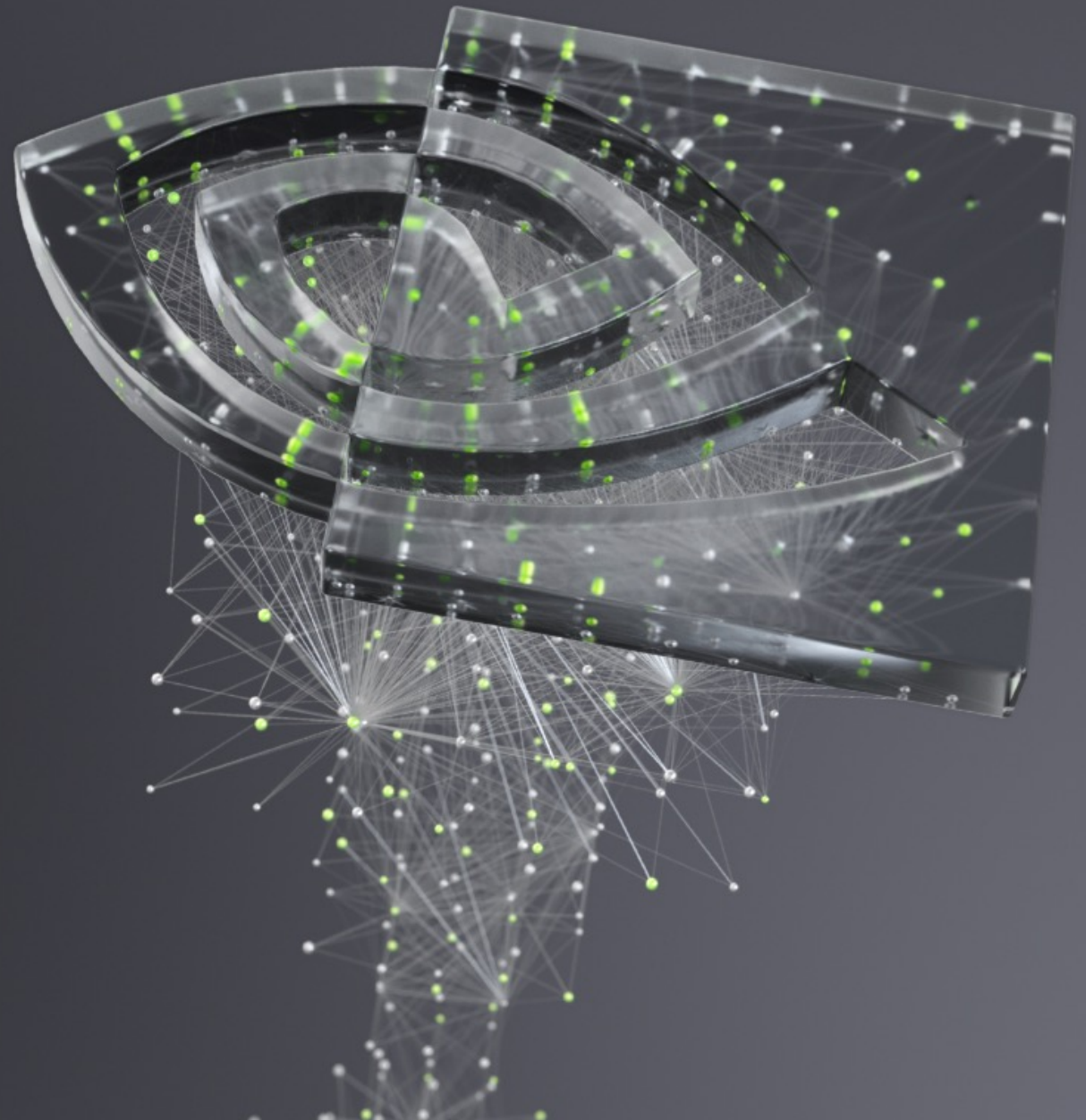


VASP 6

July 2022

Jonathan Lefman, Developer Relations

AGENDA

Introduction to VASP

Supported Accelerated features in VASP 6

Compiling and running VASP 6 with OpenACC

INTRODUCTION TO VASP

Scientific Background

Most widely used GPU-accelerated software for electronic structure of solids, surfaces, and interfaces

Generates

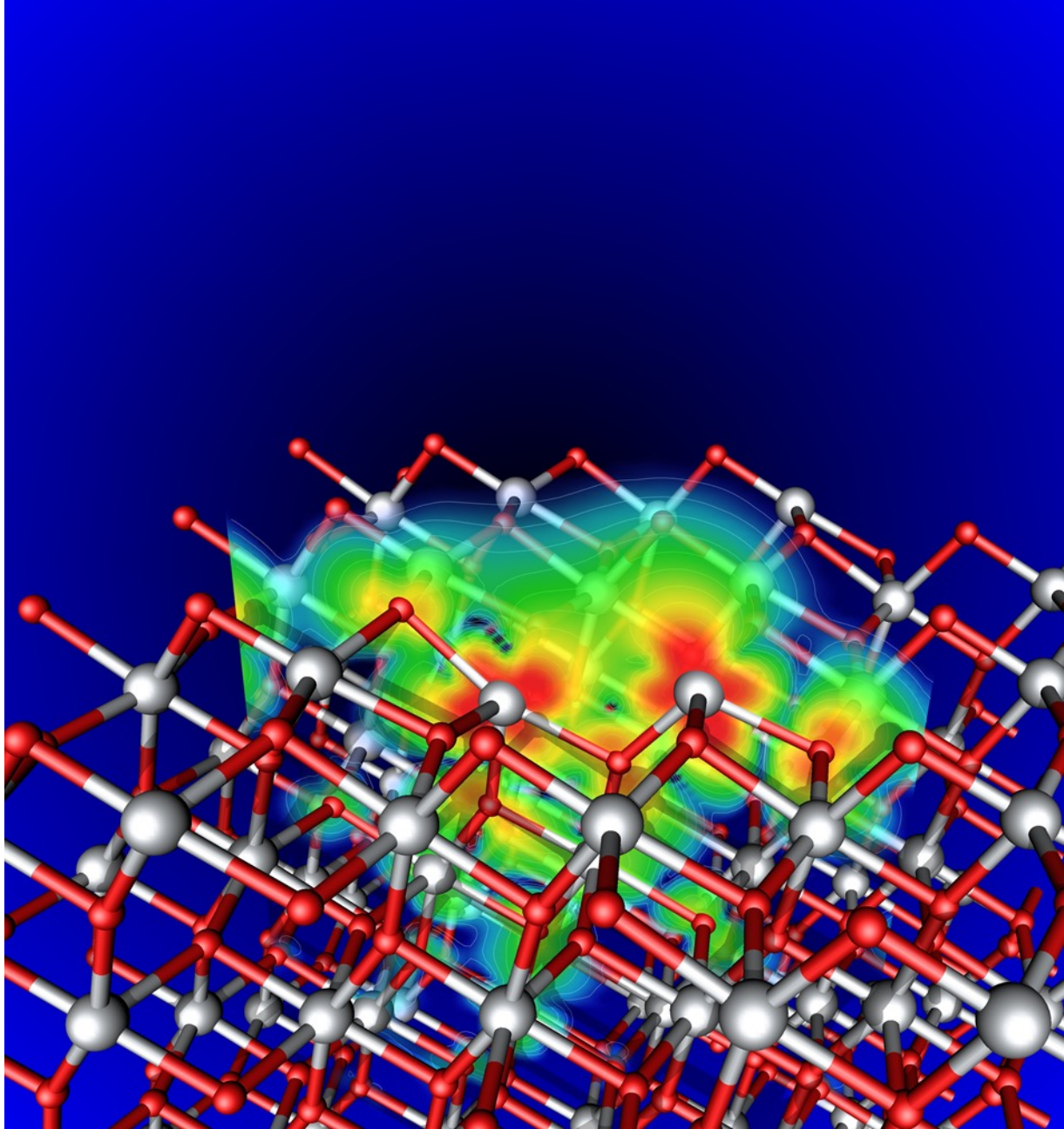
- Chemical and physical properties
- Reactions paths

