AGENDA

- Short introduction to VASP
- Status of the CUDA port
- Prioritizing Use Cases for GPU Acceleration
- OpenACC in VASP
- Comparative Benchmarking
VASP OVERVIEW
Leading electronic structure program for solids, surfaces, and interfaces. Used to study chemical / physical properties, reactions paths, etc.

Atomic scale materials modeling from first principles

Simulate 1-1000s atoms (mostly solids/surfaces)

Liquids, crystals, magnetism, semiconductor/insulators, surfaces, catalysts

Solve many-body Schrödinger equation

\[ H\Psi = i\hbar \frac{\partial}{\partial t} \Psi \]

\[ H\Psi = E\Psi \]

\[ H = -\sum_i \frac{\hbar^2}{2m_e} \frac{\Delta_i}{\tau} + \sum_i \sum_{i'<i} \frac{1}{4\pi \varepsilon_0} \frac{e^2}{|\vec{r}_i - \vec{r}_{i'}|} - \sum_{i} \sum_{a} \frac{1}{4\pi \varepsilon_0} \frac{Z_a e^2}{|\vec{r}_i - \vec{R}_a|} V_{ext}(\vec{r}_i) \]

\[ V_{ext} \]
Density Functional Theory (DFT)  

\[ n(\vec{r}_1) = N_e \int \cdots \int |\Psi(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_{N_e})|^2 \, d\vec{r}_2 \cdots d\vec{r}_{N_e} \]

Enables solving sets of Kohn-Sham equations  

\[ \left( -\frac{1}{2} \Delta_n + v_{\text{eff}}(\vec{r}_n) \right) \psi_n(\vec{r}) = \epsilon_n^{KS} \psi_n(\vec{r}) \]

In a plane-wave based framework (PAW)  

\[ \psi_n(\vec{k}, \vec{r}) = \frac{1}{\sqrt{\Omega}} \sum_{\vec{G}} C_{\vec{G},n}(\vec{k}) e^{i(\vec{k}+\vec{G}) \cdot \vec{r}} \]

Hybrid DFT adding (parts of) of exact-exchange (Hartree-Fock) and even beyond!
Developed at G. Kresse’s group at University of Vienna (and external contributors)
Under development/refactoring for about 25 years
460K lines of Fortran 90, some FORTRAN 77
MPI parallel, OpenMP recently added for multicore
First endeavors on GPU acceleration date back to <2011 timeframe with CUDA C
VASP USERS / USAGE
12-25% of CPU cycles @ supercomputing centers

Academia
- Material Sciences
- Chemical Engineering
- Physics & Physical Chemistry

Companies
- Large semiconductor companies
- Oil & gas
- Chemicals - bulk or fine
- Materials - glass, rubber, ceramic, alloys, polymers and metals

Top 5 HPC Applications

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GROMACS</td>
</tr>
<tr>
<td>2</td>
<td>ANSYS - Fluent</td>
</tr>
<tr>
<td>3</td>
<td>Gaussian</td>
</tr>
<tr>
<td>4</td>
<td>VASP</td>
</tr>
<tr>
<td>5</td>
<td>NAMD</td>
</tr>
</tbody>
</table>

Source: Intersect360 2017 Site Census Mentions
AGENDA

Short introduction to VASP

Status of the CUDA port

Prioritizing Use Cases for GPU Acceleration

OpenACC in VASP

Comparative Benchmarking
Minimization algorithms to calculate electronic ground state:

- Blocked Davidson (ALGO = NORMAL & FAST) and RMM-DIIS (ALGO = VERYFAST & FAST)
- Parallelization over \( k \)-points
- Exact-exchange calculations

Earlier Work

- Speeding up plane-wave electronic-structure calculations using graphics-processing units, Maintz, Eck, Dronskowski
- VASP on a GPU: application to exact-exchange calculations of the stability of elemental boron, Hutchinson, Widom
- Accelerating VASP Electronic Structure Calculations Using Graphic Processing Units, Hacene, Anciaux-Sedrakian, Rozanska, Klahr, Guignon, Fleurat-Lessard
CUDA ACCELERATED VERSION OF VASP

Available today on NVIDIA Tesla GPUs

- All GPU acceleration with CUDA C
- Not all use cases are ported to GPUs
- Different source trees for Fortran vs CUDA C
- CPU code gets continuously updated and enhanced, required for various platforms
- Challenge to keep CUDA C sources up-to-date
- Long development cycles to port new solvers
INTEGRATION WITH VASP 5.4.4 (CUDA)

Original Routine - Fortran

GPU-accelerated Routine, Drop-in Replacement - Fortran

Custom Kernels and support code - CUDA-C

davidson.F

makefile switch

davidson_gpu.F

davidson.cu

cuda_helpers.h
cuda_helpers.cu...

