VASP - Best Practices Seminar

Weine Olovsson
National Supercomputer Centre (NSC), Linköping University
NAISS training, online 3rd Oct 2025

LINKÖPINGS UNIVERSITET

NAISS

3. Running & Performance

LINKÖPINGS UNIVERSITET

NAISS

Running & Performance

- General considerations
- Focus on practical aspects of running VASP
  …at specific supercomputer clusters
- Influential parameters, NPAR/NCORE, ALGO, NSIM, KPAR, …
- Memory usage
- Benchmarks, examples
- Common problems

Parallel calculations

Examples for different NAISS HPC clusters

Computation - considerations

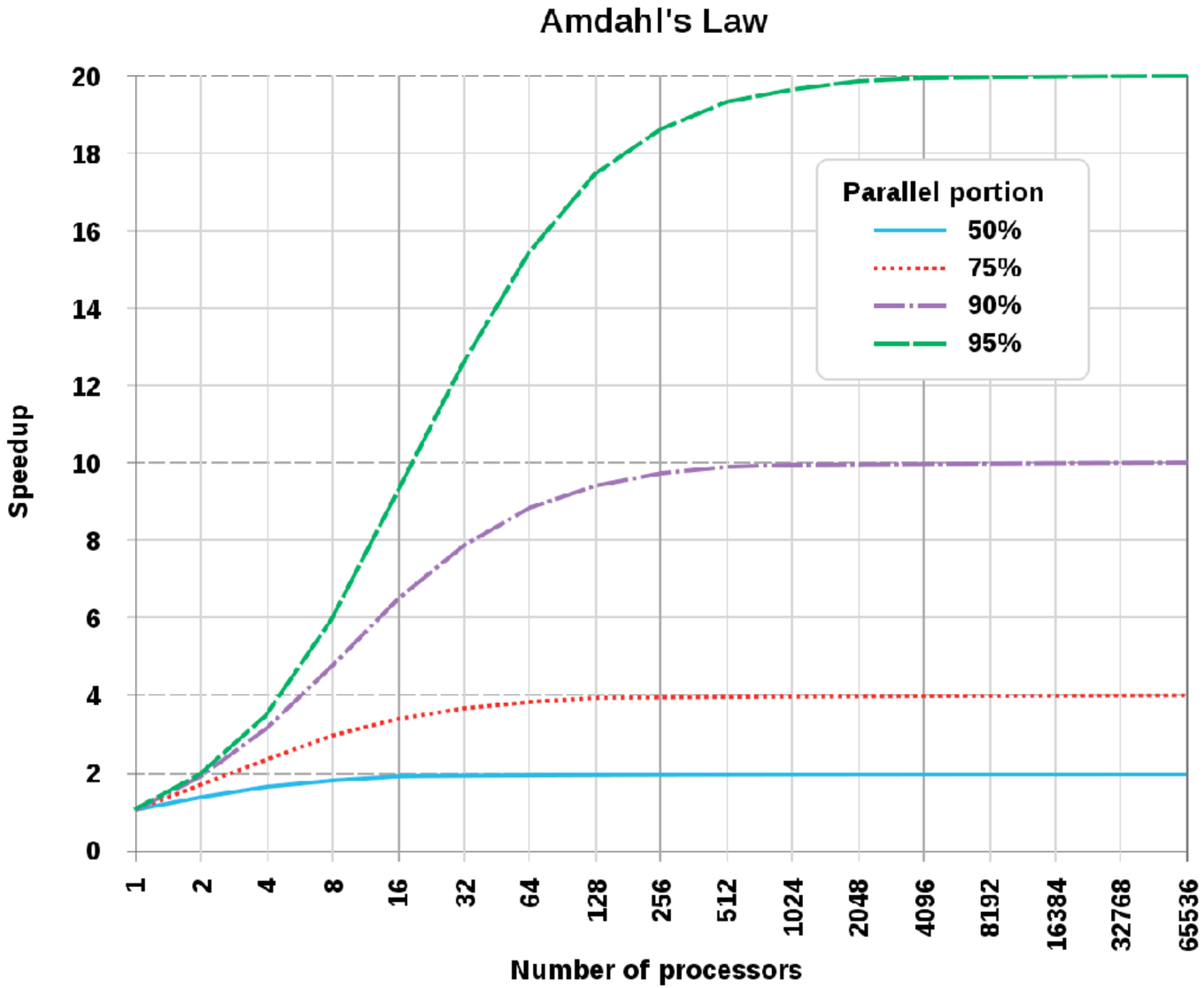
Efficiency:
Running as many jobs as possible for a given allocation of computer time

Speed:
The amount of time (real, "human time") to run a specific simulation from when it starts

Time-to-solution:
Speed + the time waiting in queue

@Tetralith: wall-time limit **7 days**
@Dardel: **24h** (7 day option)

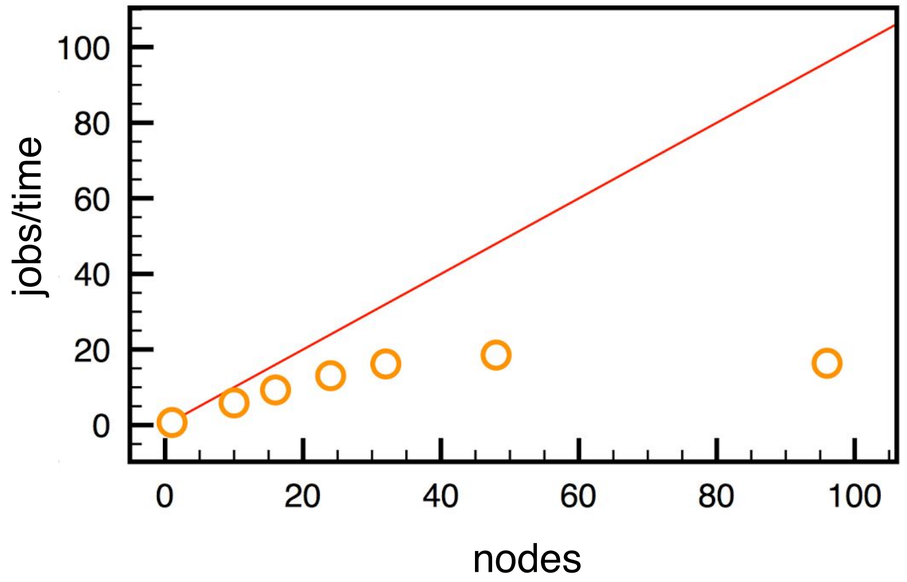
Parallelization - limitations

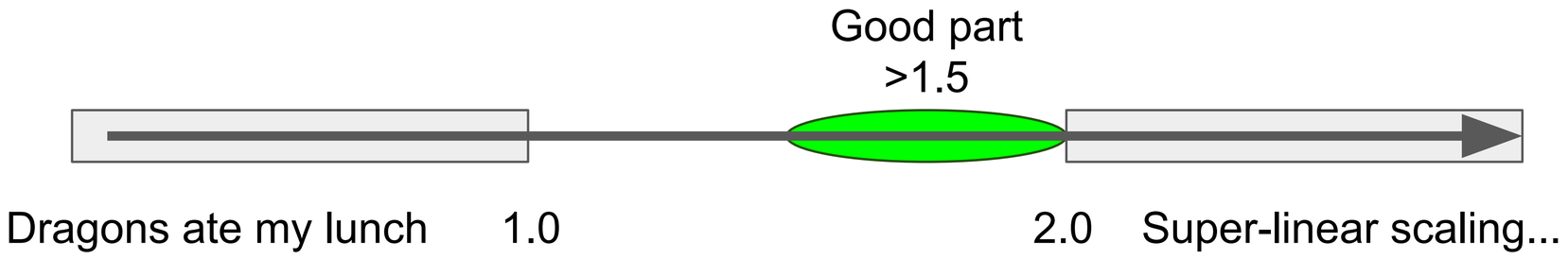


Simple scaling analysis

A minimal scaling analysis can save lots of allocated core hours…

1. Tool your runscript to time your simulation
2. Run an initial best guess number of cores (n)
3. Run the same test on half the number of cores (n/2)
4. Score = time(n/2) / time(n)