Capabilities

- First principles scaled to 1000s of atoms
- Materials and properties - liquids, crystals, magnetism, semiconductors/insulators, surfaces, catalysts
- Solves many-body Schrödinger equation

Quantum-mechanical methods and solvers

- Density Functional Theory (DFT)
- Plane-wave based framework
- New implementations for hybrid DFT (HF exact exchange)



VASP SOFTWARE ORIGINS

Key facts

Developed by Kresse group at the University of Vienna and VASP Software GmbH

Development began >25 years ago

460K lines of Fortran code

MPI parallel, OpenMP recently added for multicore

GPU acceleration efforts started prior to 2011 with CUDA C

Computational characteristics

Many small Fast-Fourier-Transformations ~100^3

All-to-all communications

Matrix operations

- Matrix-Matrix multiplications
- Matrix-Vector multiplications
- Diagonalizations

Custom kernels

AGENDA

Introduction to VASP

Supported Accelerated features in VASP 6

Compiling and running VASP 6 with OpenACC

FEATURES AVAILABLE AND ACCELERATED FROM VASP 5

LEVELS OF THEORY

Standard DFT
Hybrid DFT
RPA (ACFDT, GW)
Bethe-Salpeter Equations (BSE)
…

SOLVERS / MAIN ALGORITHM

Davidson
RMM-DIIS
Davidson+RMM-DIIS
Direct optimizers (Damped, All)
Linear response

PROJECTION SCHEME

Real space
Reciprocal space

EXECUTABLE FLAVORS

Standard variant
Gamma-point simplification variant
Non-collinear spin variant

- Existing acceleration
- **New acceleration**
- Acceleration work in progress
- On acceleration roadmap

FEATURES AVAILABLE AND ACCELERATED IN VASP 6.1

LEVELS OF THEORY

Standard DFT
Hybrid DFT **(double buffered)**
Cubic-scaling RPA (ACFDT, GW)
Bethe-Salpeter Equations (BSE)

SOLVERS / MAIN ALGORITHM

Davidson (+**Adaptively Compressed Exch.**)
RMM-DIIS
Davidson+RMM-DIIS
Direct optimizers (Damped, All)
Linear response

PROJECTION SCHEME

Real space
Reciprocal space

EXECUTABLE FLAVORS

Standard variant
Gamma-point simplification variant
Non-collinear spin variant

- Existing acceleration
- **New acceleration**
- Acceleration work in progress
- On acceleration roadmap

FEATURES AVAILABLE AND ACCELERATED IN VASP 6.2

LEVELS OF THEORY

Standard DFT **(incl. meta-GGA, vdW-DFT)**
Hybrid DFT **(double buffered)**
Cubic-scaling RPA (ACFDT, GW)
Bethe-Salpeter Equations (BSE)

SOLVERS / MAIN ALGORITHM

Davidson **(+Adaptively Compressed Exch.)**
RMM-DIIS
Davidson+RMM-DIIS
Direct optimizers (Damped, All)
Linear response

PROJECTION SCHEME

Real space
Reciprocal space

EXECUTABLE FLAVORS

Standard variant
Gamma-point simplification variant
Non-collinear spin variant

- Existing acceleration
- **New acceleration**
- Acceleration work in progress
- On acceleration roadmap

FEATURES AVAILABLE AND ACCELERATED IN VASP 6.3

LEVELS OF THEORY

Standard DFT **(incl. meta-GGA, vdW-DFT)**
Hybrid DFT **(double buffered)**
Cubic-scaling RPA (ACFDT, GW)
Bethe-Salpeter Equations (BSE)

SOLVERS / MAIN ALGORITHM

Davidson **(+Adaptively Compressed Exch.)**
RMM-DIIS
Davidson+RMM-DIIS
Direct optimizers (Damped, All)
Linear response

