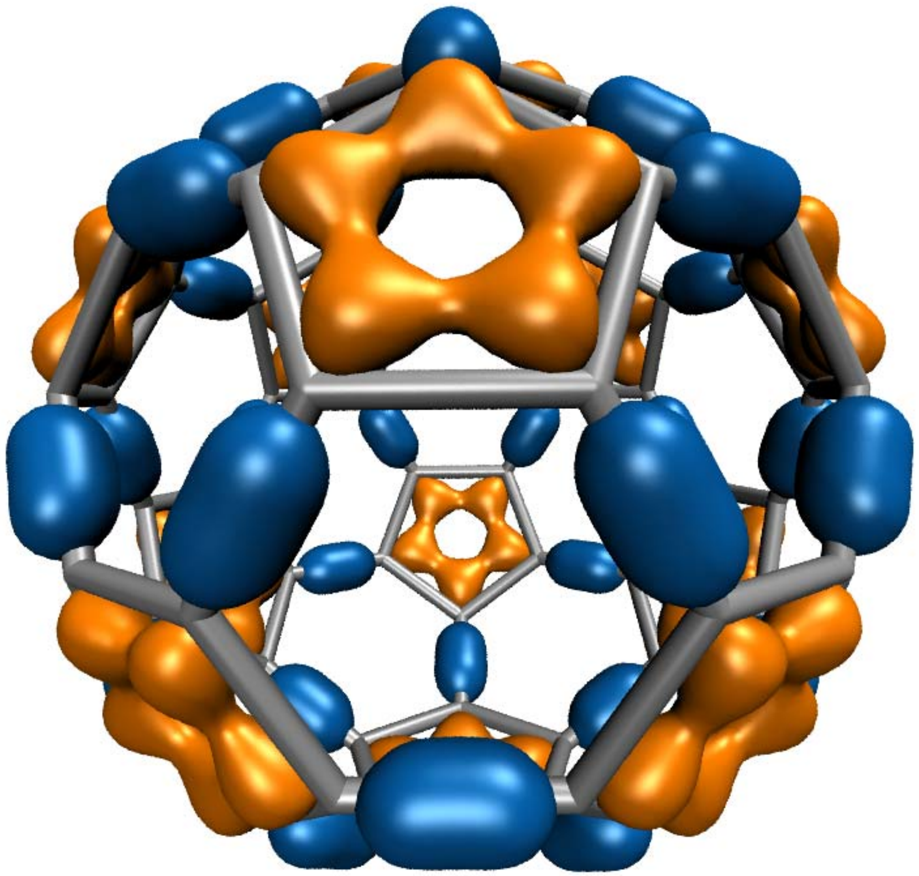
Computing Molecular Orbitals

- 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



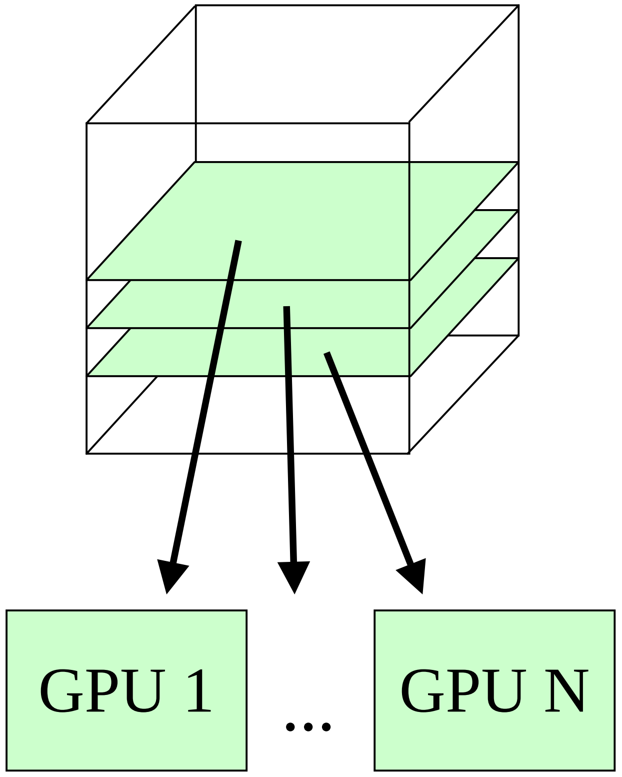
Animating Molecular Orbitals

- 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!!