Hardware - affects best practices

- soon to retire

- **Tetralith** - Intel Xeon Gold 6130 2.1GHz

  1 node = 32 cores (96GB RAM, fat node 384GB)

  1832 x **60 x**

- **Dardel** - HPE Cray EX, AMD EPYC 2.25 GHz

  1 node = 128 cores (256GB RAM, 512GB, 1024GB, 2048GB)

  488 x **20 x** **8 x** **2 x**

Hardware - affects best practices

- installation in 2025-6!

- **Arrhenius** - AMD Epyc Turin + Nvidia Grace Hopper

CPU part **192 x** 1 node = 256 cores AMD Epyc Turin 2.7 GHz x2 (768GB RAM)

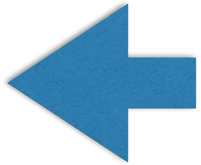
20 x 1 node = 256 cores AMD Epyc Turin 2.4 GHz x2 (3072GB RAM)

GPU part **328 x** 1 node = 4 x Nvidia Grace Hopper, GH200
(72 Arm Neoverse v2 cores, H100 GPU)

vasp-gpu version!

Running VASP @Arrhenius training 2026

Starting advice (reminder)

- Read the documentation!
- VASP default settings  good starting point
- Caution: “inherited” starting files
- Avoid overly complex INCAR
- Possible differences in centres installations

refer to respective webpages / documentation

Quick check your run

- How much/what resources to use?
  - Check NBANDS `$ grep NBANDS OUTCAR`
  - Use ca. 8 bands/core
- How long will it take?

```
$ grep LOOP OUTCAR
$ grep LOOP+ OUTCAR
```

  - scales with k-points (IBZKPT) `$ grep k-points OUTCAR`
- Does it converge? `$ cat OSZICAR`
- Problems? `$ less slurm*.out`

Quick check your run: tools

- `sacct`
- login to node & run `top`, `htop`
- @Tetralith: `jobload`, `jobsh`

```
$ man <command>
```

"-s r" for running job

```
$ sacct --user=<username> -X --format=Elapsed,State,AllocCPUS%9,CPUTimeRaw%20 --starttime=2019-10-01
```

```
$ sacct -e
```

example: `--format=JobID,Submit,Start,End,Elapsed,Nodelist,State,AllocCPUS%9,CPUTimeRaw%20`

```
$ seff <jobid>
```

summary of run

```
$ squeue -u <username>
$ scancel <jobid>
```

@Tetralith:

```
$ jobload <jobid>
$ jobsh <node>
```

INCAR parameters

- PREC - “precision”, ENCUT and FFT grids
- ENCUT - plane wave energy cutoff
- ALGO - wf optimisation
- NBANDS - if not set, auto-determined
- NSIM - for RMM-DIIS algorithm (ALGO)
- NCORE or NPAR - bands treated in parallel
- KPAR - k-point parallel

INCAR parameters

accuracy / method

- PREC - “precision”, ENCUT and FFT grids
- ENCUT - plane wave energy cutoff **Completeness of basis-set** **Recommended to set!**
- ALGO - wf optimisation
- NBANDS - if not set, auto-determined **Must be the same for Etot comparison!**

parallel calcs.

- NSIM - for RMM-DIIS algorithm (ALGO)
- NCORE or NPAR - bands treated in parallel
- KPAR - k-point parallel

PREC

- PREC = “precision”, sets ENCUT and FFT grids
- PREC = Normal, **default**
- PREC = Accurate, highly accurate forces
- OBS: Recommended to set ENCUT by hand

More on accuracy

- NGX, NGY, NGZ — coarse plane wave FFT grid
  can edit directly (otherwise PREC)
- NGXF, NGYF, NGZF — finer FFT grid
  - also see ENAUG
- LREAL=.FALSE. — default, might be needed for high accuracy

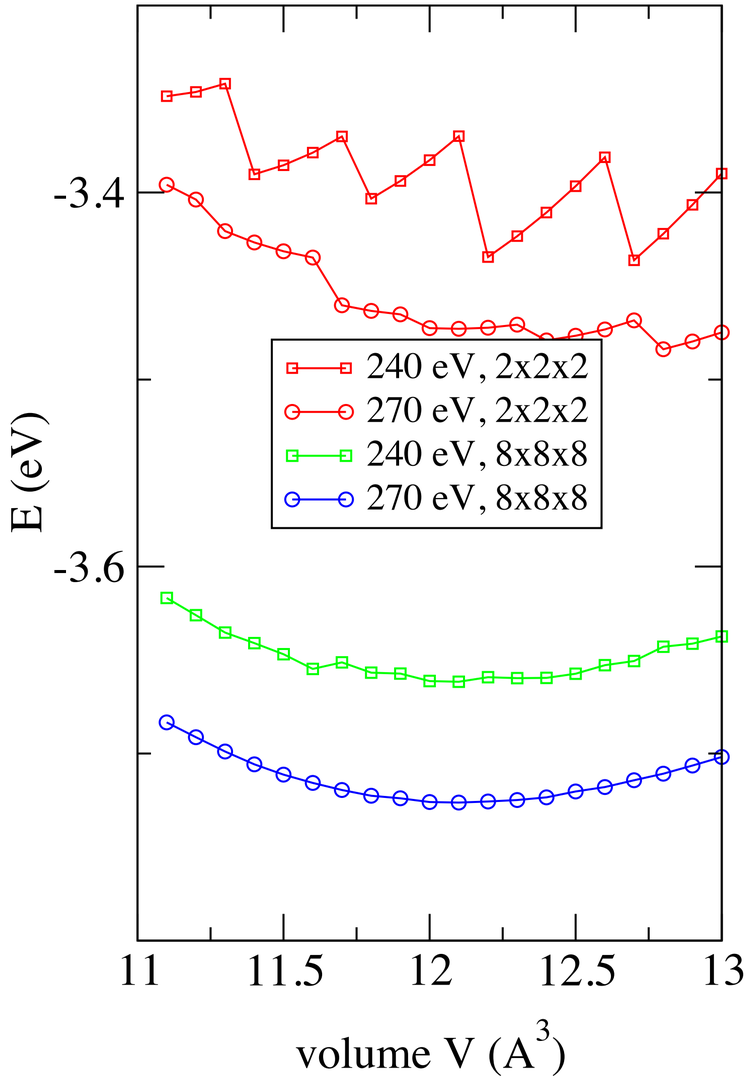
if proj. operators determined in real space, or not

otherwise use faster: LREAL = Auto

In some cases, need VASP installation with no special optimization flags

Convergence, ENCUT and k-mesh

- Cu example by G. Kresse
- Basis-set **changes** with volume
- Cell-shape relaxations, increase ENCUT = ENMAX x1.3
- Read section on structure relaxation



