Multi-GPU Accelerated Large Scale Electronic Structure Calculations

Samuli Hakala
COMP Centre of Excellence
Department of Applied Physics
Aalto University School of Science
Email: samuli.hakala@aalto.fi

GPAW Workshop, May 2013

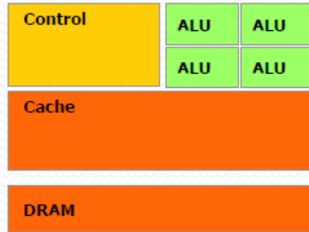
Aalto University
School of Science

Overview

1. Background
2. Methods
3. Results
4. Summary

Collaborators:

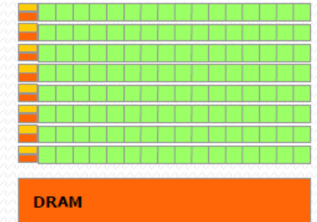
Ville Havu, Jussi Enkovaara and Risto Nieminen @ Aalto University
Christopher O'Grady, Lin Li, Jun Yan @ SUNCAT Center in Stanford/SLAC



CPU

- Large caches
- Sophisticated control
- Powerful ALU
  - Reduced operation latency

GPU



- Small caches
- Simple control
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies

Porting to GPUs

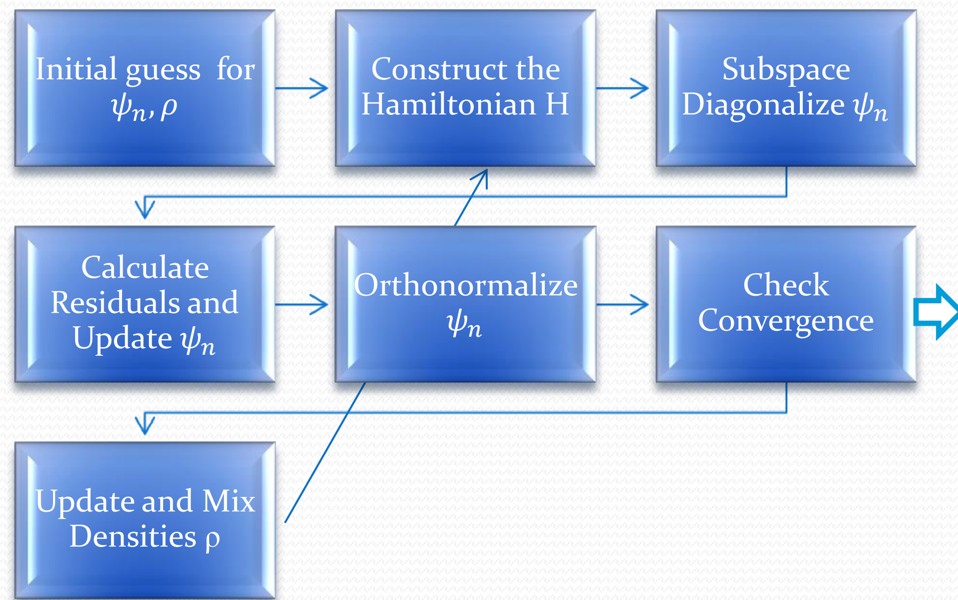
- Identify numerical bottlenecs on CPU and replace them with GPU equivalents.
- Minimize memory transfers between host and devive.
- Usually attaining good performance requires also porting a lot of non-intensive routines.
- Performance:
  - GPU: Nvidia Tesla M2070 (DP 515 Gflops. Mem bw: 150GB/s) Tesla K20 (DP 1170 Gflops. Mem bw: 208GB/s)
  - CPU: Intel Xeon X5650 (DP 10.664 Gflops per core. Total: 64 Gflops. Mem bw: 32GB/s)
  - Theoretically GPU 18 times faster and has 6.5 times the bandwidth!

