STRATEGIES TO ACCELERATE VASP WITH GPUS USING OPENACC

Stefan Maintz, Dr. Markus Wetzstein

smaintz@nvidia.com; mwetzstein@nvidia.com

VASP USERS AND USAGE

12-25% of CPU cycles @ supercomputing centers

Academia

Material Sciences
Chemical Engineering
Physics & Physical Chemistry

Companies

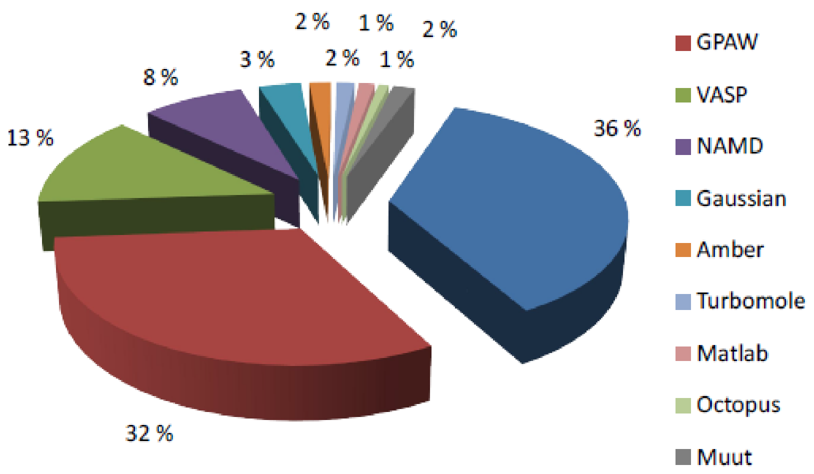
Large semiconductor companies
Oil & gas
Chemicals – bulk or fine
Materials – glass, rubber, ceramic, alloys, polymers and metals

Top 5 HPC Applications

Rank	Application
1	GROMACS
2	ANSYS - Fluent
3	Gaussian
4	VASP
5	NAMD

Source: Intersect360 2017 Site Census Mentions

CSC, Finland (2012)



VASP

The Vienna Ab initio Simulation Package

Developed by 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

COLLABORATION ON CUDA C PORT

Collaborators

ENS DE LYON · ifp Energies nouvelles · Carnegie Mellon University · U of Chicago · RWTH AACHEN UNIVERSITY · universität wien · OAK RIDGE National Laboratory · materials design · NVIDIA

CUDA Port Project Scope

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

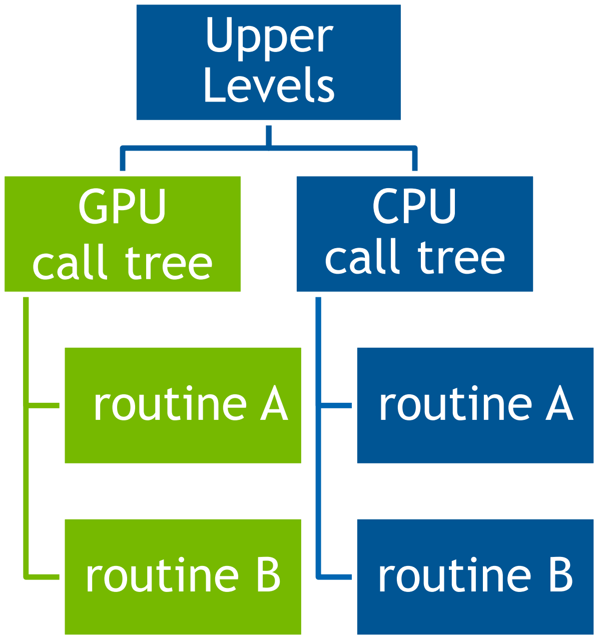
VASP on a GPU: application to exact-exchange calculations of the stability of elemental boron, Hutchinson, Widom

Accelerating VASP Electronic Structure Calculations Using Graphic Processing Units, Hacene, Anciaux-Sedrakian, Rozanska, Klahr, Guignon, Fleurat-Lessard