Fortran-C Interface
CUDA Accelerated Version of VASP
Available today on NVIDIA Tesla GPUs

Source code duplication in CUDA C in VASP led to:

• increased maintenance cost
• improvements in CPU code need replication
• long development cycles to port new solvers

Explore OpenACC as an improvement for GPU acceleration
AGENDA

Short introduction to VASP
Status of the CUDA port
Prioritizing Use Cases for GPU Acceleration
OpenACC in VASP
Comparative Benchmarking
**CATEGORIES FOR METHODOLOGICAL OPTIONS**

This does not contain options influencing parallelization

<table>
<thead>
<tr>
<th>LEVELS OF THEORY</th>
<th>SOLVERS / MAIN ALGORITHM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard DFT</td>
<td>Davidson</td>
</tr>
<tr>
<td>Hybrid DFT (exact exchange)</td>
<td>RMM-DIIS</td>
</tr>
<tr>
<td>RPA (ACFDT, GW)</td>
<td>Davidson+RMM-DIIS</td>
</tr>
<tr>
<td>Bethe-Salpeter Equations (BSE)</td>
<td>Damped</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECTION SCHEME</th>
<th>EXECUTABLE FLAVORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real space</td>
<td>Standard variant</td>
</tr>
<tr>
<td>Real space (automatic optimization)</td>
<td>Gamma-point only (simplifications possible)</td>
</tr>
<tr>
<td>Reciprocal space</td>
<td>Non-collinear variant (more interactions)</td>
</tr>
</tbody>
</table>
EXAMPLE BENCHMARK: SILICA_IFPEN

- Standard DFT level of theory
  - Davidson solver
  - RMM-DIIS solver
  - Dav.+RMM-DIIS solver
  - Damped solver

- Hybrid DFT level of theory
  - Realspace proj. scheme
  - Reciprocal proj. scheme
  - Automatic proj. scheme

- RPA level of theory

- BSE level of theory

- Hybrid DFT level of theory
  - Gamma-point exec. flavor
  - Non-collinear exec. flavor

- Standard exec. flavor
- KPAR, NSIM, NCORE parallelization options

PGI
# PARALLELIZATION OPTIONS

<table>
<thead>
<tr>
<th><strong>KPAR</strong></th>
<th><strong>NSIM</strong></th>
<th><strong>NCORE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributes $k$-points</td>
<td>Blocking of orbitals</td>
<td>Distributes plane waves</td>
</tr>
<tr>
<td>Highest level parallelism, more or less embarrassingly parallel</td>
<td>No parallelism here, just grouping that can influence communication</td>
<td>Lowest level parallelism, needs parallel 3D FFT, inserts lots of MPI msgs</td>
</tr>
<tr>
<td>Can help for smaller systems</td>
<td>Ideal value different between CPU and GPU</td>
<td>Can help with load balancing problems</td>
</tr>
<tr>
<td>Not always possible</td>
<td>Needs to be tuned</td>
<td><em>No support in CUDA port</em></td>
</tr>
</tbody>
</table>
PARALLELIZATION LAYERS IN VASP

Wavefunction

Spins

$k$-points

Bands/Orbitals

Plane-wave coefficients

Physical quantities

$\psi$

$\uparrow$

$\downarrow$

$k_1, k_2, k_3, \ldots$

$n_1, n_2, \ldots$

$C_1, C_2, C_3, \ldots$

$\ldots$

$KPAR>1$

Default

NCORE>1

Parallelization feature
POSSIBLE USE CASES IN VASP
Each with a different computational profile

Supports a plethora of run-time options that define the workload (use case)

Those methodological options can be grouped into categories

Some, but not all are combinable

Combination determines if GPU acceleration is supported and also how well

Benchmarking the complete situation is tremendously complex
WHERE TO START
You cannot accelerate everything (at least soon)

Ideally every use case would be ported

Standard and Hybrid DFT alone give 72 use cases (ignoring parallelization options)!