GPAW GPU Implementation

- GPAW coded in Python with extensions written in C for performance critical parts.
- Goal for GPU implementation: High level algorithms and code stay same. Change only low level routines
- NumPy toolkit is used to perform operations on Python using multidimensional arrays.
- Standard libraries are used for linear algebra operations (BLAS, LAPACK, SCALAPACK)
- Use PyCUDA toolkit, NVIDIA CuBLAS library and several custom CUDA kernels.
- Double precision arithmetic

Ground state solution in GPAW

- An iterative procedure called Self-Consistent Field (SCF) calculation
- Most computationally intensive parts: Construction of the Hamiltonian, subspace diagonalization, refining of of wavefunctions and ortrhonormalization.
- Uniform real space grids
- A coarse grid is used for the wave functions and a fine grid for potentials and densities



Constructing the Hamiltonian

- The most time-consuming parts are the calculation of the Hartree and the exchange-correlation potentials
- Hartree potential is solved from the Poisson equation
- Done for a fine grid using a multi-grid solver
- Basic operations are: finite difference stencils for the Laplace operator and restriction and interpolation between coarser and finer grids
- Custom CUDA kernels for all of these operations
- Solved entirely on GPUs
- Speed-ups between 10-20 on a single GPU

LibXC on GPUs

- A reusable library of >250 exchange-correlation functionals
- Used by 15 different codes (Abinit, GPAW, BigDFT, etc.)
- Can be a performance bottleneck for small systems
- Can "clone" existing functionals for GPU use with fairly minimal changes to existing LibXC code and parallelizes well over grid points
- More information:
  - https://confluence.slac.stanford.edu/display/SUNCAT/libxc+on+GPUs
- Work by Lin Li, Jun Yan, Christopher O'Grady (Stanford/SLAC)

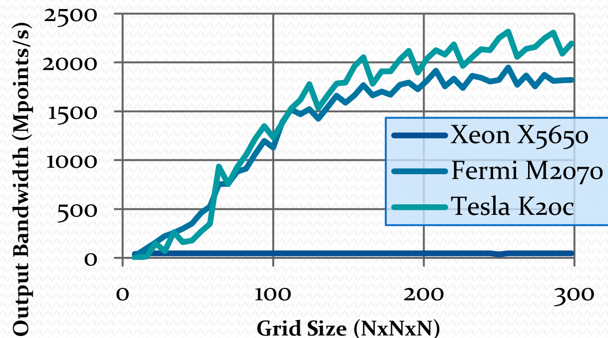
Functional	Type	Speedup ((GPU+CPU)/CPU)
PW, PW Mode, OB PW, PW RPA	LDA Correlation	**23,23,23,37**
PBE, PBE sol, xPBE, PBE JRGX, RGE2, APBE	GGA Correlation	**56, 58, 58, 58, 58, 58**
RPBE	GGA Exchange	**95**
TPSS	MGGA Exchange	**51**