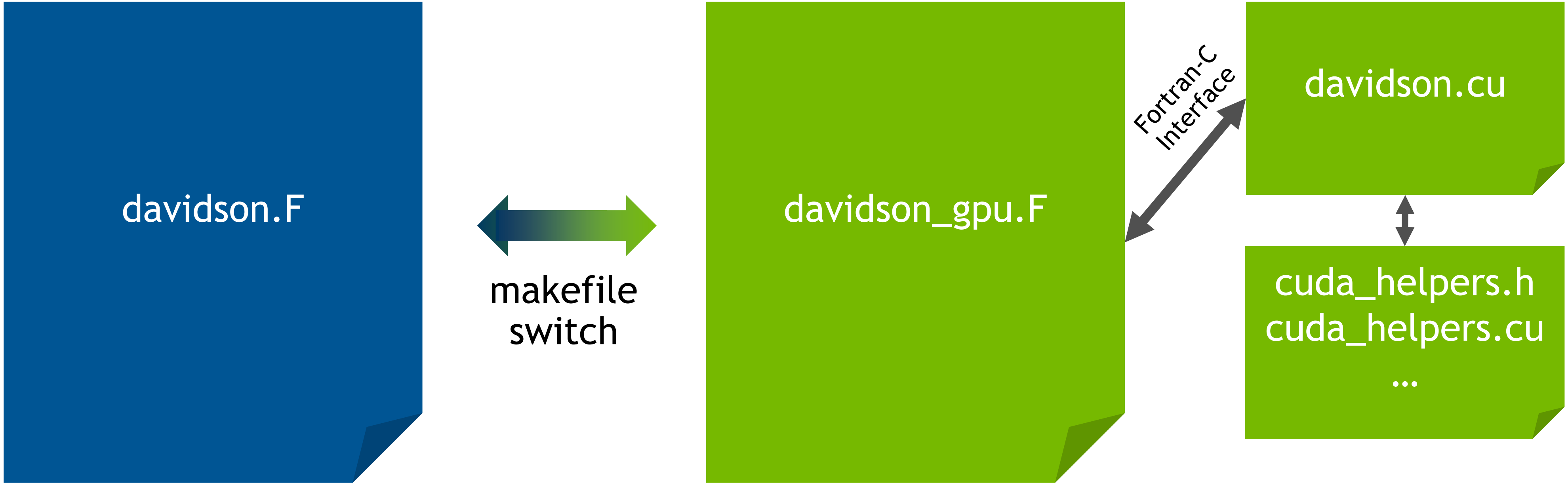
CUDA ACCELERATED VERSION OF VASP

Available today on NVIDIA Tesla GPUs

- All GPU acceleration done 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 C)



davidson.F

makefile switch

davidson_gpu.F

Fortran-C Interface

davidson.cu

cuda_helpers.h
cuda_helpers.cu
…

Original Routine
- Fortran

GPU-accelerated Routine, Drop-in Replacement
- Fortran

Custom Kernels and support code
- CUDA C

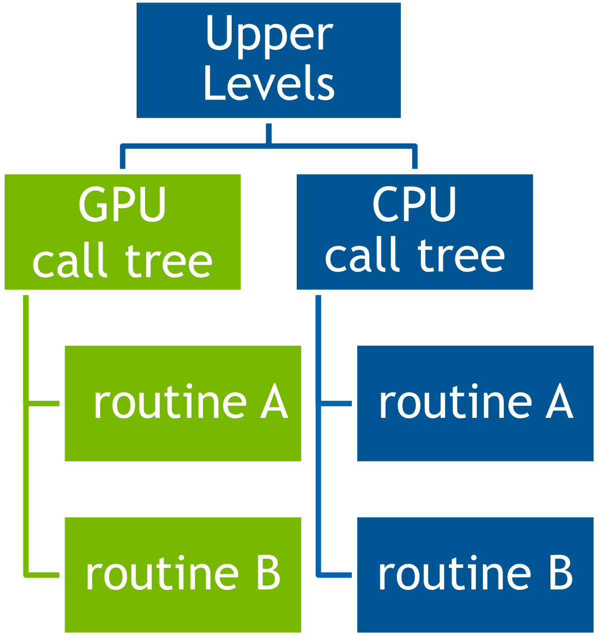
CUDA ACCELERATED VERSION OF VASP

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
- Only some of the plethora of solvers accelerated