PROJECTION SCHEME

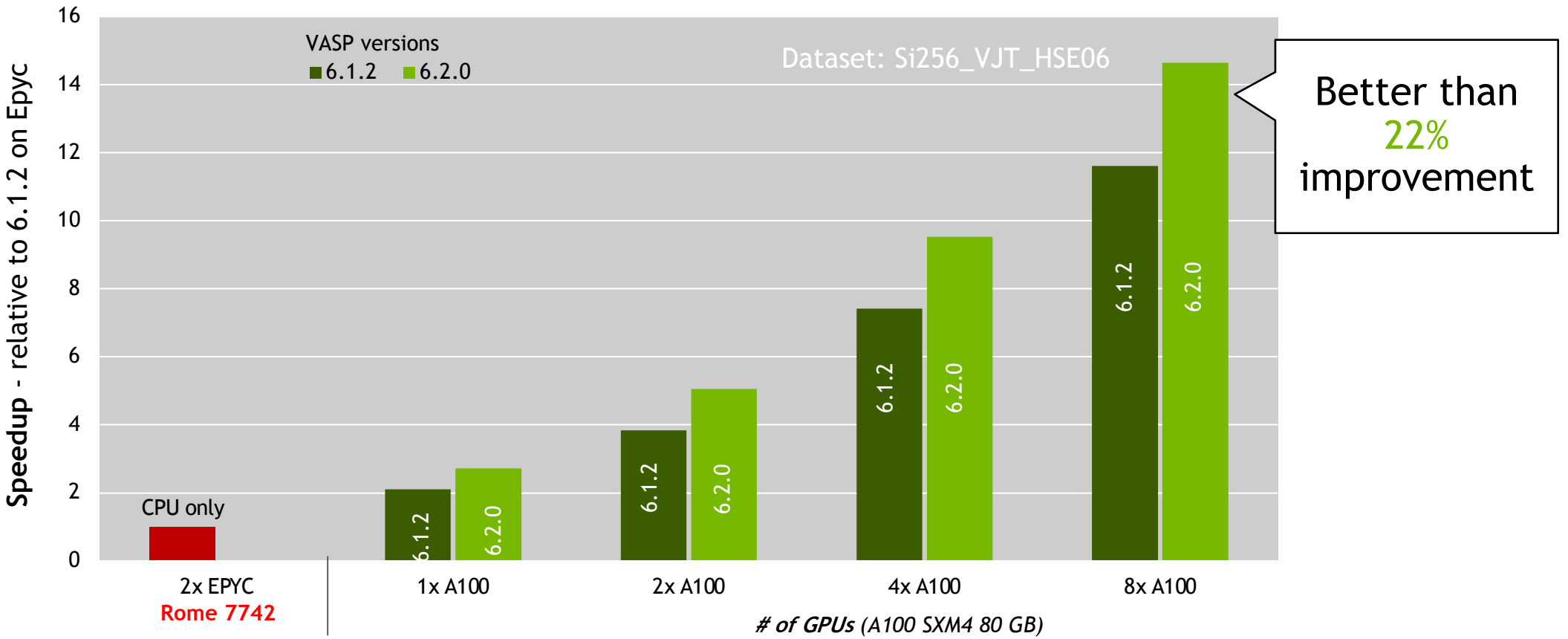
Real space
Reciprocal space

EXECUTABLE FLAVORS

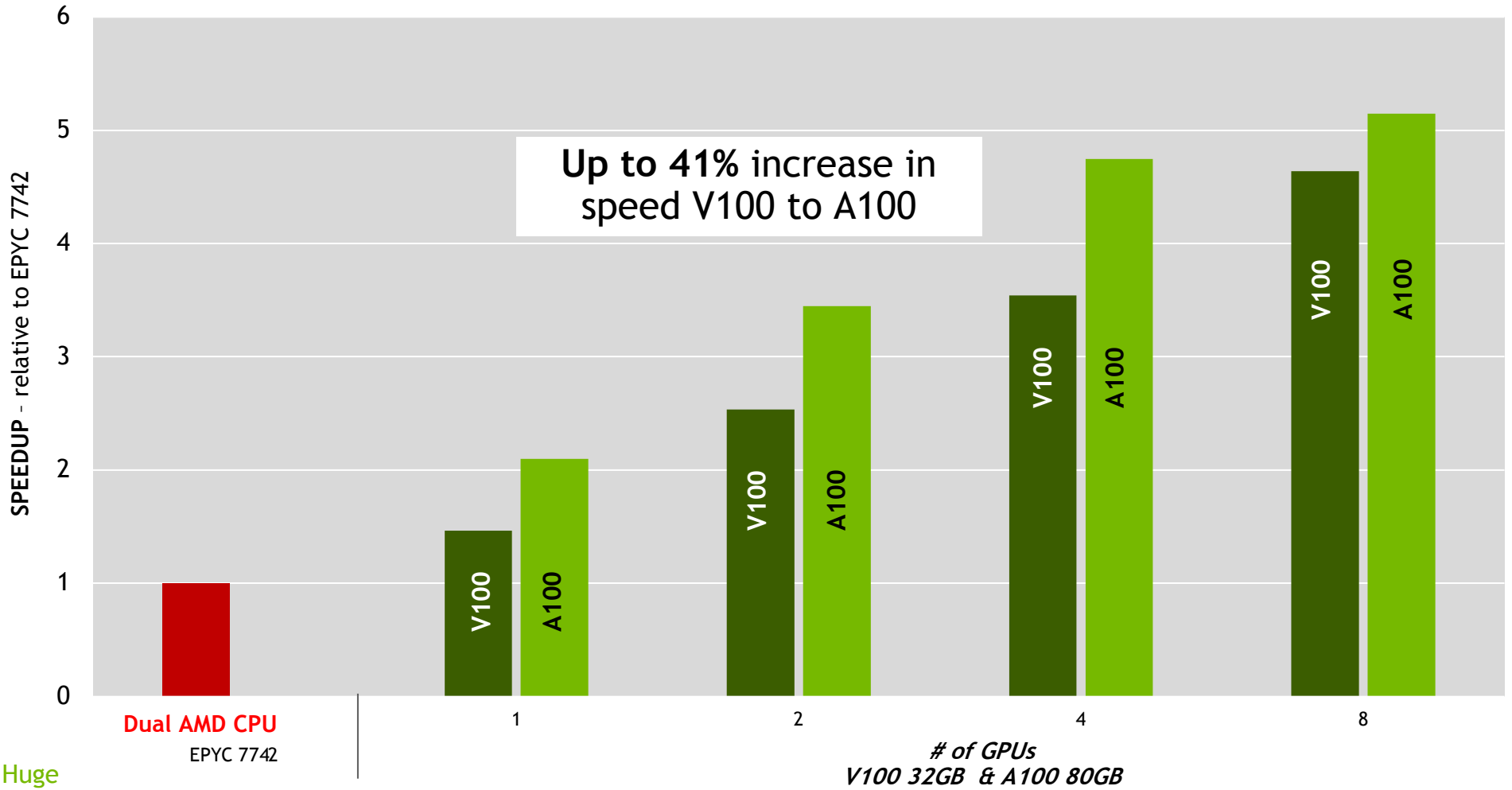
Standard variant
Gamma-point simplification variant
Non-collinear spin variant

- Existing acceleration
- **New acceleration**
- Acceleration work in progress
- On acceleration roadmap