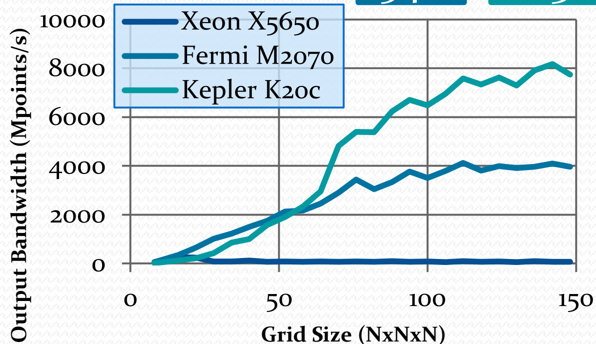
Updating the Wave functions

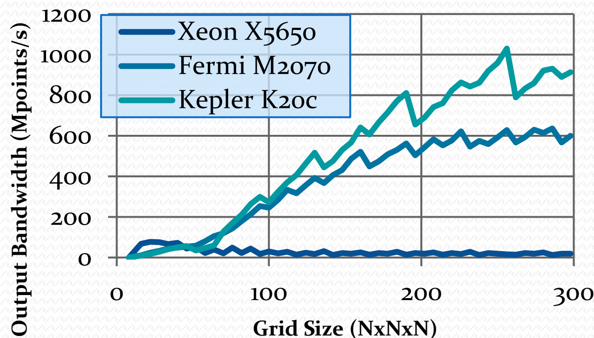
- Eigensolver: Residual minimization scheme - direct inversion in the iterative subspace (RMM-DIIS)
- Wave functions are updated with the residuals $R_{nG} = (\hat{H} - \epsilon_n \hat{S})\tilde{\psi}_{nG}$
- Accelerated using preconditioned residuals by solving approximately a Poisson equation with a multigrid method
- Explicit subspace diagonalization and orthonormalization required
  - Subspace diagonalization: Hamiltonian is applied to the wave functions. The resulting matrix then diagonalized and multiplied by the wave-functions.
  - Orthonormalization: Overlap matrix is constructed by applying an ovelap operator. This is then Cholesky decomposed and multiplied with the wave functions.
  - Integrals of projector functions multiplied by wave functions and addition of projector functions multiplied by a matrix to the wave functions
- Avoid large memory transfers:
  - All wave functions are stored in GPU memory
  - All operations performed using GPU

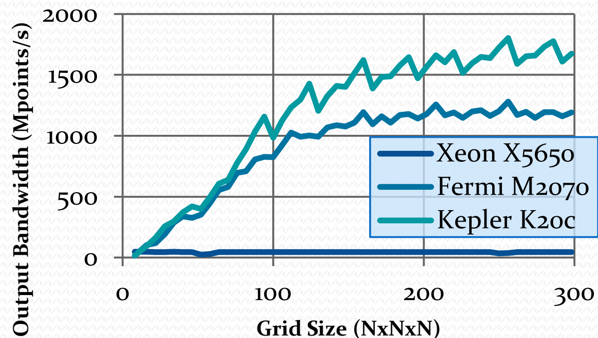
Stencil Operations

- Process the grid slice-by-slice
- Calculations perfomed using combination of registers and shared memory
- Parallelized over grid points
- Supports real, complex and periodic grids
- Speed-ups on large grids with Fermi 27-54x and with Kepler 37-103x