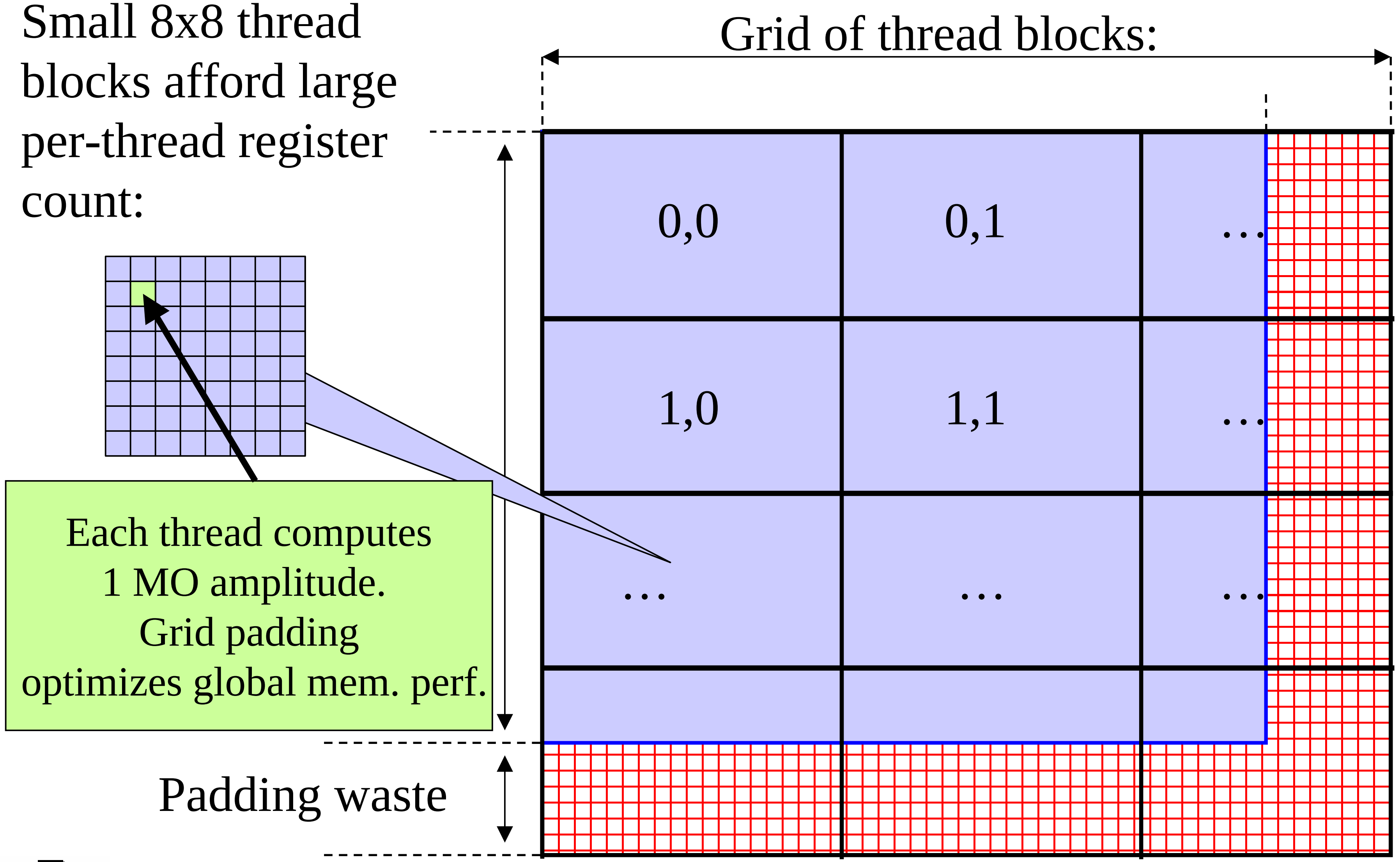
MO Data-parallel Decomposition

- 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.