VASP VERSION UPDATES BRING NEW ACCELERATION

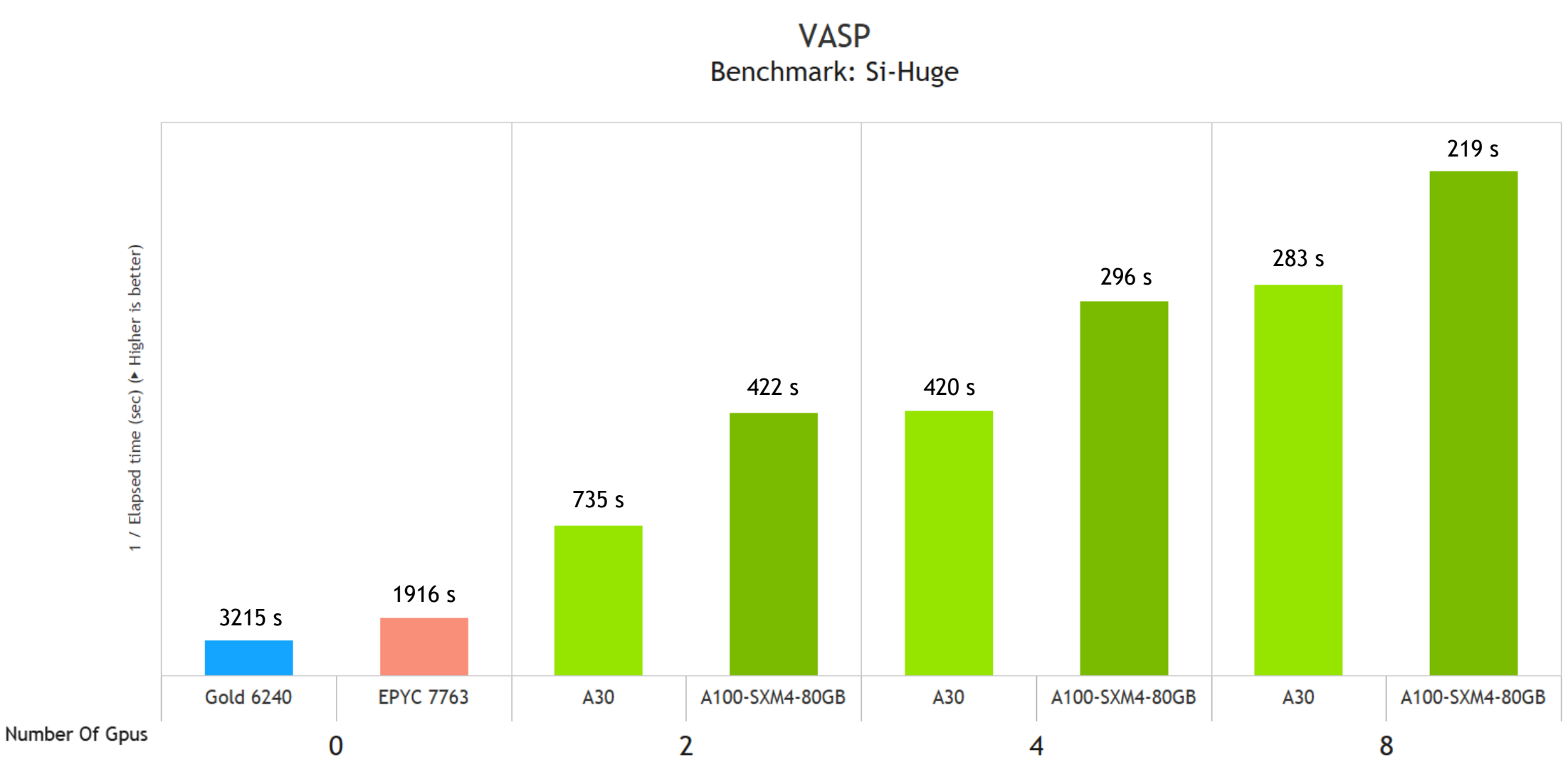


NEW NVIDIA GPU PLATFORMS - ADDITIONAL ACCELERATION

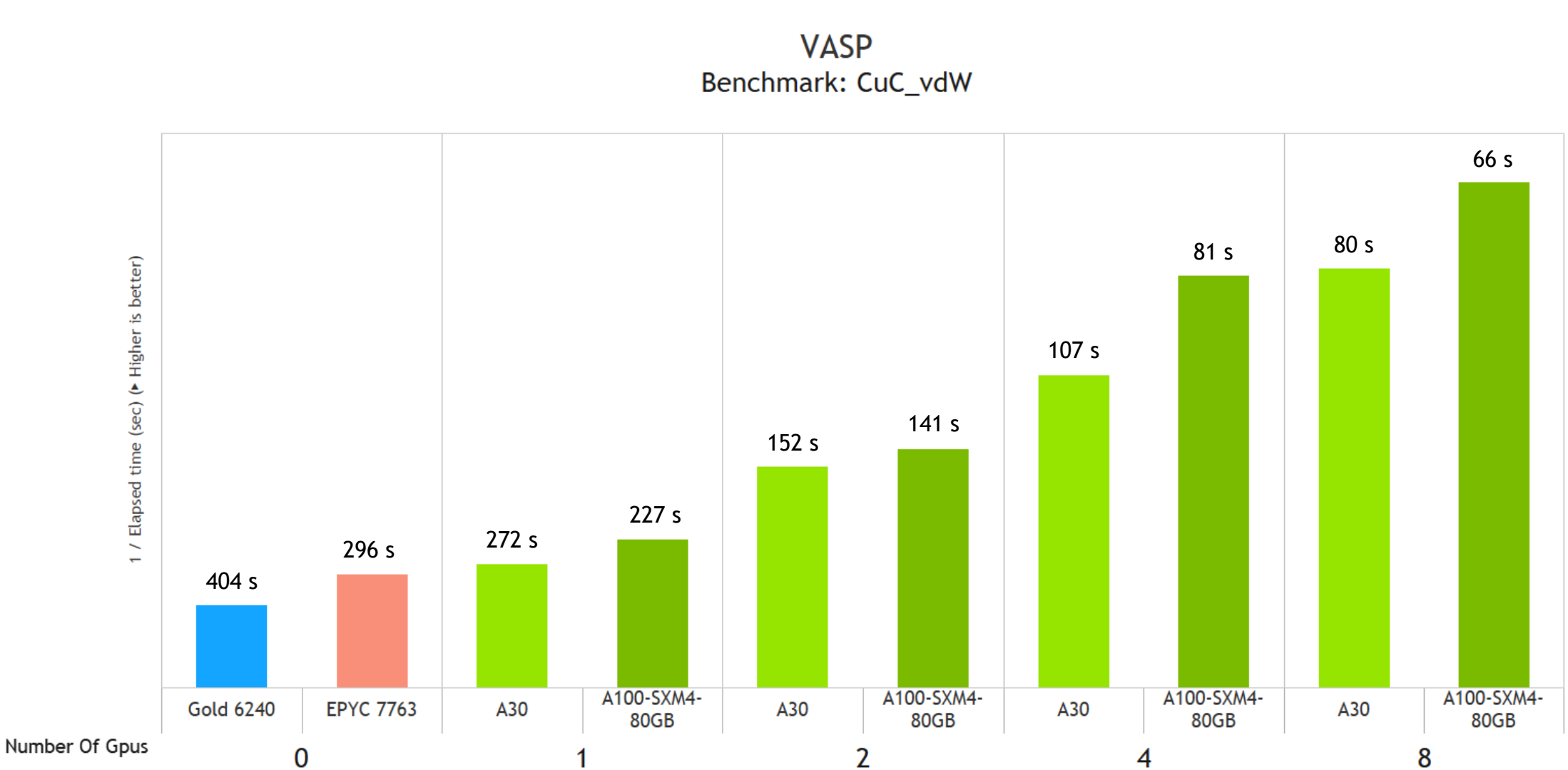


Dataset: Si-Huge

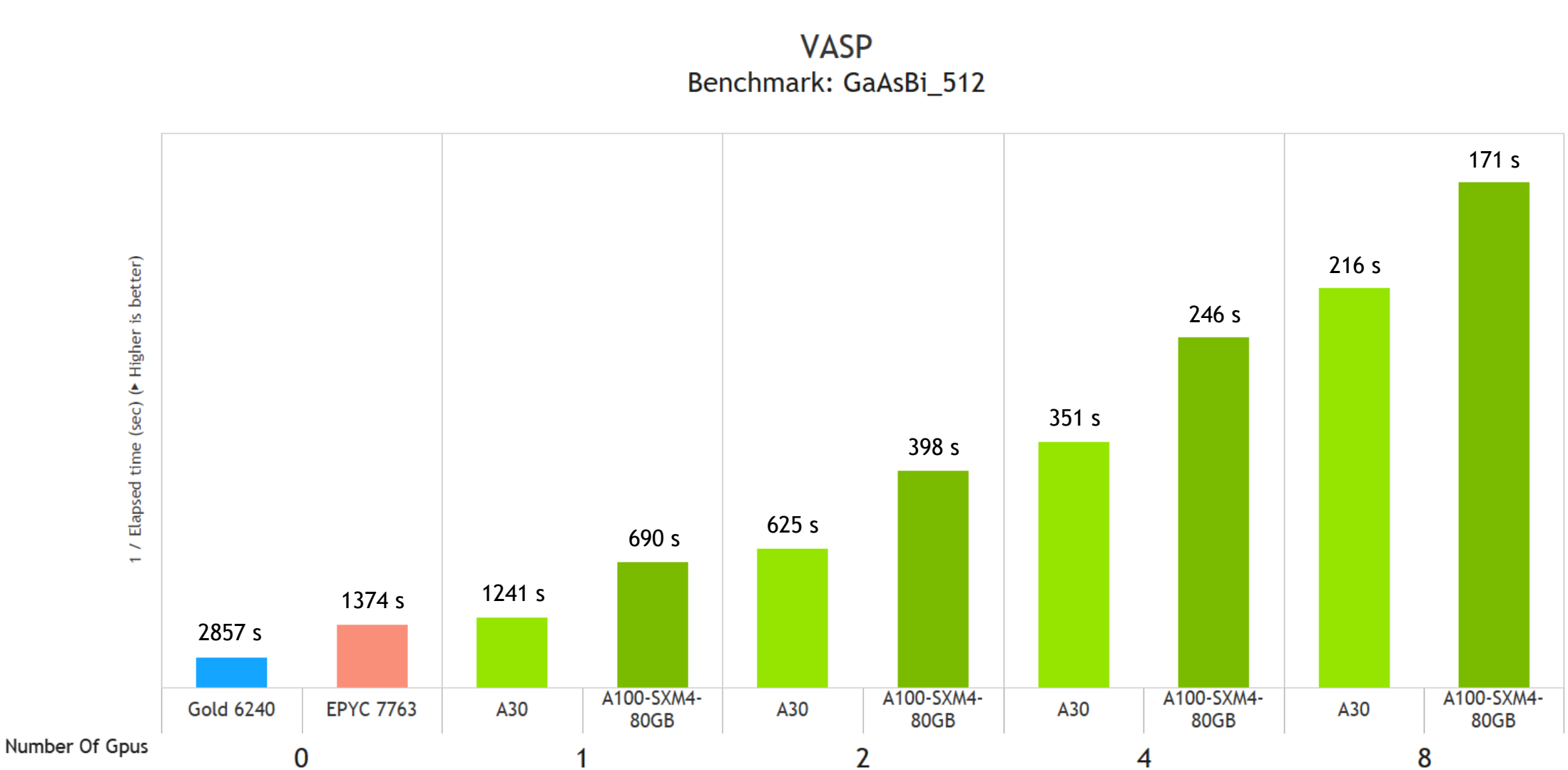
VASP 6.1.2


VASP
Benchmark: Si-Huge
1 / Elapsed time (sec) (▸ Higher is better)
3215 s
1916 s
735 s
422 s
420 s
296 s
283 s
219 s