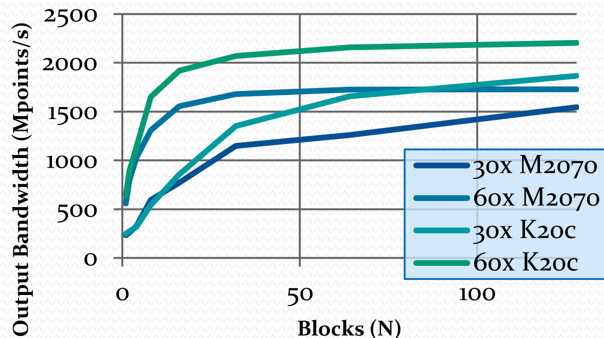


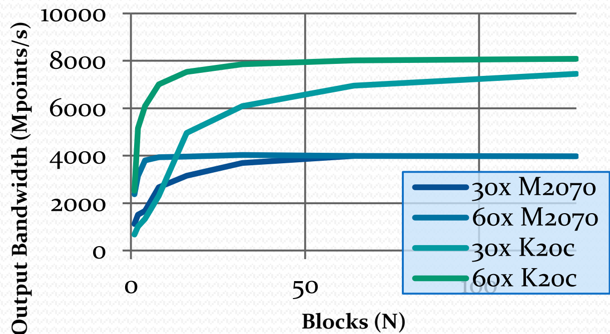


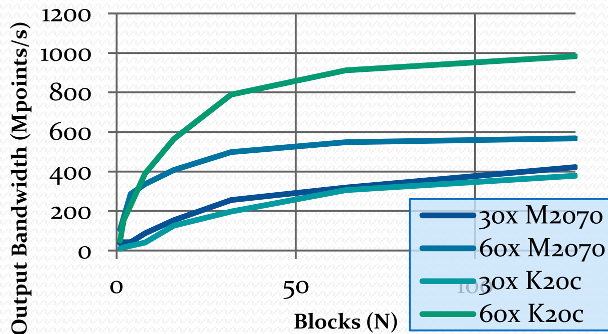


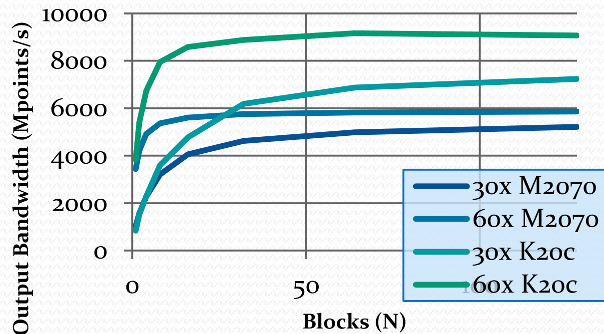
Batching on GPU

- Small grids cause performance issues
- Block of grids using one kernel
- Used in stencil operations and in several BLAS functions
- Can increase performance up to 10 times on a 30x30x30 grid and up to 5 times on a 60x60x60 grid









Getting and Building GPU GPAW

- Libraries needed: Libxc (2.0.0 or newer), CUDA Toolkit, PyCUDA 2012.1
- Get the CUDA branch:
  `svn co https://svn.fysik.dtu.dk/projects/gpaw/branches/cuda`
- In the `gpaw/c/cuda` directory:
  - edit `make.inc` for correct library/include directories
  - run make
- Add to `customize.py`:
  - `define_macros += [('GPAW_CUDA', '1')]`
  - Add libraries: `gpaw-cuda, cublas, cuda, xc`
- Continue with normal installation
- Most of the tests test suite should pass successfully

Using GPU GPAW

- Using CUDA in calculations:
  `gpaw-python --cuda Au224.py`
- Or pass cuda parameter to GPAW calculator:
  `calc = GPAW( ... , cuda=True , ... )`
- `Additional command line arguments:`
  - `--debug-cuda`
    Performs same operations with GPU and CPU and compares results
  - `--cuda-sync`
    Synchronizes CUDA context with GPAW timers

Ground State Performance

Bulk Silicon

- 95 atoms with periodic boundary conditions, 380 bands and 1 k-points. Complex grid size: 56x56x80.
- Time is in seconds per one SCF iteration.
- Intel Xeon X5650, NVIDIA Tesla M2070

Si95	CPU	GPU	%	S-Up
Poisson Solver	2.1	0.12	0.5%	**17**
Orthonormalization	60	5.0	23%	**12**
Precondition	19	1.5	6.6%	**12**
RMM-DIIS other	48	4.7	22%	**10**
Subspace Diag	84	5.6	25%	**14**
Other	4.5	5.1	23%	**0.8**
Total (SCF-Iter)	217	22		9.8

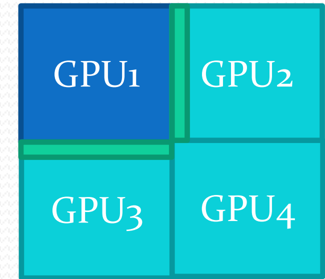
Fullerene

- C60 molecule with 240 valence electrons. Grid size: 80x80x84
- Intel Xeon X5650, NVIDIA Tesla M2070

C60	CPU	GPU	%	S-Up
	12	0.64	7.6%	**19**
	10	0.95	11%	**11**
	17	0.93	11%	**18**
	7.2	0.58	7%	**12**
	21	1.8	22%	**11**
	3.6	3.5	41%	**1.1**
	71	8.4		8.5

Multi-GPU Parallelization

- Parallelization is done with MPI
- Multiple GPUs can be used by domain decomposition or parallelization over k-points or spins
- Domain decomposition for the stencil operations involves exchanging boundary regions between neighboring nodes
- Communications between nodes require data movement: device memory → host memory → destinations node host memory → destinations node device memory.
- Overlaps receives, sends and computations in the middle part of the grid, BUT this causes issues with small grids
  - Small grids: Synchronous transfers
  - Medium grids: Asynchronous transfers
  - Large grids: Overlap calculations and asynchronous transfers
  - Combine of several wave functions and boundary regions into few large transfers