NBANDS

- NBANDS = NELECT/2 + NION/2 (ISPIN=1)
- May change due to parallelization!
- **Easy to divide**, 2^n, 4, 8, 12, 16, …
- select NBANDS = 511 or 512?
- Min limit, 1 band/core
- **Affects Etot!**

Run e.g. quick job to check NBANDS:

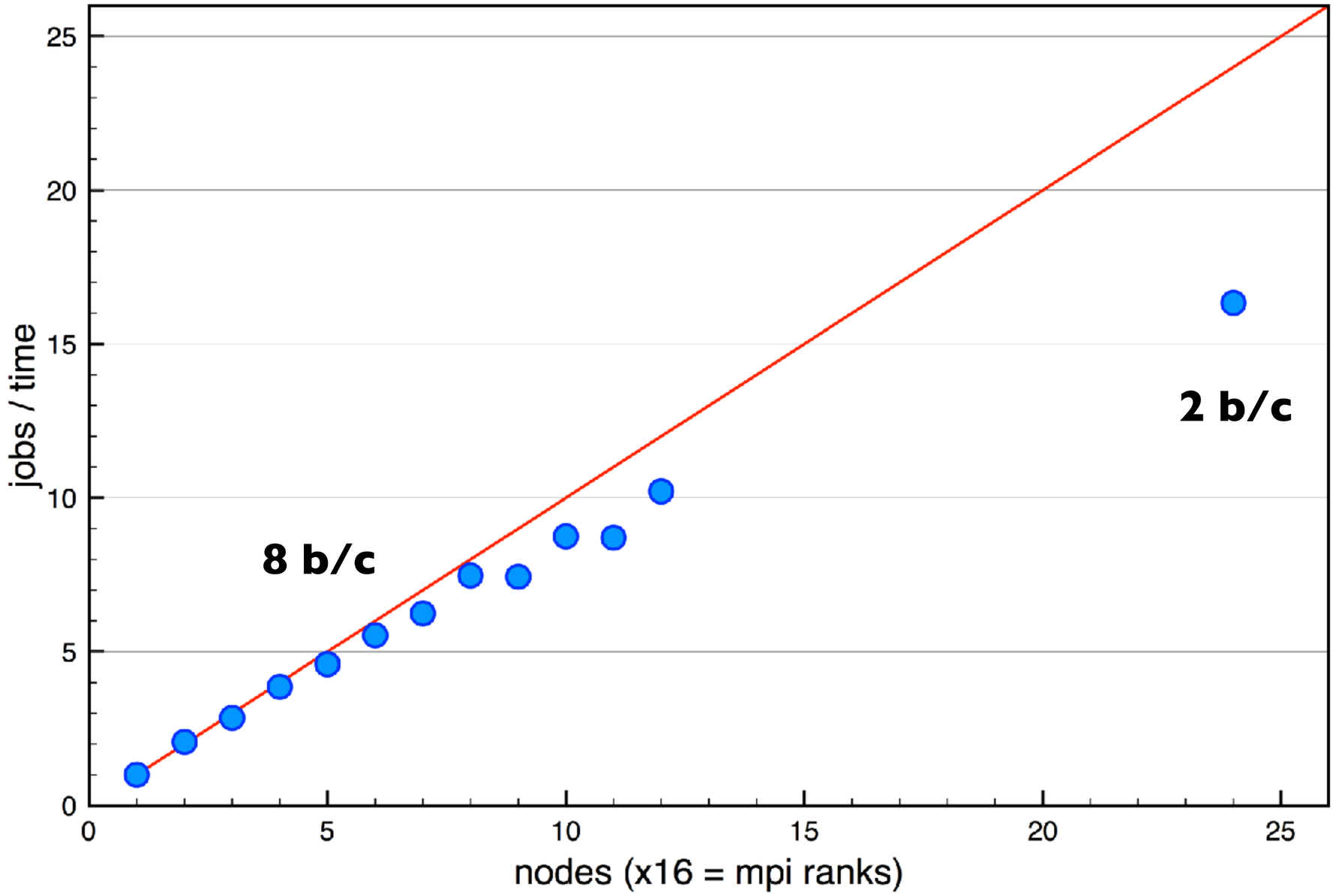
```
#SBATCH --reservation=devel   @Tetralith

$ grep NBANDS OUTCAR
```

How many cores - efficient and/or fast?

- Start from # of bands, **NBANDS**
- 1 band/core: typically inefficient
- 2 bands/core: ~50% parallel efficiency
- 8 bands/core: good starting point
  - try e.g. **cores ≈ NBANDS / 8**

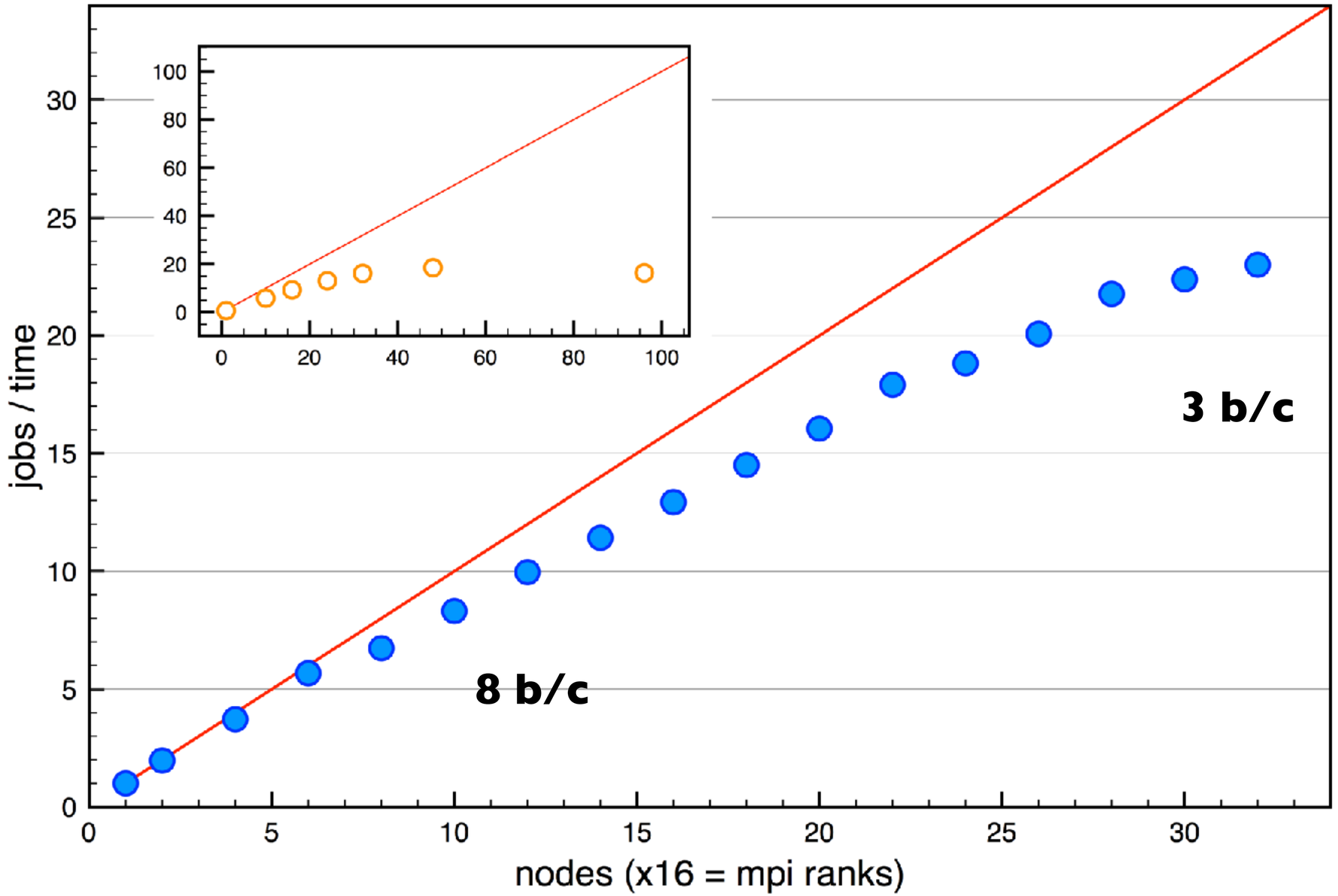
Si-H/Ag(111) 129 atoms, VASP PBE @Triolith (old)