MO Kernel for One Grid Point (Simplified 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

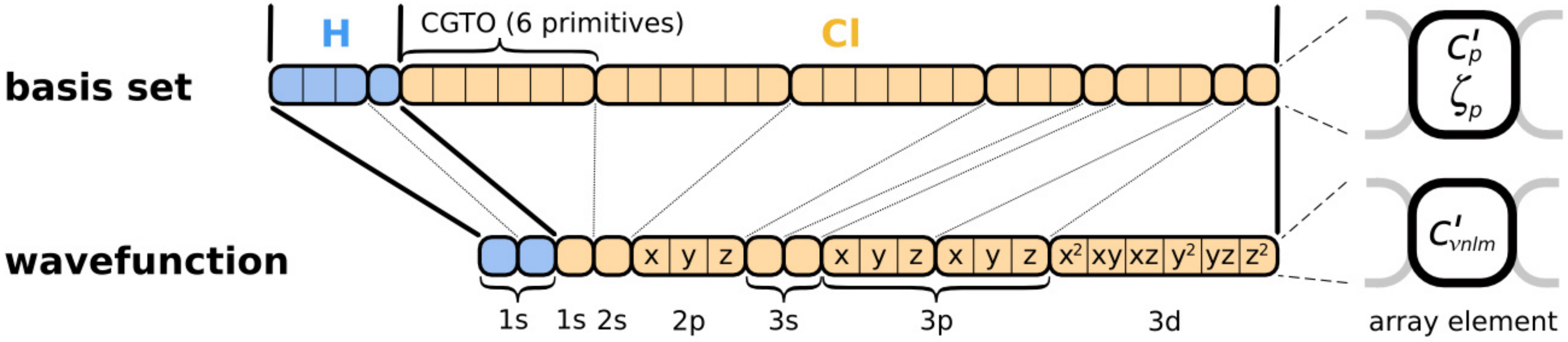
Preprocessing of Atoms, Basis Set, and Wavefunction Coefficients

- 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

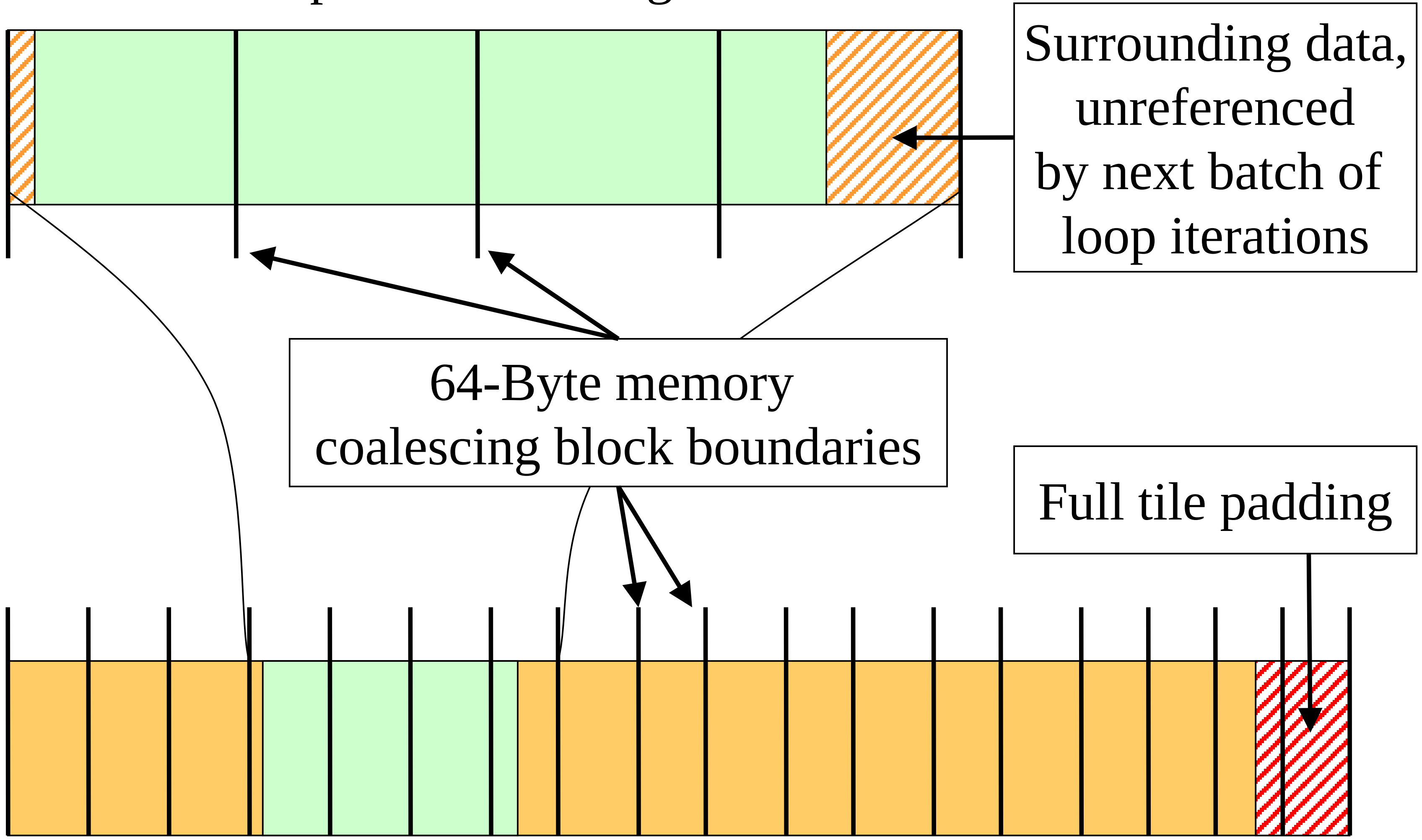
- If total data less than 64 kB:
  - GPU constant memory can broadcast to all threads
  - No global memory accesses!
- For large models, shared memory can be used as a program-managed cache:
  - Load tiles of coefficient data on-demand
  - Key to performance is to pull tile loading checks outside of 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)