Need to select most important use-cases

Selection should be based on real-world or supercomputing-facility scenarios
STATISTICS ON VASP USE CASES
NERSC job submission data 2014

Zhengji Zhao collected such data (INCAR) for 30397 VASP jobs over nearly 2 months

Data is based on job count, but has no timing information

Includes 130 unique users on Edison (CPU-only system)

No 1:1-mapping of parameters possible, expect large error margins

Data does not include calculation sizes, but it’s a great start
EMPLOYED MAIN ALGORITHMS AND LEVELS OF THEORY

Source: based on data provided by Zhengji Zhao, NERSC, 2014
SUMMARY
Where to start

Start with standard DFT, to accelerate most jobs
RMM-DIIS and Davidson nearly equally important, share a lot of routines anyway
Realspace projection more important for large setups
Gamma-point executable flavor as important as standard, so start with general one
Support as many parallelization options as possible (KPAR, NSIM, NCORE)
Communication is important, but scaling to large node counts is low priority
(62% fit into 4 nodes, 95% used ≤12 nodes)
VASP OPENACC PORTING PROJECT

feasibility study

• Can we get a working version, with today’s compilers, tools, HW?

• Decision to focus on one algorithm: RMM-DIIS

• Guidelines:
  • work out of existing CPU code
  • minimally invasive to CPU code

• Goals:
  • allow for performance comparison to CUDA port
  • assess maintainability, threshold for future porting efforts
AGENDA

Short introduction to VASP
Status of the CUDA port
Prioritizing Use Cases for GPU Acceleration
OpenACC in VASP
Comparative Benchmarking
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
DATA REGIONS IN OPENACC

intrinsic data types, static and dynamic

All static intrinsic data types of the programming language can appear in an OpenACC data directive, e.g. `real`, `complex`, `integer scalar` variables in Fortran.

Same for all fixed size arrays of intrinsic types, and dynamically allocated arrays of intrinsic type, e.g. `allocatable` and `pointer` variables in Fortran.

The compiler will know the base address and size (in C size needs to be specified in directive).

... So what about derived types? Two variants:

```
type stat_def
  integer a,b
  real c
end type stat_def

type(dyn_def)::var_stat
```

```
type dyn_def
  integer m
  real, allocatable, dimension(:) :: r
end type dyn_def

type(dyn_def)::var_dyn
```
The generic case is a main goal for a future OpenACC 3.0 specification. This is often referred to as full deep copy.

Until then, writing a manual deep copy is the best way to handle derived types:

- OpenACC 2.6 provides functionality (attach/detach)
- Static members in a derived type are handled by the compiler.
- The programmer manually copies every dynamic member in the derived type.
- AND ensures correct pointer attachment/detachment in the parent!

For more, see Daniel Tian’s talk, S8805, today 11:30, Grand Ballroom 220C!
**DERIVED TYPE**

manual copy

```fortran
type dyn_def
    integer m
    real, allocatable, dimension(:) :: r
end type dyn_def

type(dyn_def)::var_dyn

allocate(var_dyn%r(some_size))

!$acc enter data copyin(var_dyn, var_dyn%r)

!$acc exit data copyout(var_dyn%r, var_dyn)
```

1. allocates device memory for `var_dyn`
2. copies `m` (H2D)
3. copies host pointer for `var_dyn%r`
   - > device ptr invalid

1. allocates device memory for `r`
2. copies `r` (H2D)
3. attaches the device copy's pointer `var_dyn%r` to the device copy of `r`

1. copies `r` (D2H)
2. deallocates device memory for `r`
3. detaches `var_dyn%r` on the device, i.e. overwrites `r` with its host value!
   - > device ptr invalid

1. copies `m` (D2H)
2. copies `var_dyn%r` -> host pointer intact!
3. deallocates device memory for `var_dyn`
Important:

- the invalid pointers *must* not be dereferenced!
- use `update` directive only on members, never on a parent (it will overwrite the member pointers)!
- OpenACC 2.6 directives/API calls (`acc_attach/acc_detach`) are invoked internally by the data directives like `copyin(var_dyn%r)`, or must be invoked explicitly if parent information missing (e.g. `copyin(r)`, followed by `attach(var_dyn%r)`)

Typically, we need separate routines for `create`, `copyin`, `copyout`, `delete` directives.
OPENACC 2.6 MANUAL DEEPCOPY

VASP: managing one aggregate data structure

Derived Type 1
Members:
3 dynamic
1 derived type 2

Derived Type 2
Members:
21 dynamic
1 derived type 3
1 derived type 4

Derived Type 3
Members:
only static

Derived Type 4
Members:
8 dynamic
4 derived type 5
2 derived type 6

Derived Type 5
Members:
3 dynamic

Derived Type 6
Members:
8 dynamic

!$acc data copyin(array1)
call my_copyin(array1)

-> 12 lines of code
-> 48 lines of code
-> 26 lines of code
-> 8 lines of code
-> 13 lines of code

-> 107 lines of code just for COPYIN

Plus additional lines of code for COPYOUT, CREATE, UPDATE
MANUAL DEEPCOPY IN VASP

Necessary step to port VASP with OpenACC (currently).