NBANDS=750
4 k-points

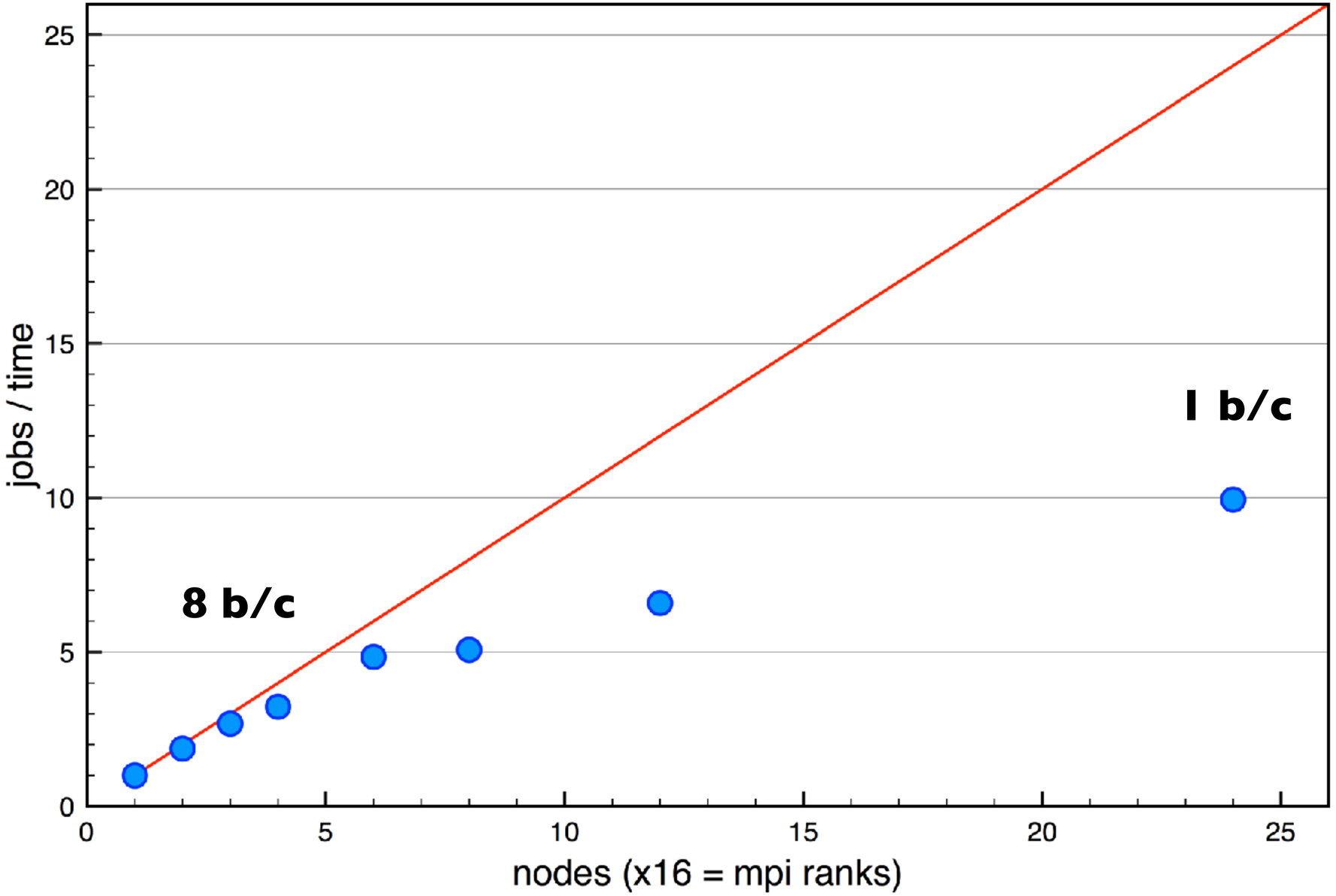
Triolith had **16** c/node
Tetralith: **32** c/node
Kebnekaise: **28** c/node

GaAsBi 512 atoms, VASP PBE @Triolith (old)



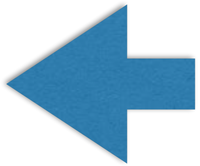
NBANDS=1536
4 k-points

GaAsBi 128 atoms, VASP HSE06 @Triolith (old)