Gold 6240
EPYC 7763
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
Number Of Gpus
0
2
4
8

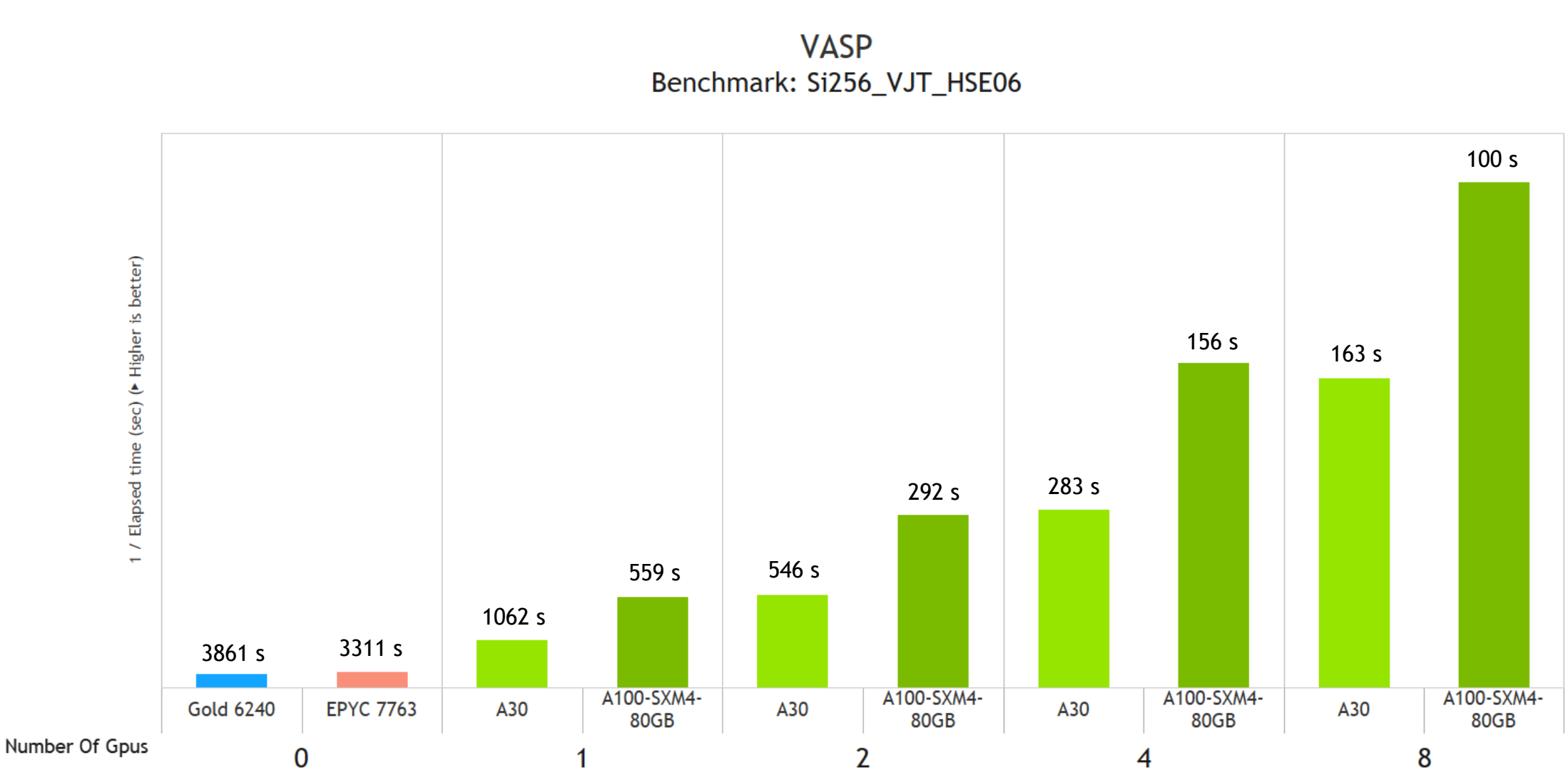
VASP 6.3.0


VASP
Benchmark: CuC_vdW
1 / Elapsed time (sec) (▸ Higher is better)
404 s
296 s
272 s
227 s
152 s
141 s
107 s
81 s
80 s
66 s
Gold 6240
EPYC 7763
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
Number Of Gpus
0
1
2
4
8