Increased amount of code, but well encapsulated. Future versions (OpenACC 3.0) will work without need for manual deepcopy and hence with less code.

Unified memory (UM) not an option right now: not all data is dynamically allocated! Ongoing work to support all types of data in UM. HMM will improve the situation.

Manual deepcopy allowed to port RMM-DIIS
PORTING VASP WITH OPENACC

- Successfully ported the **RMM-DIIS** solver, plus some additional functionality
- Very little code refactoring was required
- Interfacing to cuFFT, cuBLAS and cuSolver math libraries
- Manual deepcopy was key
- OpenACC integrated into latest VASP development source version
- Public availability expected with next VASP release
AGENDA

Short introduction to VASP
Status of the CUDA port
Prioritizing Use Cases for GPU Acceleration
OpenACC in VASP

Comparative Benchmarking
VASP OPENACC PERFORMANCE
silica_IFPEN on V100

- Total elapsed time for entire benchmark
- 634 s on CPU, includes EDDRMM part, initialization, diagonalization, orthonormalization, etc.
- without MPS: same nr. of MPI ranks
- with MPS: tuning nr. of MPI ranks to optimize load on GPU (for CUDA and OpenACC versions individually)
- for more than 2 GPUs, OpenACC version with MPS is slower than without

CPU: dual socket Broadwell E5-2698 v4, compiler Intel 17.0.1
CUDA version: Intel 17.0.1, OpenACC version: PGI 18.1

NCORE>1 helps CPU to perform (less work, more MPI)
NCORE=1 same workload/parallelization as on GPU
VASP OPENACC PERFORMANCE

silica_IFPEN on V100

- NCORE=40: smaller workload on CPU than on GPU versions improves CPU performance
- compared against ‘tuned setup’ on CPU
- GPUs still outperform dual socket CPU node, in particular OpenACC version
- 97 seconds on Volta-based DGX1 with OpenACC

Full benchmark, speedup over CPU, with param. NCORE=40

<table>
<thead>
<tr>
<th>Speedup</th>
<th>Number of V100 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

CUDA no MPS
CUDA with MPS
OpenACC no MPS
OpenACC with MPS

CPU: dual socket Broadwell E5-2698 v4, compiler Intel 17.0.1
CUDA version: Intel 17.0.1, OpenACC version: PGI 18.1
VASP OPENACC PERFORMANCE

Kernel-level comparison for energy expectation values

<table>
<thead>
<tr>
<th></th>
<th>CUDA PORT</th>
<th>OPENACC PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernels per orbital</td>
<td>1 (69 µs)</td>
<td>8 (90 µs total)</td>
</tr>
<tr>
<td>kernels per NSIM-block (4 orbitals)</td>
<td>1 (137 µs)</td>
<td>0 (0 µs)</td>
</tr>
<tr>
<td>runtime per orbital</td>
<td>104 µs</td>
<td>90 µs</td>
</tr>
<tr>
<td>runtime per NSIM-block (4 orbitals)</td>
<td>413 µs</td>
<td>360 µs</td>
</tr>
</tbody>
</table>

- NSIM independent reductions
- Additional NSIM-fused kernel was probably better on older GPU generations
- Unfusing removes a synchronization point
- OpenACC adapts optimization to architecture with a flag
### VASP OPENACC PERFORMANCE

Section-level comparison for orthonormalization

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

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 more recent version of CPU code, while CUDA implementation is older

Can we find a subset which allows for fairer comparison? use EDDRMM
VASP OPENACC PERFORMANCE
silica_IFPEN on V100

- 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 nr. of GPUs

NCORE>1 helps CPU to perform (less work, more MPI)
NCORE=1 same workload/parallelization as on GPU
VASP OPENACC PERFORMANCE

silica_IFPEN on V100

• NCORE=40: smaller workload on CPU than on GPU versions improves CPU performance

• compared against ‘tuned setup’ on CPU

• GPUs still outperform dual socket CPU node, in particular OpenACC version

EDDRMM part, speedup over CPU, with param. NCORE=40

Speedup

Number of V100 GPUs

CPU: dual socket Broadwell E5-2698 v4, compiler Intel 17.0.1
CUDA version: Intel 17.0.1, OpenACC version: PGI 18.1
For VASP, OpenACC is *the* way forward for GPU acceleration. Performance is similar and in some cases better than CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts. We’re excited to collaborate with NVIDIA and PGI as an early adopter of CUDA Unified Memory.