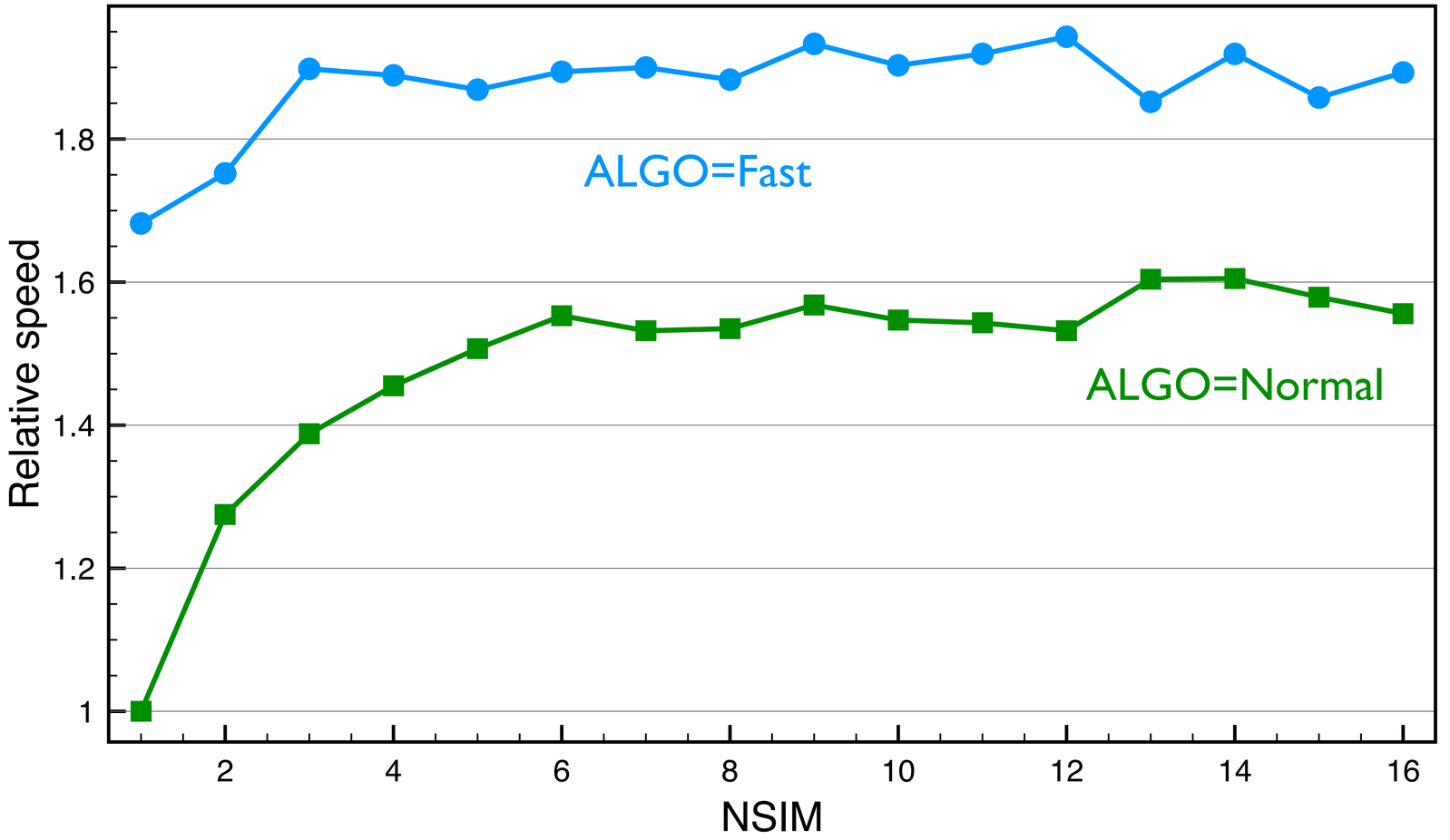


NBANDS=384
12 k-points

ALGO & NSIM

- Blocking mode for RMM-DIIS algorithm
- ALGO = Fast (Dav + R-D) / VeryFast (R-D)
- ALGO = Normal (Davidson algorithm), **default**
- **not for hybrid-DFT**, HSE06 (Damped, All, Normal)
- NSIM = 4, **default**  usually good (CPU)
- Tetralith: NSIM = 4, 8, 16, … (good to check!)

Si-H/Ag(111) 129 atoms, VASP PBE @Triolith (old)



default NSIM=4 seems OK here

NBANDS=750, 4 k-points

NCORE or NPAR
(default)

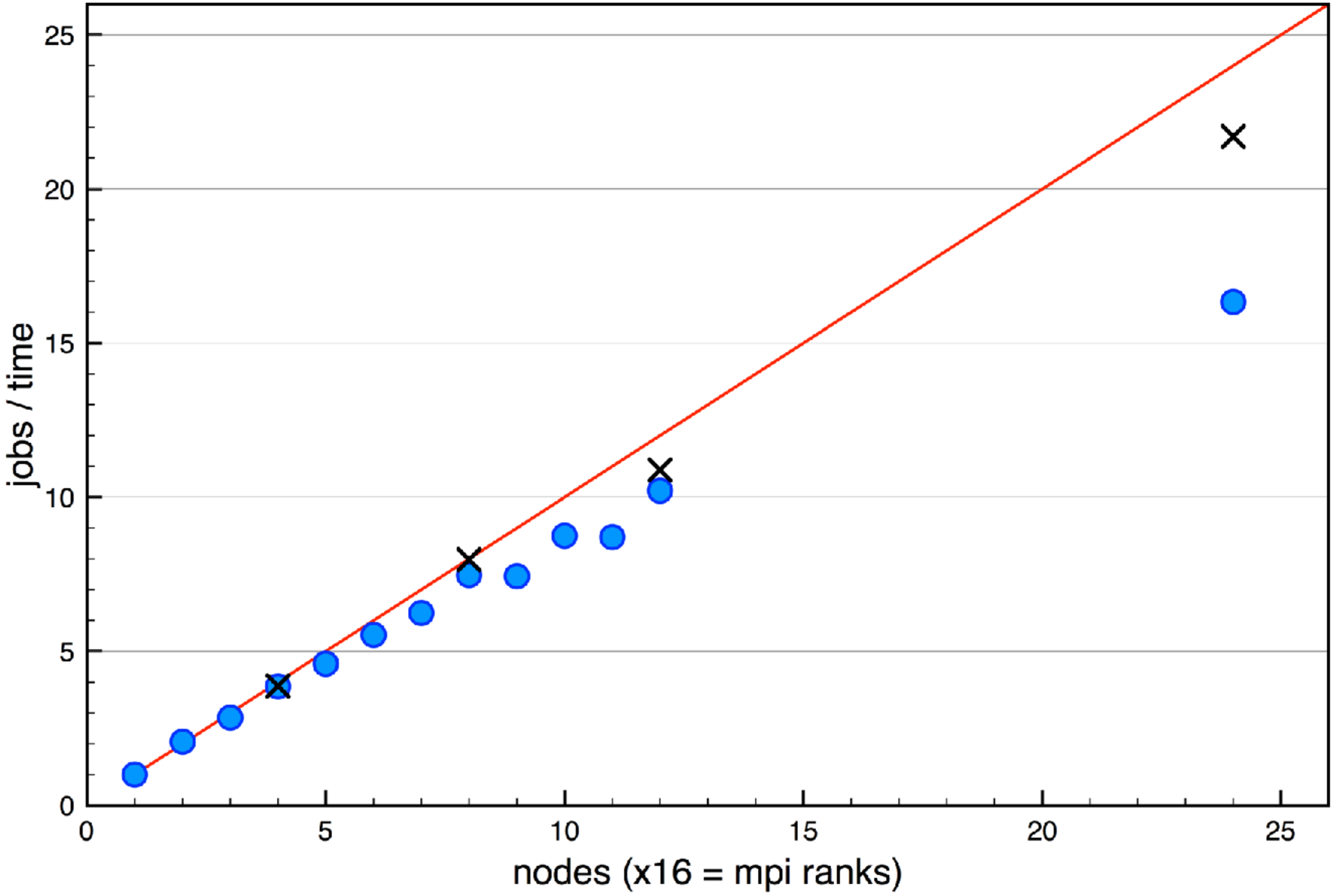
- cores per orbital / bands treated in parallel
- Davidson & RMM-DIIS algorithm
- ALGO = Normal & Fast, VeryFast
- NPAR = 1, saves memory
- NPAR = number of compute nodes
- NCORE = cores per compute node (or socket)