Traversal of Atom Type, Basis Set, Shell Type, and Wavefunction Coefficients



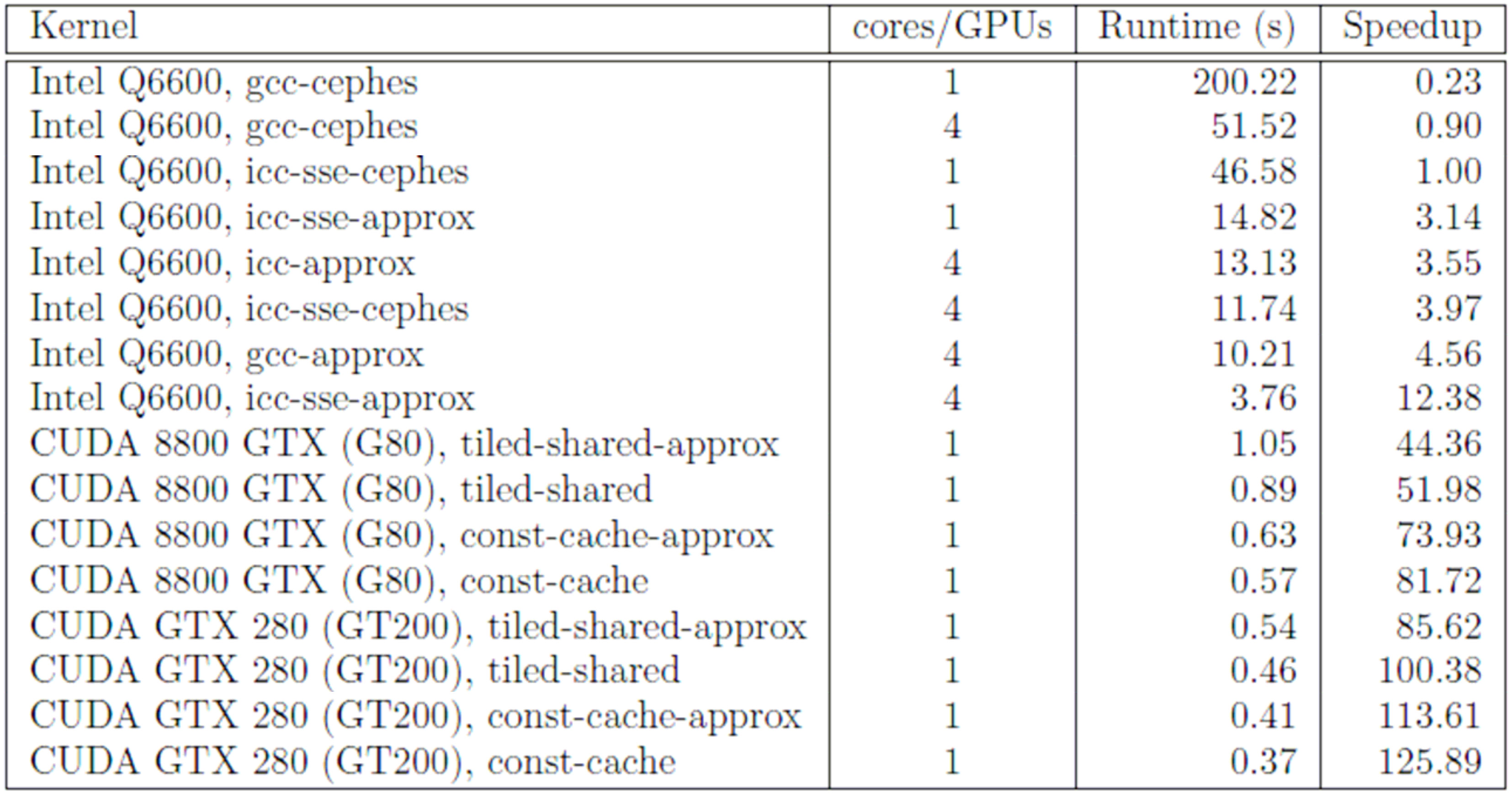
- Loop iterations always access same or consecutive array elements:
  - Yields good constant memory cache performance
  - Increase shared memory tile reuse