VASP 6.3.0

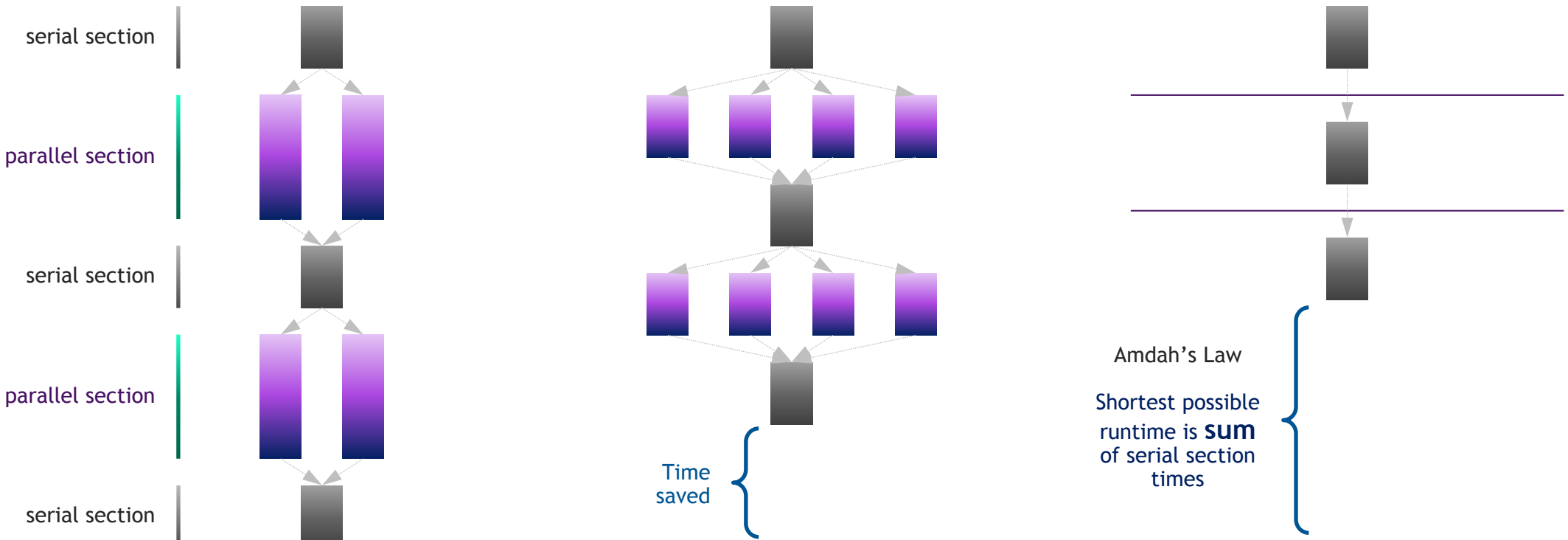

VASP
Benchmark: GaAsBi_512
1 / Elapsed time (sec) (▸ Higher is better)
2857 s
1374 s
1241 s
690 s
625 s
398 s
351 s
246 s
216 s
171 s
Gold 6240
EPYC 7763
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
Number Of Gpus
0
1
2
4
8

VASP 6.3.0


VASP
Benchmark: Si256_VJT_HSE06
1 / Elapsed time (sec) (▸ Higher is better)
3861 s
3311 s
1062 s
559 s
546 s
292 s
283 s
156 s
163 s
100 s
Gold 6240
EPYC 7763
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
A30
A100-SXM4-80GB
Number Of Gpus
0
1
2
4
8

VASP 6.3.0

AMDAHL'S LAW



Some Parallelism

Program time =
sum(serial times + parallel times)

Increased Parallelism

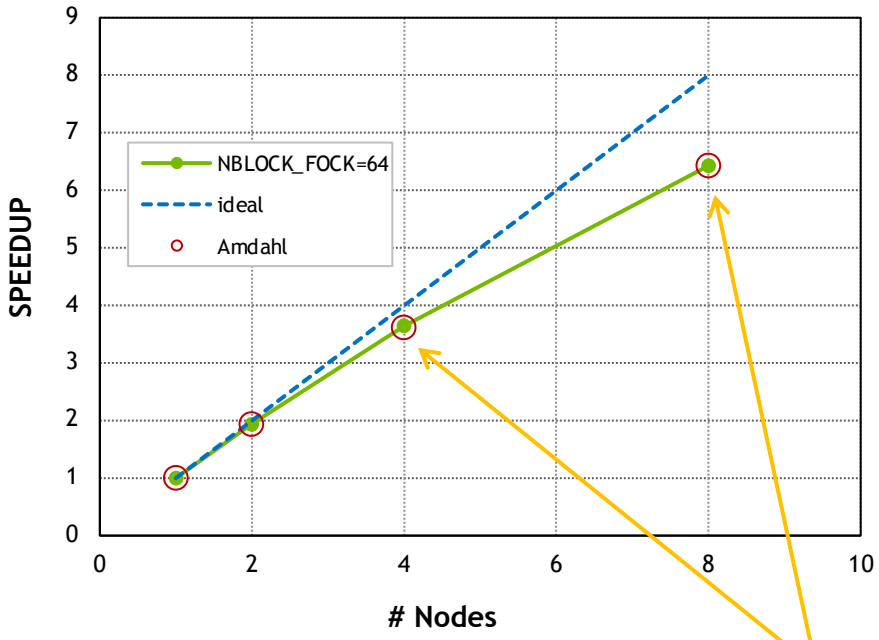
Parallel sections take less time
Serial sections take same time