→ **Explore OpenACC as an improvement for GPU acceleration**



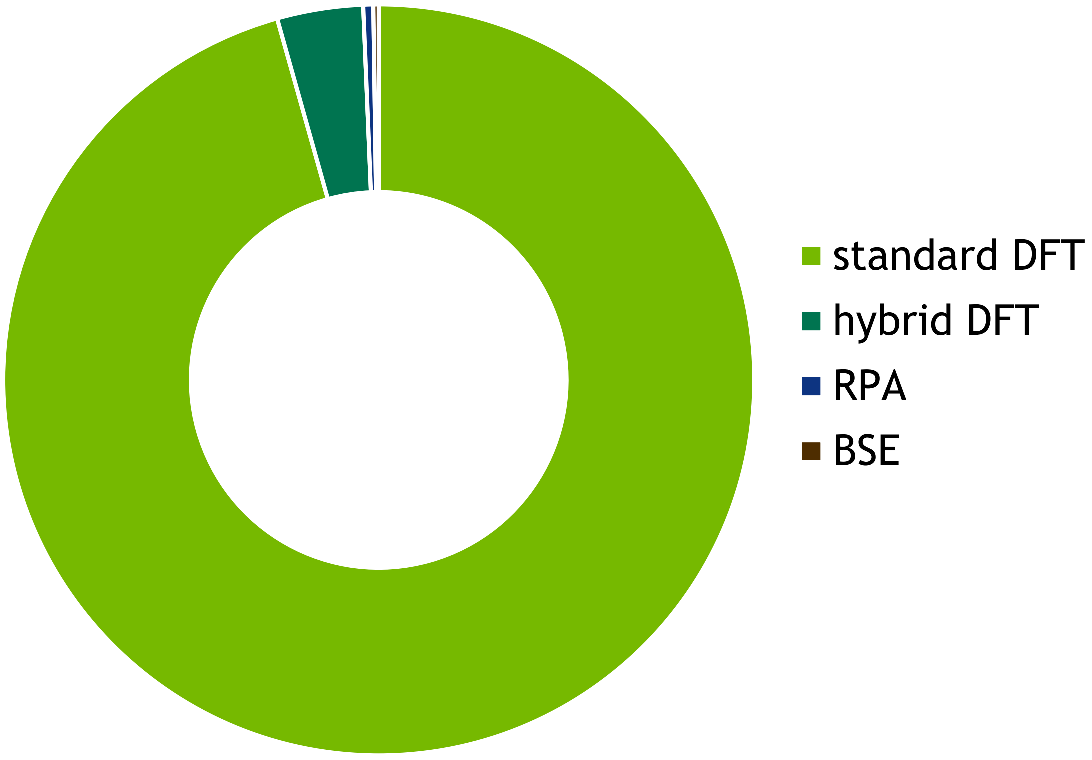
VASP OPENACC PORTING PROJECT

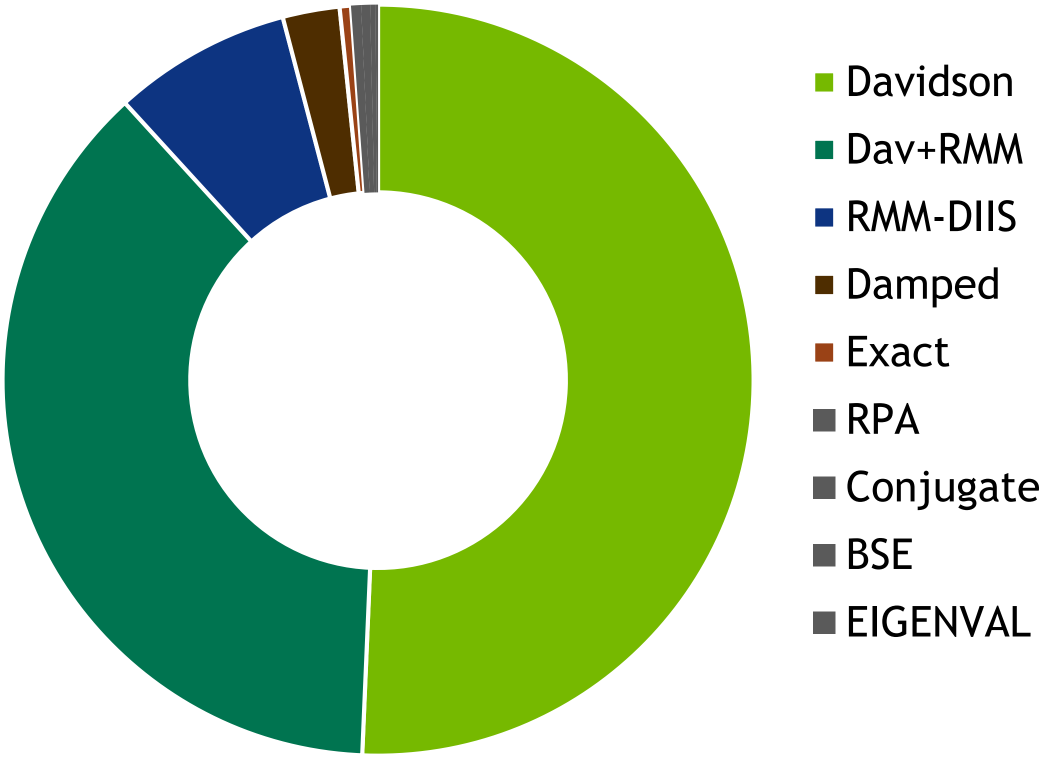
feasibility study

- Can we get a working version, with today’s compilers, tools and hardware?
- Focus on whole application performance for most important algorithms
- Guidelines:
  - work out of existing CPU code
  - minimally invasive to CPU code
- Goals:
  - performance competitive with CUDA port
  - assess maintainability, threshold for future porting efforts
  - source base fully portable to other compilers/platforms