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}



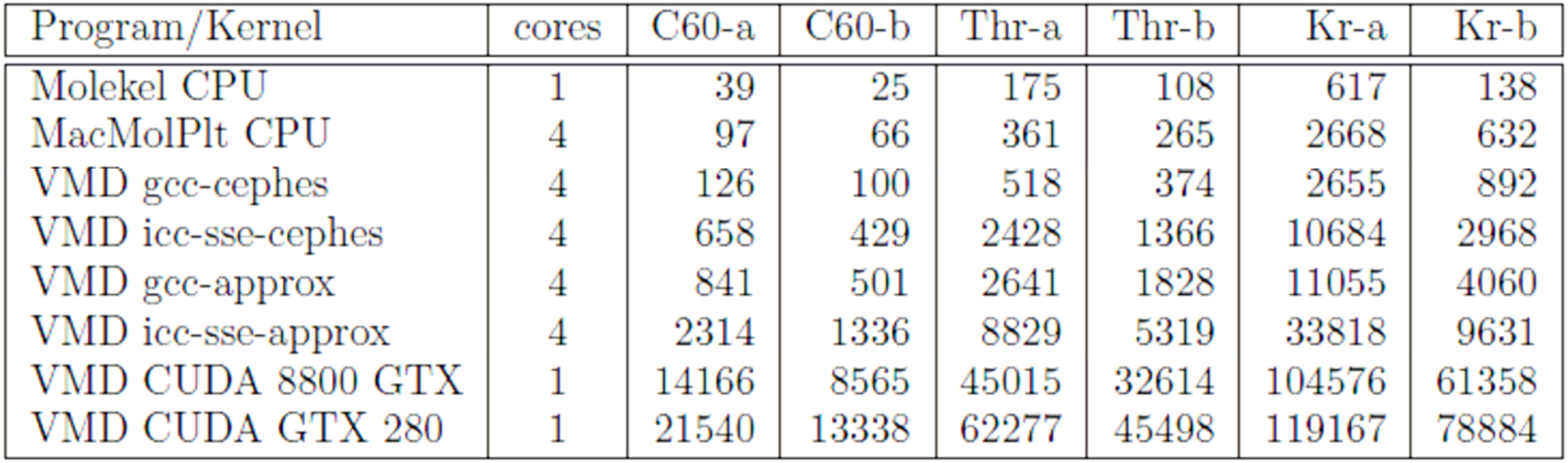
Kernel	cores/GPUs	Runtime (s)	Speedup
Intel Q6600, gcc-cephes	1	200.22	0.23
Intel Q6600, gcc-cephes	4	51.52	0.90
Intel Q6600, icc-sse-cephes	1	46.58	1.00
Intel Q6600, icc-sse-approx	1	14.82	3.14
Intel Q6600, icc-approx	4	13.13	3.55
Intel Q6600, icc-sse-cephes	4	11.74	3.97
Intel Q6600, gcc-approx	4	10.21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

