Porting VASP to GPU using OpenACC

Martijn Marsman, Stefan Maintz, Andreas Hehn, Markus Wetzstein, and Georg Kresse

SC19, Denver, 19th Nov. 2019
The Vienna Ab-initio Simulation Package: VASP

Electronic structure from first principles:

\[ H\psi = E\psi \]

- Approximations:
  - Density Functional Theory (DFT)
  - Hartree-Fock/DFT-HF hybrid functionals
  - Random-Phase-Approximation (GW, ACFDT)
- 3500+ licensed academic and industrial groups worldwide.
- 10k+ publications in 2015 (Google Scholar), and rising.
- Developed in the group of Prof. G. Kresse at the University Vienna.
VASP: Computational Characteristics

**VASP does:**
- Lots of “smallish” FFTs: (e.g. 100×100×100)
- Matrix-Matrix multiplication (DGEMM and ZGEMM)
- Matrix diagonalization: $\mathcal{O}(N^3)$ ($N \approx \#-of-electrons$)
- All-2-all communication

**Using:**
- fftw3d (or fftw-wrappers to mkl-ffts)
- LAPACK BLAS3 (mkl, OpenBLAS)
- scaLAPACK (or ELPA)
- MPI (OpenMPI, impi, ...) [+ OpenMP]

VASP is pretty well characterized by the SPECfp2006 benchmark
VASP on GPU

• VASP has organically grown over more than 25 years (450k+ lines of Fortran 77/90/2003/2008/... code)
• Current release: some features were ported with CUDA C (DFT and hybrid functionals)
• Upcoming VASP6 release: re-ported to GPU using OpenACC
• The OpenACC port is more complete already than the CUDA port (Gamma-only version and support for reciprocal space projectors)
Porting VASP to GPU using OpenACC

- Compiler-directive based: single source, readability, maintainability, ...
- cuFFT, cuBLAS, cuSOLVER, CUDA aware MPI, NCCL
- Some dedicated kernel versions: e.g. batching FFTs, loop re-ordering
- “Manual” deep copies of derived types (nested and/or with pointer members)
- Multiple MPI ranks sharing a GPU (using MPS)
- Combine OpenACC and OpenMP (OpenMP threads driving asynchronous execution queues)
OpenACC directives

Data directives are designed to be optional

Manage Data Movement
Initiate Parallel Execution
Optimize Loop Mappings

#$acc data copyin(a,b) copyout(c)
...
#$acc parallel
#$acc loop gang vector
do i=1, n
  c(i) = a(i) + b(i)
...
enddo
#$acc end parallel
...
#$acc end data
Nested derived types

- OpenACC + Unified Memory not an option yet
- OpenACC 2.6 manual deep copy was key
- Requires large numbers of directives in some cases,
- ...but well encapsulated
- Future versions of OpenACC (3.0) will add true deep copy, require far fewer data directives
- When CUDA Unified Memory + HMM supports all classes of data, potential for a VASP port with no data directives at all
VASP on GPU benchmarks

CuC\_vdW

- C@Cu surface ($\Omega \cong 2800 \, \text{Å}^3$)
- 96 Cu + 2 C atoms (1064 e–)
- \text{vdW-DFT}
- \text{RMM-DIIIS}
- OpenACC port outperforms the previous CUDA port ...

• CPU: 2\times E5-2698 v4 @ 2.20 GHz: 40 physical cores
CUDA C vs. OpenACC port

- Full benchmark timings are interesting for time-to-solution, but are not an ‘apples-to-apples’ comparison between the CUDA and OpenACC versions:
- Amdahl’s law for non-GPU accelerated parts of code affects both implementations, but blurs differences
- Using OpenACC allowed to port additional kernels with minimal effort, has not been undertaken with CUDA version
- OpenACC version uses GPU-aware MPI to help more communication heavy parts, like orthonormalization
- OpenACC version was forked out of a more recent version of CPU code, while CUDA implementation is older

Can we find a fairer comparison? Let’s look at the RMM-DIIS algorithm …
Iterative diagonalization: RMM-DIIS (EDDRMM)

- EDDRMM part has comparable GPU-coverage for CUDA and OpenACC versions
- CUDA version uses kernel fusing, OpenACC version uses two refactored kernels
- minimal amount of MPI communication
- OpenACC version improves scaling with number of GPUs

EDDRMM section (silica_IFPEN), speedup over CPU

CPU: dual socket Broadwell E5-2698 v4, compiler Intel 17.0.1
GPU: 5.4.4 compiler Intel 17.0.1; dev_OpenACC compiler: PGI 18.3 (CUDA 9.1)
Orthonormalization

<table>
<thead>
<tr>
<th></th>
<th>CUDA C PORT</th>
<th>OPENACC PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Redistributing wavefunctions</strong></td>
<td>Host-only MPI (185 ms)</td>
<td>GPU-aware MPI (110 ms)</td>
</tr>
<tr>
<td><strong>Matrix-Matrix-Muls</strong></td>
<td>Streamed data (19 ms)</td>
<td>GPU local data (15 ms)</td>
</tr>
<tr>
<td><strong>Cholesky decomposition</strong></td>
<td>CPU-only (24 ms)</td>
<td>cuSolver (12 ms)</td>
</tr>
<tr>
<td><strong>Matrix-Matrix-Muls</strong></td>
<td>Default scheme (30 ms)</td>
<td>better blocking (13 ms)</td>
</tr>
<tr>
<td><strong>Redistributing wavefunctions</strong></td>
<td>Host-only MPI (185 ms)</td>
<td>GPU-aware MPI (80 ms)</td>
</tr>
</tbody>
</table>

- GPU-aware MPI benefits from NVLink latency and B/W
- Data remains on GPU, CUDA port streamed data for GEMMs
- Cholesky on CPU saves a (smaller) mem-transfer
- 180 ms (40%) are saved by GPU-aware MPI alone
- 33 ms (7.5%) by others
Si256_VJT_HSE06

- Vacancy in Si ($\Omega \approx 5200 \, \text{Å}^3$)
- 255 Si atoms (1020 e−)
- DFT/HF-hybrid functional
- Conjugate gradient
- Batched FFTs
- Explicit overlay of computation and communication using non-blocking collectives (NCCL)

**Speedup vs. CPU**

- **CPU**: 2X E5-2698 v4 @ 2.20 GHz: 40 physical cores
The OpenACC port: current limitations

• Some bottlenecks must be addressed: computation of the local potential is still done CPU-side.

• Not all features are ported yet: Currently we are porting the linear response solvers and cubic-scaling ACFDT (RPA total energies)

• Some features of VASP, e.g. cubic-scaling RPA, are very (very) memory intensive, and involve diagonalization of large complex matrices (> 100k × 100k): e.g. cusolverMgSyevd

• PGI compilers only
New Release: VASP6

- ... 
- Cubic-scaling RPA (ACFDT,GW)
- On-the-fly machine learned force-fields
- Electron-Phonon coupling
- MPI+OpenMP
- OpenACC port
- ...

- Caveat: the OpenACC port is still regarded to be “experimental” at this stage
- Actively gather feedback (from HPC sites)
- Intensive support effort

Special thanks to Stefan Maintz, Andreas Hehn, and Markus Wetzstein from NVIDIA and PGI!

And to Ani Anciaux-Sedrakian and Thomas Guignon at IFPEN!

And to you for listening!