EMPLOYED VASP FEATURES AT NERSC

Levels of theory and main algorithms w.r.t. job count

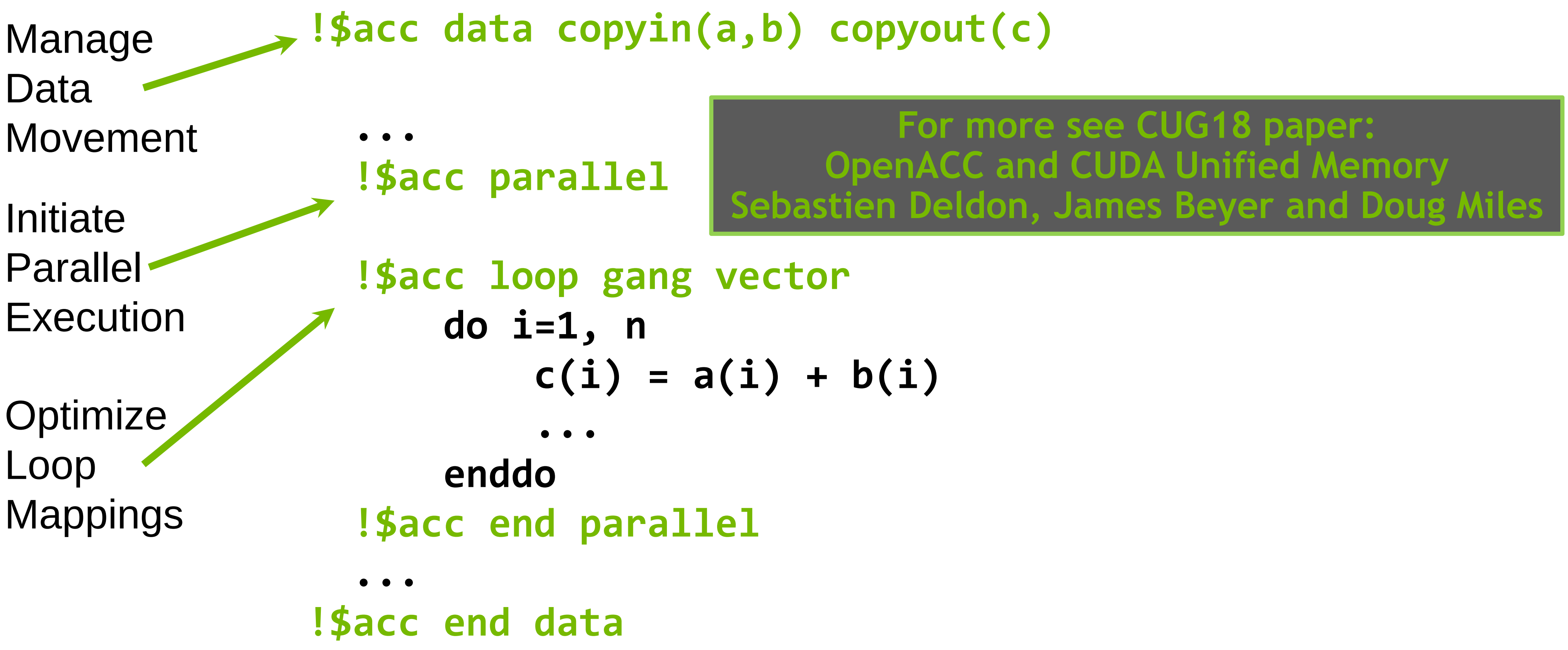




Source: based on data provided by Zhengji Zhao, NERSC, 2014

OPENACC DIRECTIVES

Data directives are designed to be optional



Manage Data Movement

Initiate Parallel Execution

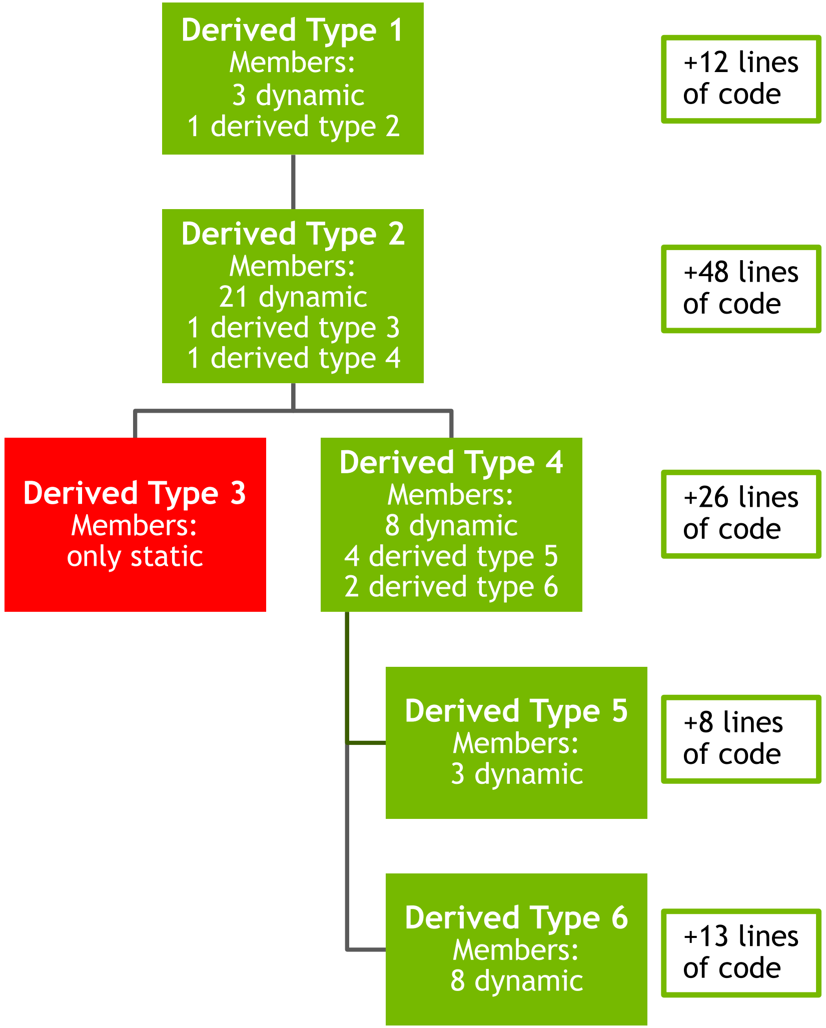
Optimize Loop Mappings

```
!$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
```

For more see CUG18 paper:
OpenACC and CUDA Unified Memory
Sebastien Deldon, James Beyer and Doug Miles

MANAGING VASP AGGREGATE DATA STRUCTURES

- OpenACC + Unified Memory not an option today, some aggregates have static members
- OpenACC 2.6 manual deep copy was key
- Requires large numbers of directives in some cases, but well encapsulated (107 lines for COPYIN)
- 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