Performance Evaluation Test Cases

	system	atoms	basis set	basis functions (unique)
C60-a	carbon-60	60	STO-3G	300 (5)
C60-b	carbon-60	60	6-31Gd	900 (15)
Thr-a	threonine	17	STO-3G	49 (16)
Thr-b	threonine	17	6-31+Gd	170 (59)
Kr-a	krypton	1	STO-3G	19 (19)
Kr-b	krypton	1	cc-pVQZ	84 (84)

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



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

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/

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

Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- David Kirk and the CUDA team at NVIDIA
- NIH support: P41-RR05969

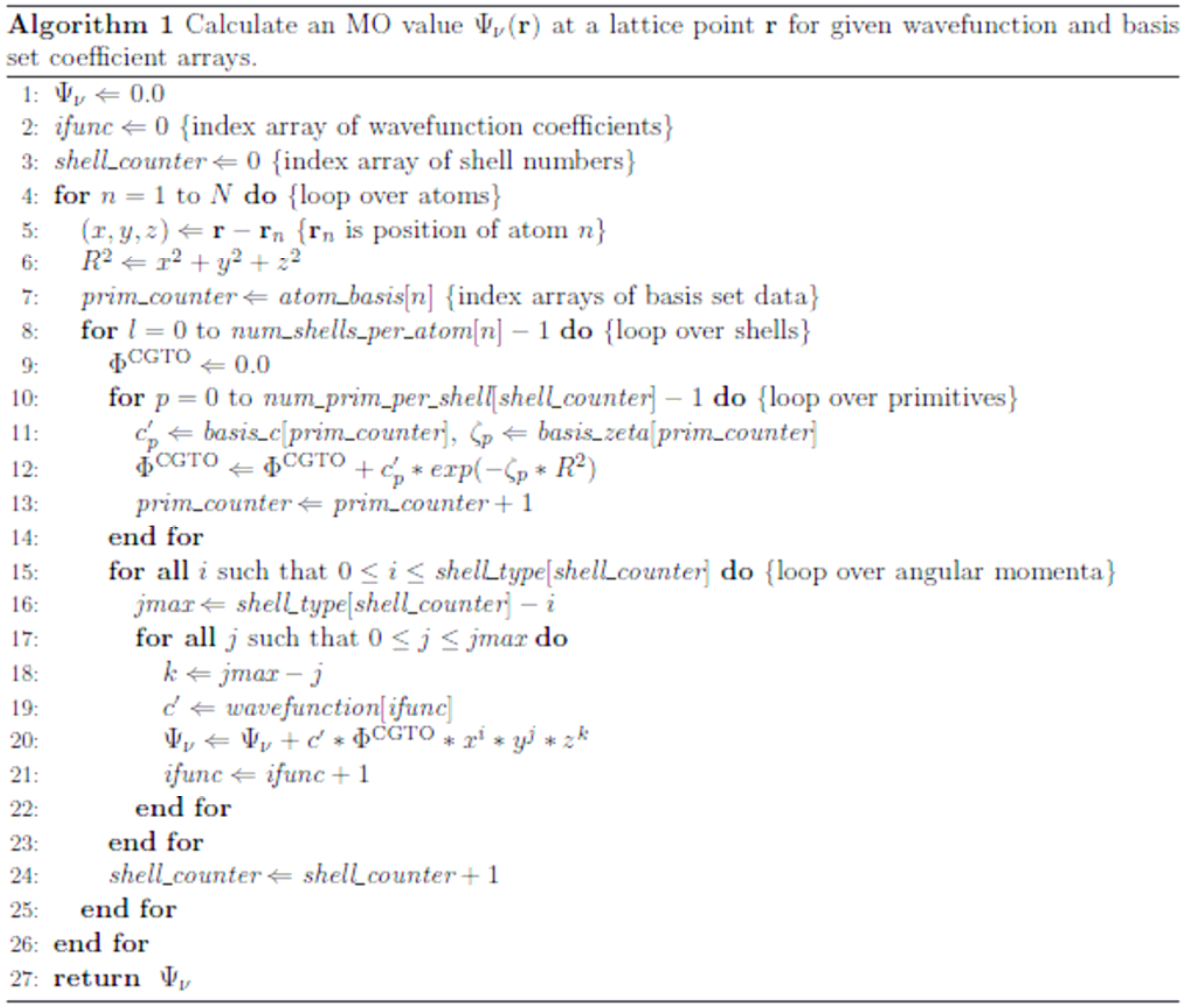
Publications

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

- Multilevel summation of electrostatic potentials using graphics processing units. David J. Hardy, John E. Stone, and Klaus Schulten. *J. Parallel Computing*, 2009. In press.
- Adapting a message-driven parallel application to GPU-accelerated clusters. J. Phillips, J. Stone, K. Schulten. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE Press, 2008.
- GPU acceleration of cutoff pair potentials for molecular modeling applications. C. Rodrigues, D. Hardy, J. Stone, K. Schulten, W. Hwu. *Proceedings of the 2008 Conference On Computing Frontiers*, pp. 273-282, 2008.
- GPU computing. J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, J. Phillips. *Proceedings of the IEEE*, 96:879-899, 2008.
- Accelerating molecular modeling applications with graphics processors. J. Stone, J. Phillips, P. Freddolino, D. Hardy, L. Trabuco, K. Schulten. *J. Comp. Chem.*, 28:2618-2640, 2007.