I find it easier to use NCORE, e.g. on Tetralith (if full node):
NCORE=32 or 16

KPAR

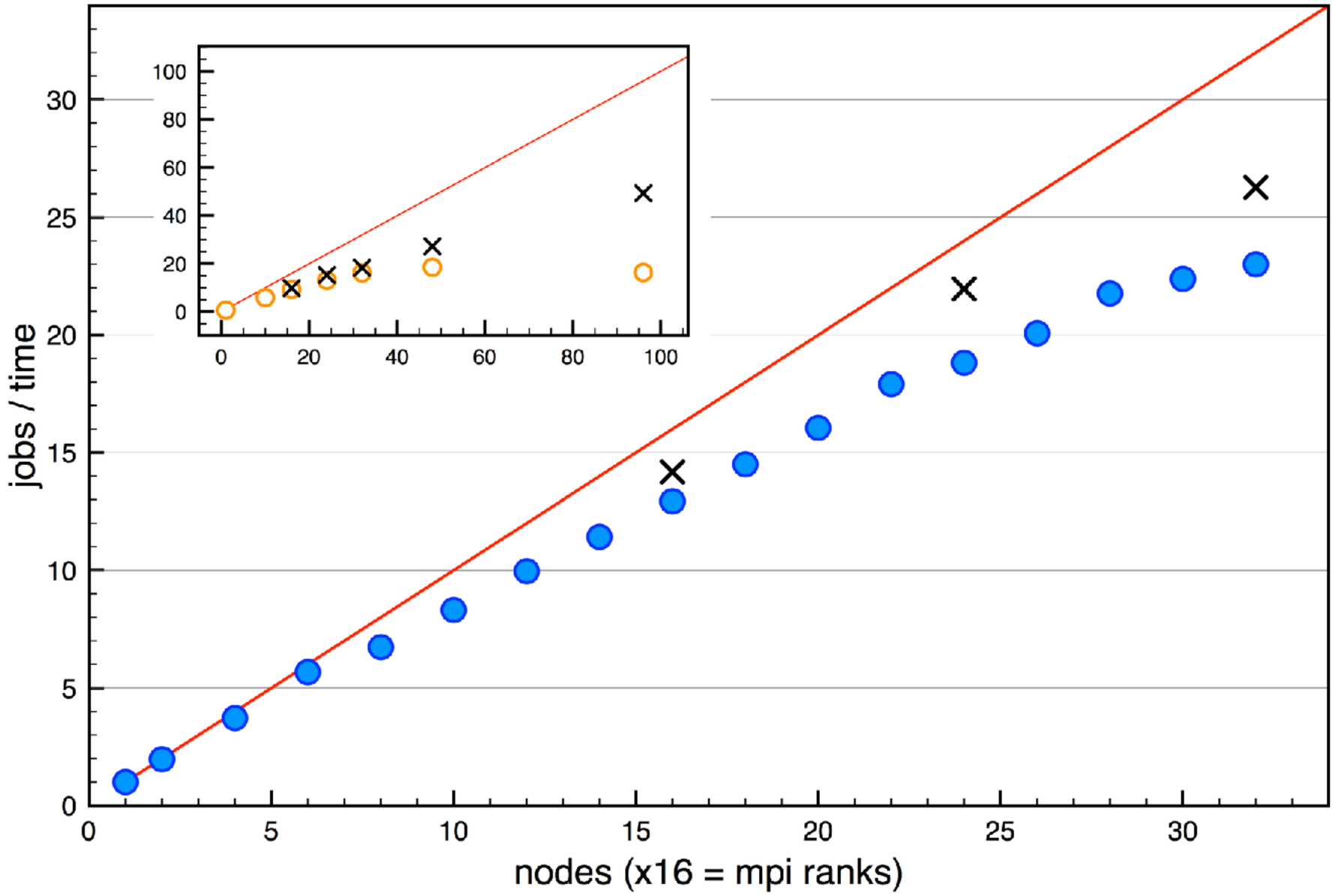
- KPAR = number of k-points treated in parallel
- in particular, good for **hybrid-DFT** jobs
- increase cores at least 2x
- try **KPAR = min (nodes, k-points)**

Si-H/Ag(111) 129 atoms, VASP PBE @Triolith (old)