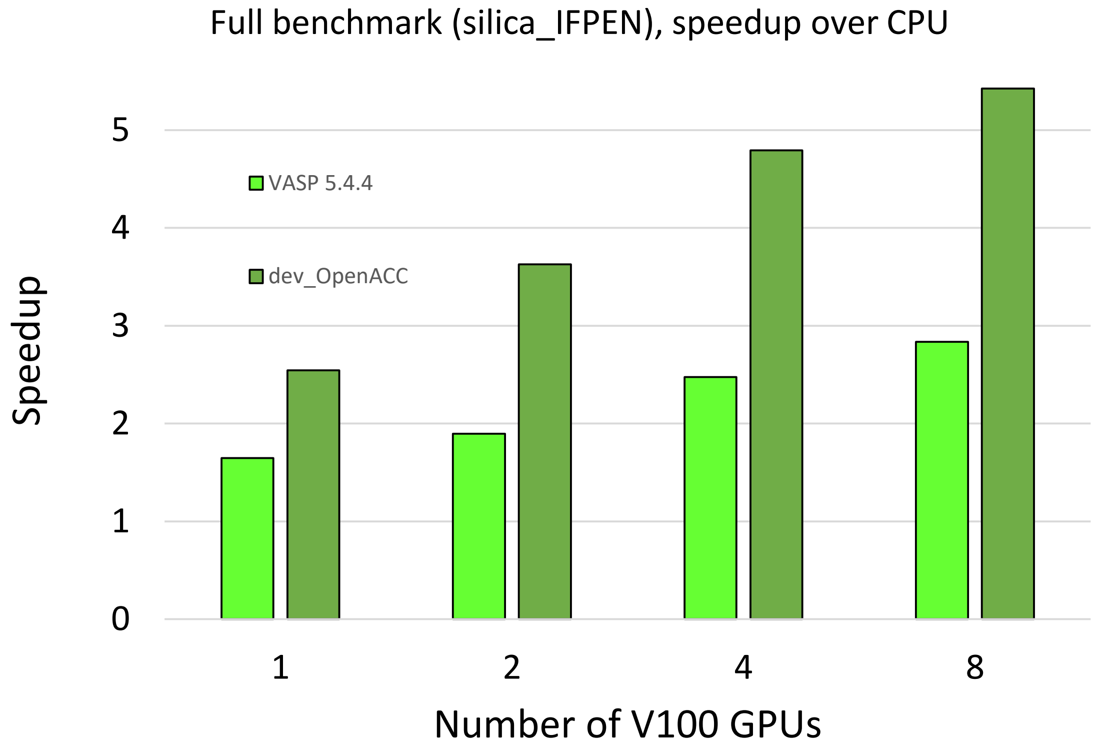
PORTING VASP WITH OPENACC

Intermediate results

- Successfully ported the **RMM-DIIS** and **blocked-Davidson** solvers, plus surrounding functionality
- Very little code refactoring was required
- Interfaced to cuFFT, cuBLAS and cuSolver math libraries
- Manual deep copy was key
- Ongoing integration of OpenACC into current VASP development source
- Public availability expected with next VASP release

VASP OPENACC PERFORMANCE

RMM-DIIS: silica_IFPEN on V100



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)

- Total elapsed time for entire benchmark
- NCORE=40 on CPU: smaller workload than on GPU versions improves CPU performance
- ‘tuned’ full run takes 465 s on CPU
- GPUs outperform dual socket CPU node, in particular OpenACC version
- 86 seconds on Volta-based DGX-1 with OpenACC
- OpenACC needs fewer ranks per GPU

VASP OPENACC PERFORMANCE

Kernel-level comparison for energy expectation values

	CUDA C PORT	OPENACC PORT
kernels per orbital	1 (69 µs)	8 (90 µs total)
kernels per NSIM-block (4 orbitals)	1 (137 µs)	0 (0 µs)
runtime per orbital	103 µs	90 µs
runtime per NSIM-block (4 orbitals)	413 µs	360 µs

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

VASP OPENACC PERFORMANCE

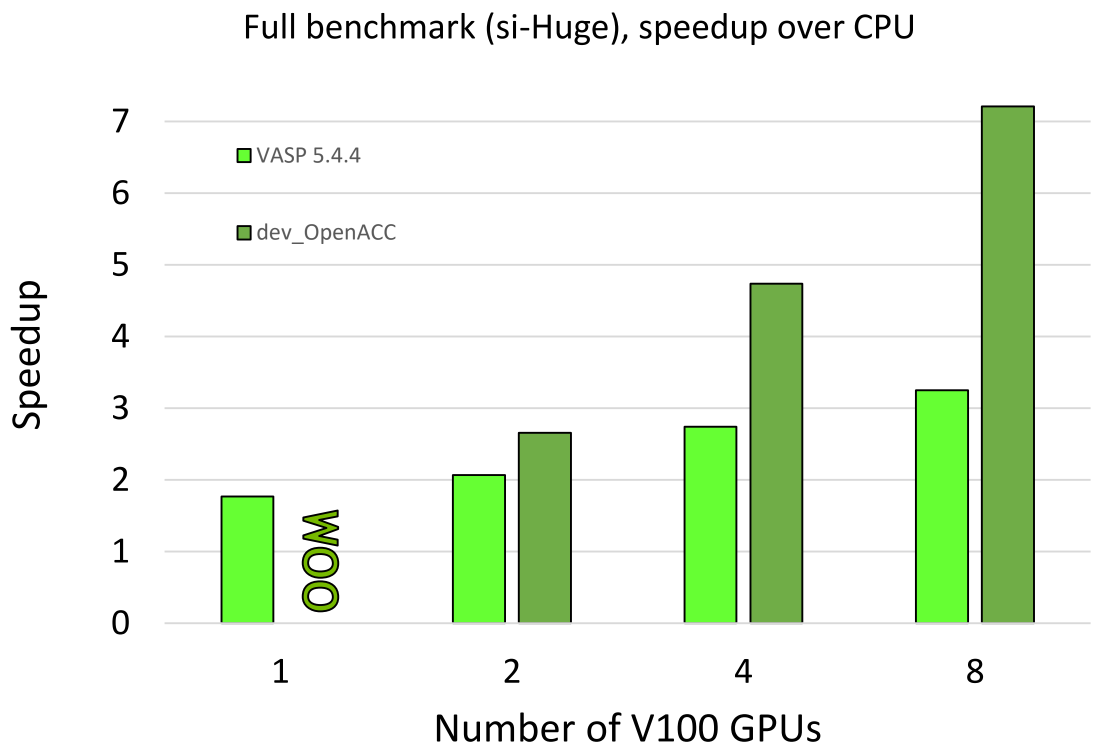
Section-level comparison for orthonormalization