- Continuous fluorescence microphotolysis and correlation spectroscopy. A. Arkhipov, J. Hüve, M. Kahms, R. Peters, K. Schulten. *Biophysical Journal*, 93:4006-4017, 2007.

Extra Slides…

MO Algorithm Detail



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:    $(x, y, z) \Leftarrow \mathbf{r} - \mathbf{r}_n$ {$\mathbf{r}_n$ is position of atom $n$}
6:    $R^2 \Leftarrow x^2 + y^2 + z^2$
7:    $prim\_counter \Leftarrow atom\_basis[n]$ {index arrays of basis set data}
8:    for $l = 0$ to $num\_shells\_per\_atom[n] - 1$ do {loop over shells}
9:      $\Phi^{\mathrm{CGTO}} \Leftarrow 0.0$
10:     for $p = 0$ to $num\_prim\_per\_shell[shell\_counter] - 1$ do {loop over primitives}
11:       $c'_p \Leftarrow basis\_c[prim\_counter]$, $\zeta_p \Leftarrow basis\_zeta[prim\_counter]$
12:       $\Phi^{\mathrm{CGTO}} \Leftarrow \Phi^{\mathrm{CGTO}} + c'_p * exp(-\zeta_p * R^2)$
13:       $prim\_counter \Leftarrow prim\_counter + 1$
14:     end for
15:     for all $i$ such that $0 \le i \le shell\_type[shell\_counter]$ do {loop over angular momenta}
16:       $jmax \Leftarrow shell\_type[shell\_counter] - i$
17:       for all $j$ such that $0 \le j \le jmax$ do
18:         $k \Leftarrow jmax - j$
19:         $c' \Leftarrow wavefunction[ifunc]$
20:         $\Psi_\nu \Leftarrow \Psi_\nu + c' * \Phi^{\mathrm{CGTO}} * x^i * y^j * z^k$
21:         $ifunc \Leftarrow ifunc + 1$
22:       end for
23:     end for
24:     $shell\_counter \Leftarrow shell\_counter + 1$
25:   end for
26: end for
27: return $\Psi_\nu$
```