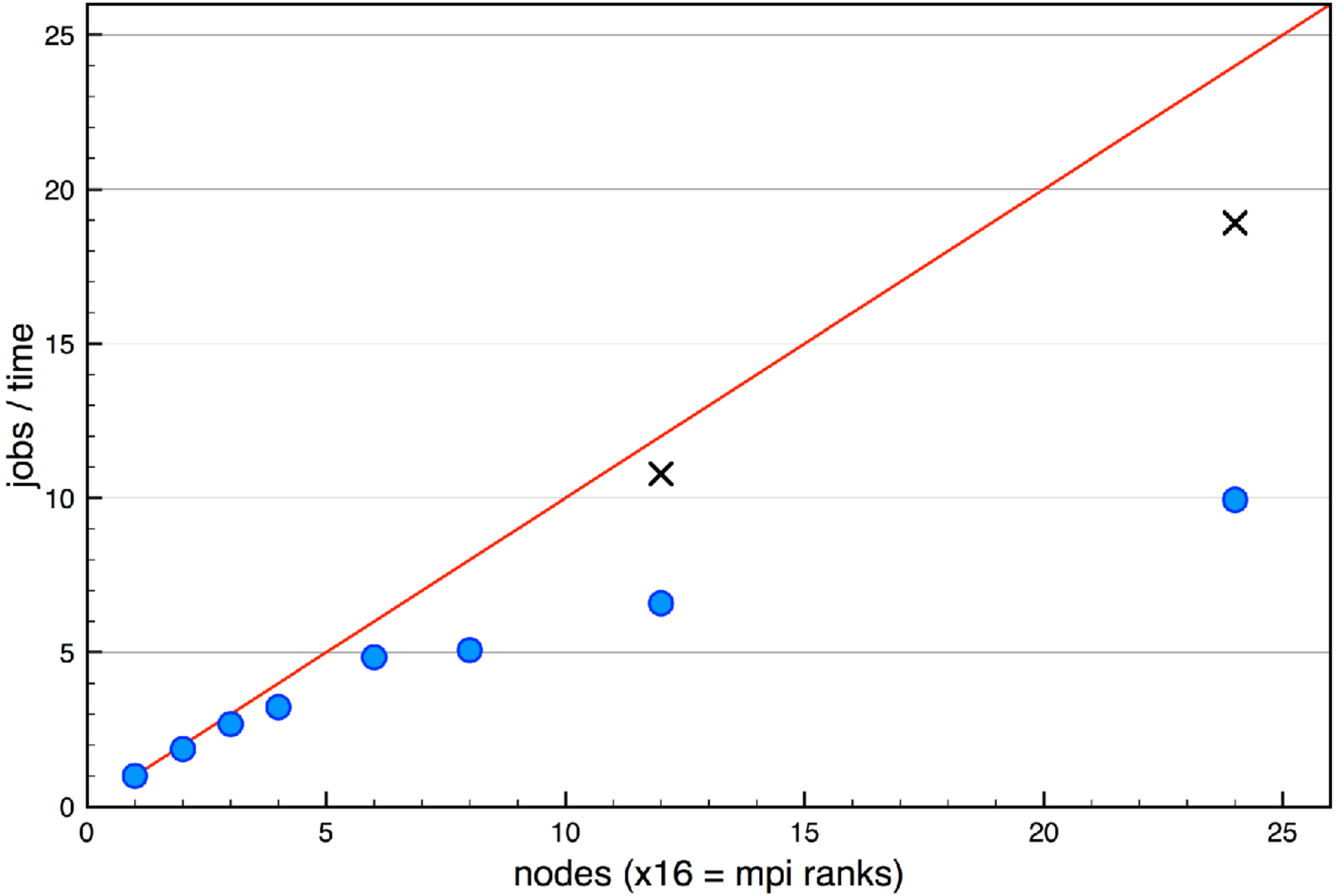
NBANDS=750
4 k-points

GaAsBi 512 atoms, VASP PBE @Triolith (old)



NBANDS=1536
4 k-points

GaAsBi 128 atoms, VASP HSE06 @Triolith (old)



NBANDS=384
12 k-points

Comparison for CPU & GPU

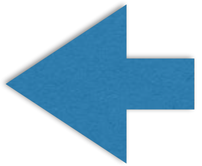
A brief test case @NSC clusters

VASP testing @NSC

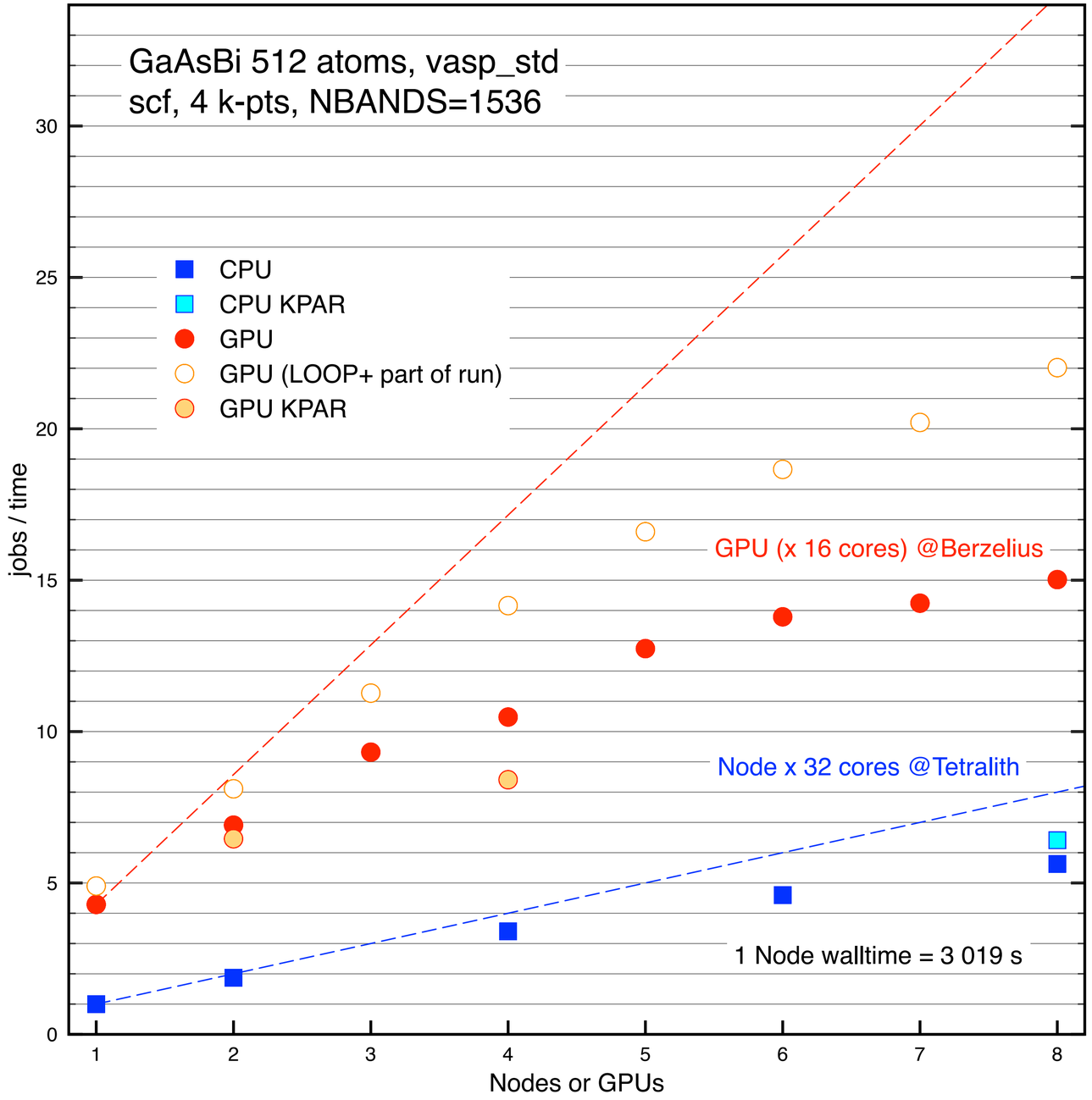
- From VASP6, test-suite included
- Peter Larsson's test-suite (old)
- Memory bandwidth important for VASP
- 10-100 runs for same job -> statistics
- Different builds (flags) for comparison
- Different (new) compiler suites, libraries
- Use collection of old problematic jobs… (BRMIX)

GPU

- VASP6 GPU **OpenACC** (old: CUDA)
- Nvidia GPUs (at this time)
- Different optimization than CPU
- KPAR - k-point parallelization ok
- NSIM - very important to check!
  - e.g. test with NSIM = 8, 16, 32
- NCORE - not supported
- **GPU RAM** possible bottleneck



different from CPU



Notes:

"Too small" job for many GPUs?