	CUDA C PORT	OPENACC PORT
Redistributing wavefunctions	Host-only MPI (185 ms)	GPU-aware MPI (110 ms)
Matrix-Matrix-Muls	Streamed data (19 ms)	GPU local data (15 ms)
Cholesky decomposition	CPU-only (24 ms)	cuSolver (12 ms)
Matrix-Matrix-Muls	Default scheme (30 ms)	better blocking (13 ms)
Redistributing wavefunctions	Host-only MPI (185 ms)	GPU-aware MPI (80 ms)

- 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 OPENACC PERFORMANCE

Blocked-Davidson: si-Huge on V100



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)

- Total elapsed time for entire benchmark
- NCORE=40 on CPU: smaller workload than on GPU versions improves CPU performance
- ‘tuned’ full run takes 4852 s on CPU
- Drastically **improved scaling** for blocked-Davidson with openACC
- 673 s on Volta-based DGX-1 with OpenACC
- MPS not needed for OpenACC

VASP BENCHMARKS

Differences between CUDA and OpenACC versions

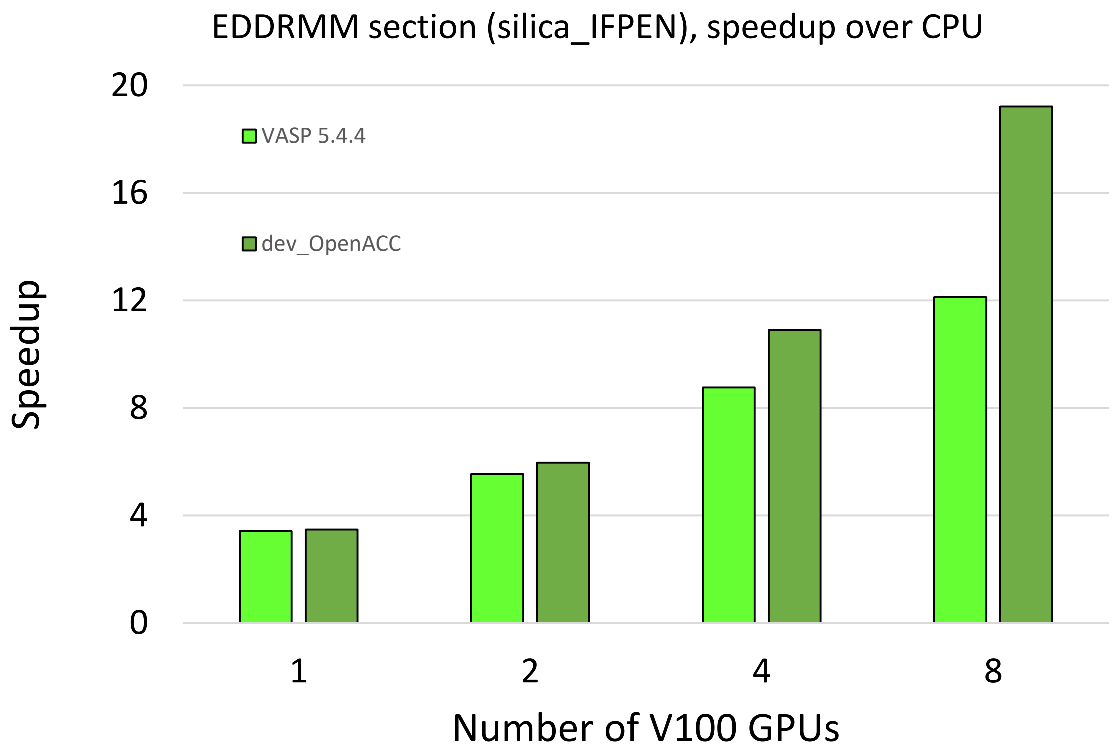
Full benchmark timings are interesting for time-to-solution, but are not an ‘apples-to-apples’ comparison between the CUDA and OpenACC versions:

- 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

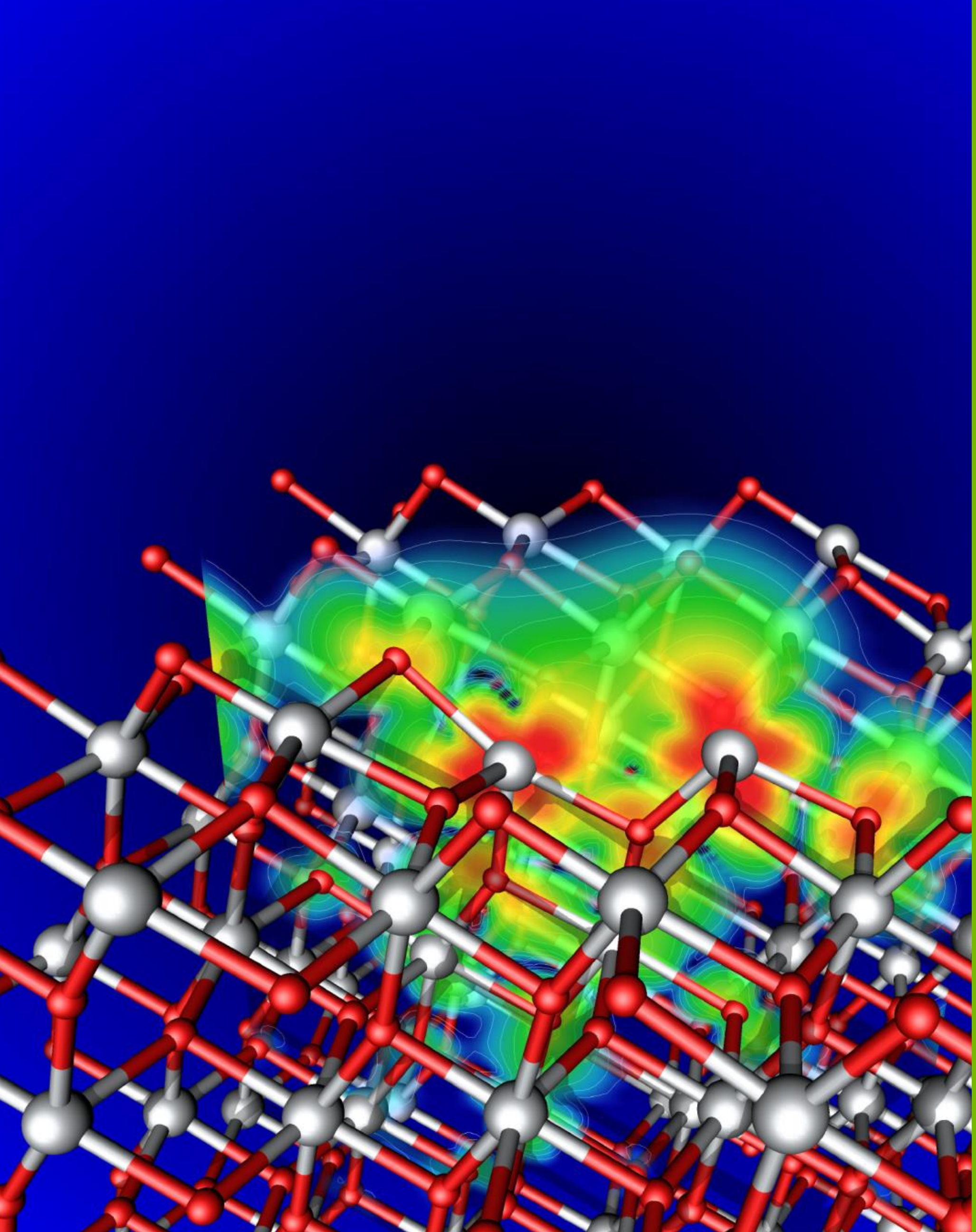
VASP OPENACC PERFORMANCE

EDDRMM section of silica_IFPEN on V100



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)

- EDDRMM takes 17% of total runtime
- benefits for expectation values included
- These high speedups are not the single aspect for overall improvement, but an important contribution
- OpenACC improves scaling yet again
- MPS always helps, but does not pay off in total time due to start-up overhead



VASP

The Vienna Ab Initio Simulation Package

Prof. Georg Kresse
Computational Materials Physics
University of Vienna

> “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.”

NVIDIA