Infinite Parallelism

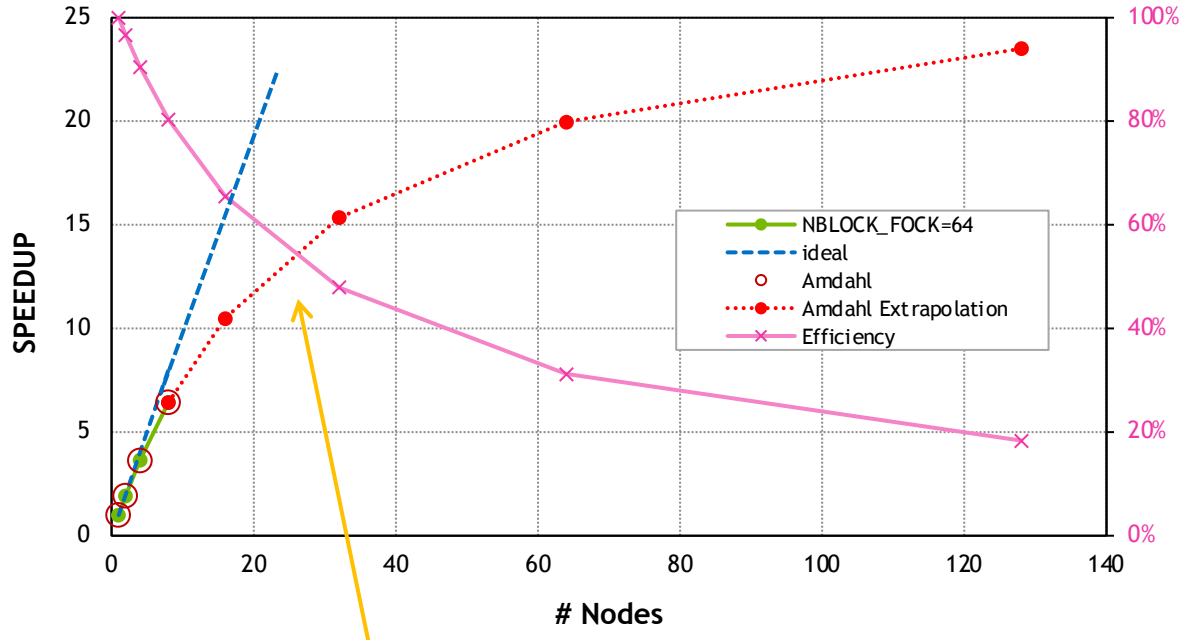
Parallel sections take no time
Serial sections take same time

MULTI NODE VASP – SCALING EXAMPLE

8 V100 GPUs nodes connected with HDR Infiniband



With an Amdahl's law numerical fit the approximation is quite good



50% Scaling efficiency occurs at 30 nodes in this example

Dataset: Si256_VJT_HSE06

AGENDA

Introduction to VASP

Supported Accelerated features in VASP 6

Compiling and running VASP 6 with OpenACC

WHY VASP DEVELOPERS CHOSE OPENACC

“ 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.”

Prof. Georg Kresse

CEO of VASP Software GmbH

Computational Materials Physics
University of Vienna

VASP 6 WITH OPENACC

Hardware requirements and recommendations

Works on all architectures supported by NVIDIA: x86, POWER and ARM

Ideally all GPUs connect with 16 PCIe lanes to the CPUs, otherwise use PCIe switches to share lanes with NICs

Best performance on NVIDIA GPUs with strong double precision (FP64) capabilities on A100. A30 is also an option. Volta generation V100 continues to provide excellent performance.

NVLink GPU-GPU-interconnects speed-up AllToAll communication

Dense GPU nodes preferred for throughput, fast network like Mellanox Infiniband is essential

VASP 6 WITH OPENACC

Software requirements and recommendations

NVIDIA HPC SDK 22.5, no cost and includes requirements

- OpenACC compiler (formerly PGI)
- NVIDIA CUDA Toolkit and Libraries: cuBLAS, cuFFT, cuSOLVER and NCCL
- CUDA-aware MPI (OpenMPI 3.1 without UCX recommended; otherwise use UCX ≥1.13)
- CPU math libraries: QD, OpenBLAS and ScaLAPACK

CPU math libraries: FFTW (compile with GCC)

VASP 6.3 WITH OPENACC

How to compile

Download the source code distribution with your license at www.vasp.at

Copy the according template for build-options file to the build dir:

```
$ cp arch/makefile.include.linux_nv_acc makefile.include
```

Adapt the `FC` and `FCL` lines of `makefile.include` to match your hard- and software environment. To compile for, e.g., A100 GPUs using CUDA 11.7:

```
FC  = mpif90 -acc -gpu=cc80,cuda11.7,noimplicitsections
FCL = mpif90 -acc -gpu=cc80,cuda11.7,noimplicitsections -c++libs
```

Adding excessive compute-capability levels will increase compile time, but not hurt runtime performance.

VASP 6.2 WITH OPENACC

How to compile

HPC SDK brings all dependencies besides `FFTW`, so you only need to adapt this variable in `makefile.include` to match the path on your system, or export them as an environment variable, e.g.:

```
export FFTW=/opt/fftw-3.3.9
```

It is recommended to build on the target system, otherwise add the appropriate `-tp` flag to the `FC`, `FCL`, `FC_LIB`, `CC_LIB` and `CXX_PARS` lines

Build the binaries accelerated using OpenACC:

```
make std gam ncl DEPS=1 -j 10
```

VASP 6 WITH OPENACC

How to run the accelerated version

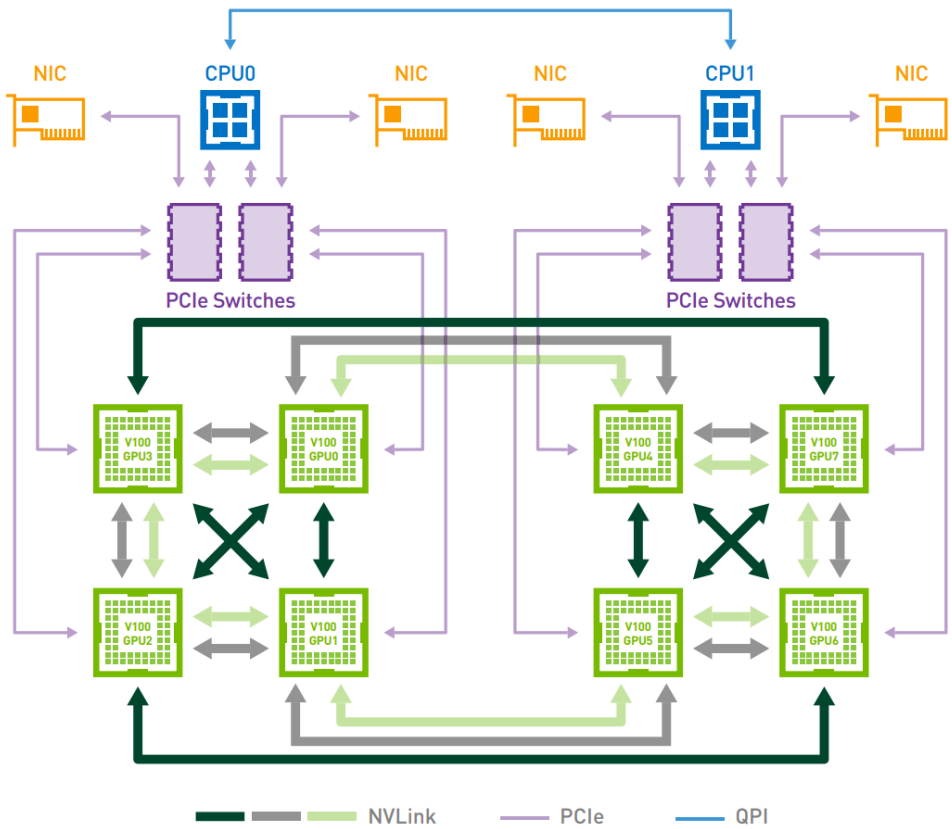
Run with 1 MPI rank per GPU
(requirement by NCCL library; don't use MPS as with the CUDA-C-port anymore)