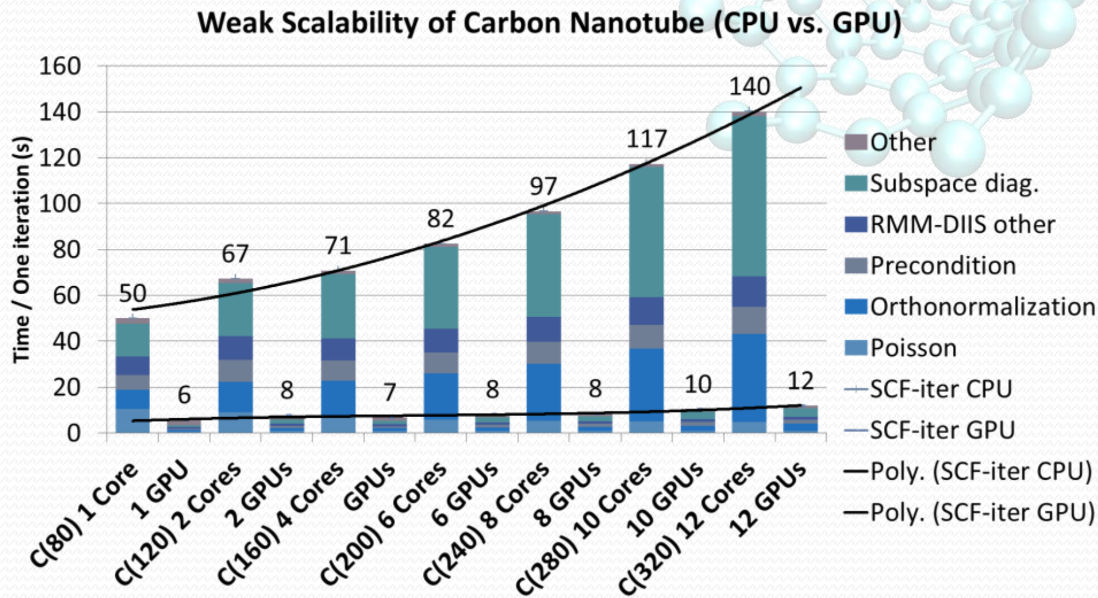


Using MPI

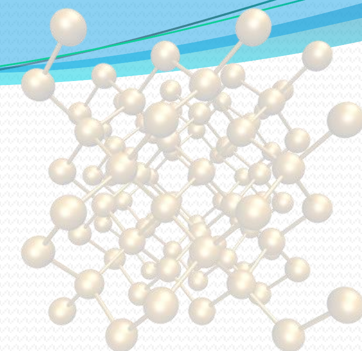
- MPI works automatically:
  `mpirun -np 4 gpaw-python --cuda Au224.py`
- GPU card selected based on MPI rank
- One-to-One mapping between GPU cards, CPU cores and MPI tasks is assumed
- Supports CUDA aware MPI implementations (mvapcih2, openmpi)
  - Needs `CUDA_MPI` definition in `customize.py` and `make.inc`
  - GPUDirect

Weak Scalability (Carbon)

- The size of a carbon nanotube and the number of MPI tasks are varied from 80 atoms (240 states) to 320 atoms (1280 states) and 1 task to 12 tasks.
- Comparison between equal number of GPUs and CPU cores.
- CPU: Intel Xeon X5650 GPU: NVIDIA Tesla M2070
- Calculations performed on Vuori cluster at CSC



# MPI tasks	1	2	4	6	8	10	12
Speed-Up	8.8	8.7	10.5	10.2	11.5	11.3	11.9



Strong Scalability

- Bulk silicon with 1151 atoms with periodic boundary conditions, 4604 bands and 1 k-point in the Brillouin zone.
- The number of GPUs is increased from 64 to 256.
- Grid size: 164x164x108
- Speed-up comparison to 64 GPUs.
- NVIDIA Tesla M2090
- Calculations performed on CURIE cluster in France at GENCI/CEA

Scalability of Si(1151)

Speed-up

4
3.5
3
2.5
2
1.5
1

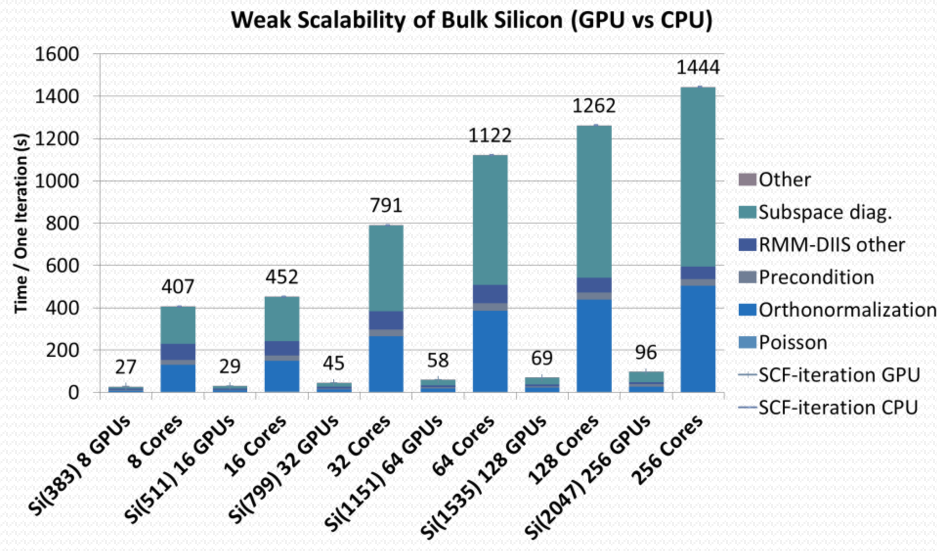
64
128
192
256

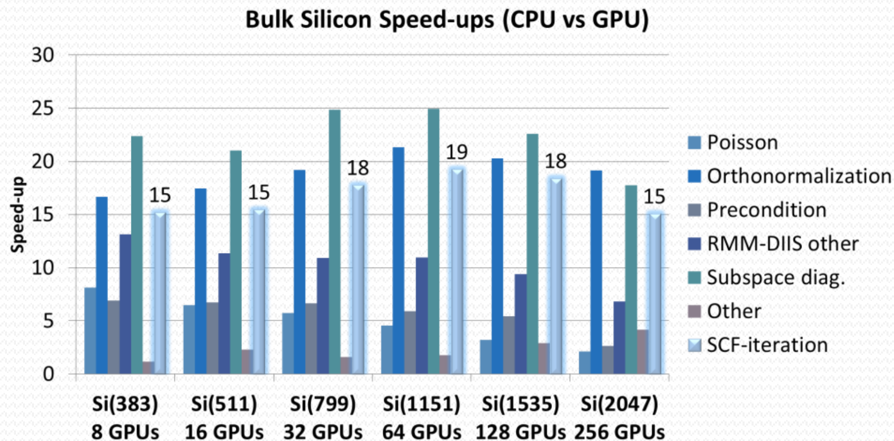
GPUs

Poisson 1.5%
Orthonormalization 33%
Precondition 11%
RMM-DIIS other 15%
Subspace diag. 38%
Other 1.6%
SCF-iteration
Ideal

Weak Scalability (Silicon)

- The size of bulk silicon system and the number of MPI tasks are varied from 383 atoms (1532 bands) to 2046 atoms (8188 bands) and 8 task to 256 tasks with periodic boundary conditions.
- The largest system requires about 1.3TB of memory for calculations.
- CPU: Intel Xeon E5640 GPU: NVIDIA Tesla M2090





Summary

- We have accelerated the most numerically intensive parts of ground state DFT calculations
- Overall speed-ups in our tests varied from 8.5 to 19 depending on system size.
- Our multi-GPU implementation scales well even on large hybrid clusters.
- Code is available at GPAW Subversion repository.
- Acknowledgements to CSC and PRACE for computing resources

Hakala S., Havu V., Enkovaara J., Nieminen R. M. ”Parallel Electronic Structure Calculations Using Multiple Graphics Processing Units (GPUs)” In: Manninen, P., Öster, P. (eds.) PARA 2012. LNCS, vol. 7782, pp. 63--76. Springer, Heidelberg (2013)