Restrict libraries (like OpenBLAS or FFTW) to run with 1 thread per process only

VASP will select the GPUs automatically and use them in sequential order: Rank 0 → GPU 0, Rank 1 → GPU 1, …

Bind your processes to the CPU sockets with correct affinities to the GPUs and NICs. In doubt check with

```
$ nvidia-smi topo -m
```



VASP 6 WITH OPENACC

Binding your processes with correct affinities

Use a script like the following and run with `mpirun -n 8 runscript.sh vasp_std`

Example `runscript.sh` for DGX1:

```
#!/usr/bin/env bash
export UCX_RNDV_THRESH=1024
export UCX_MEMTYPE_CACHE=n
export OMP_NUM_THREADS=1
NICS=(mlx5_0 mlx5_0 mlx5_1 mlx5_1 mlx5_2 mlx5_2 mlx5_3 mlx5_3)
CPUS=(0 0 0 0 1 1 1 1)
lrank=$OMPI_COMM_WORLD_LOCAL_RANK
export UCX_NET_DEVICES=${NICS[$lrank]}:1
export OMPI_MCA_btl_openib_if_include=${NICS[$lrank]}
numactl --cpunodebind=${CPUS[$lrank]} --membind=${CPUS[$lrank]} $@
```

VASP 6 WITH OPENACC

Tune your VASP jobs

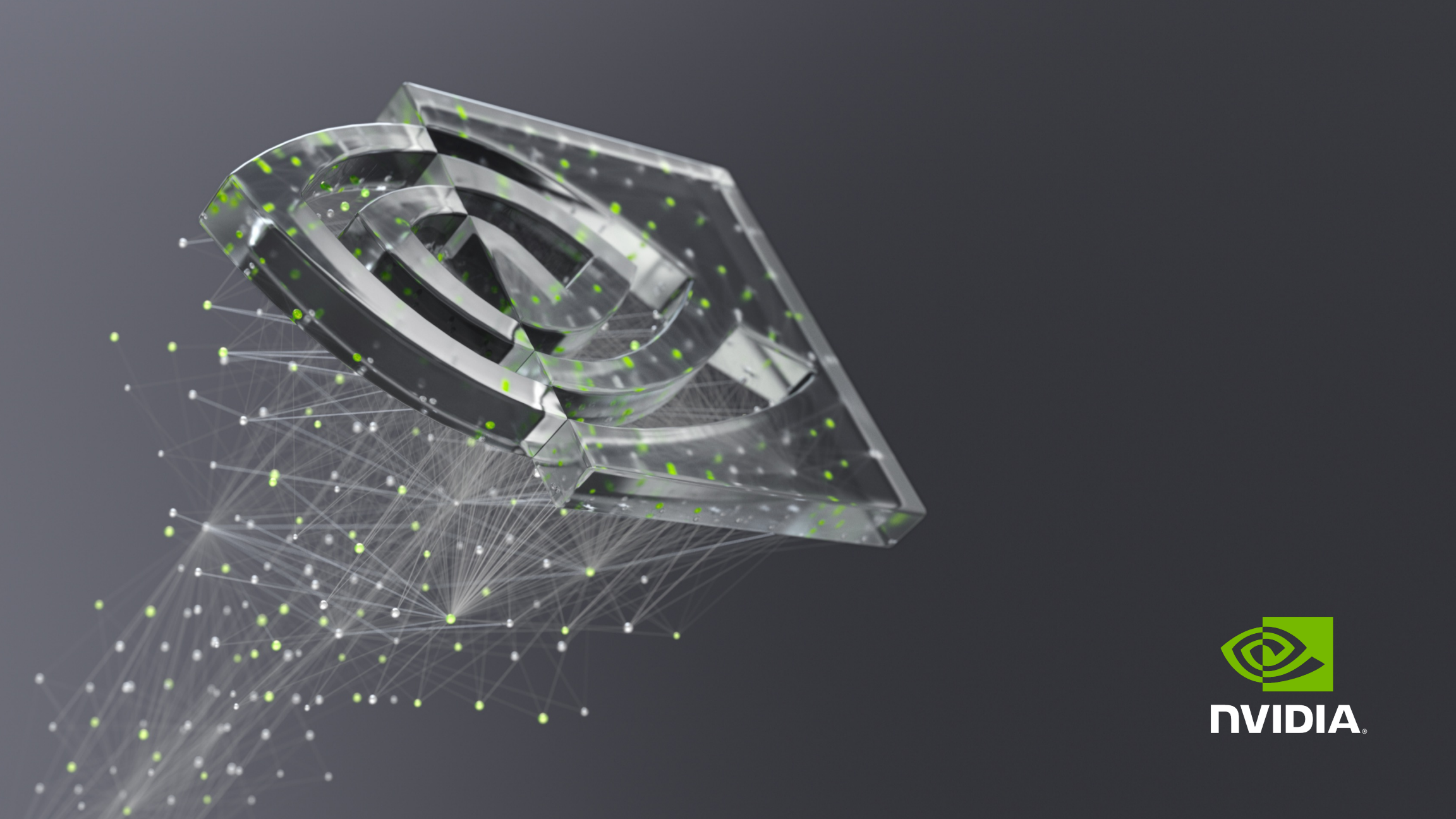
Use `vasp_gam` binary when possible! Saves memory and faster execution

`INCAR`: Remove `NPAR` and set `NCORE=1`: VASP 6.1.2 will do this for you internally, but better be safe.

`INCAR`: For `vasp_std` and `vasp_ncl` jobs, set `KPAR`: Use a value that evenly divides the number of k-points (`grep NKPTS OUTCAR`) by the number of GPUs. The higher the better. Much improved performance for increased memory usage.

`INCAR`: For standard and hybrid DFT jobs tune `NSIM` parameter. Test powers of 2 until it uses too much memory or performance stops improving.

`INCAR`: For hybrid DFT jobs, tune `NBLOCK_FOCK` parameter. Use a value that evenly divides the number of bands/orbitals (grep NBANDS OUTCAR) by the number of GPUs. As a rule of thumb, the higher the better.


NVIDIA