Few k-points (4)

Beware of "LOOP+" time ≠ wall-time

Fast, run full job and check wall-time

vasp_std, regular (scf) run

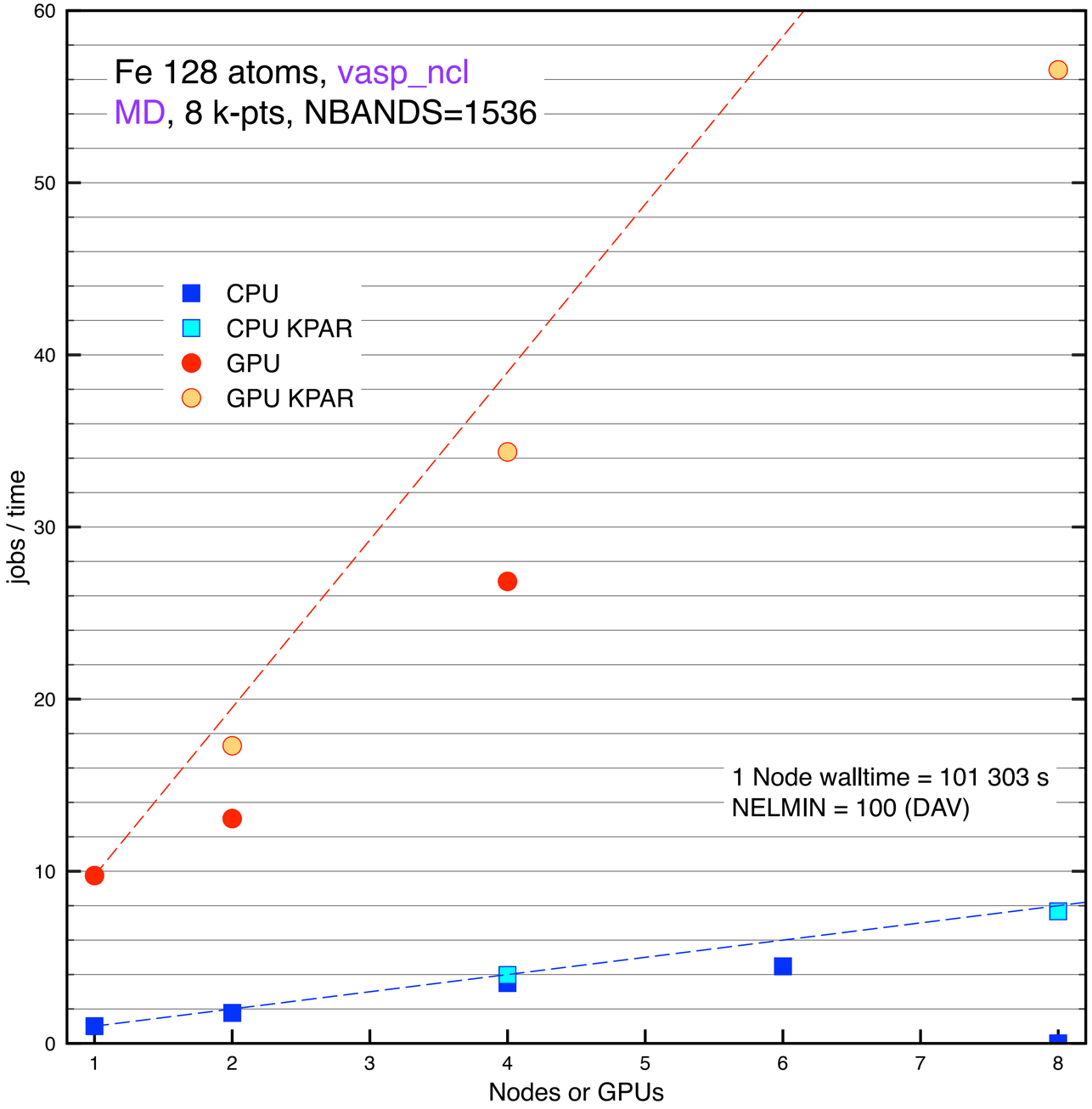
512 atoms

4 k-points

XC = PBE

NBANDS = 1536

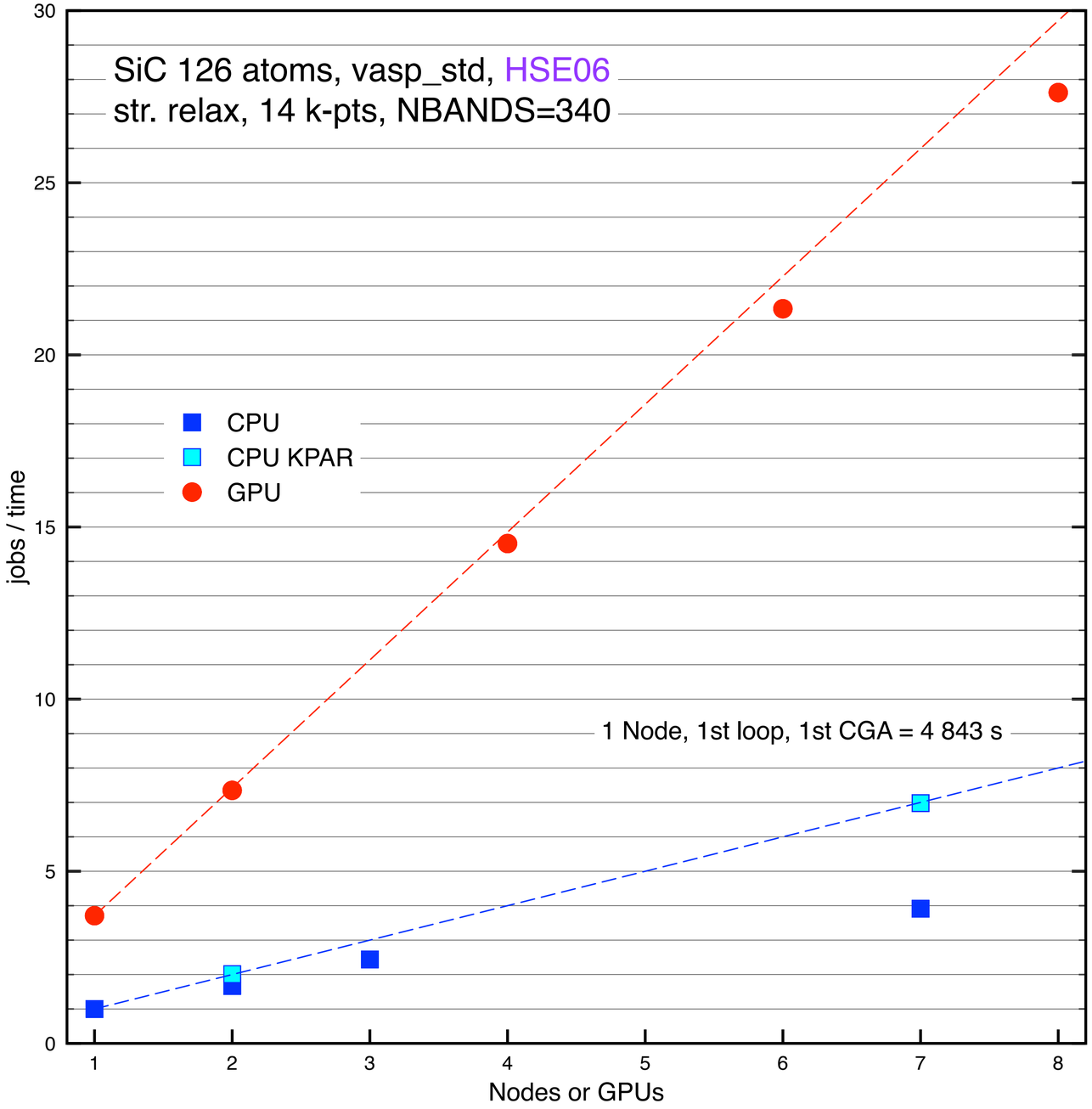
ISPIN = 1



Notes:

vasp_ncl, “heavier” job
Molecular Dynamics (MD)
1 GPU ≈ 10 x Node
Good scaling with KPAR
Run full job and check wall-time
Also runs on 1/2 GPU

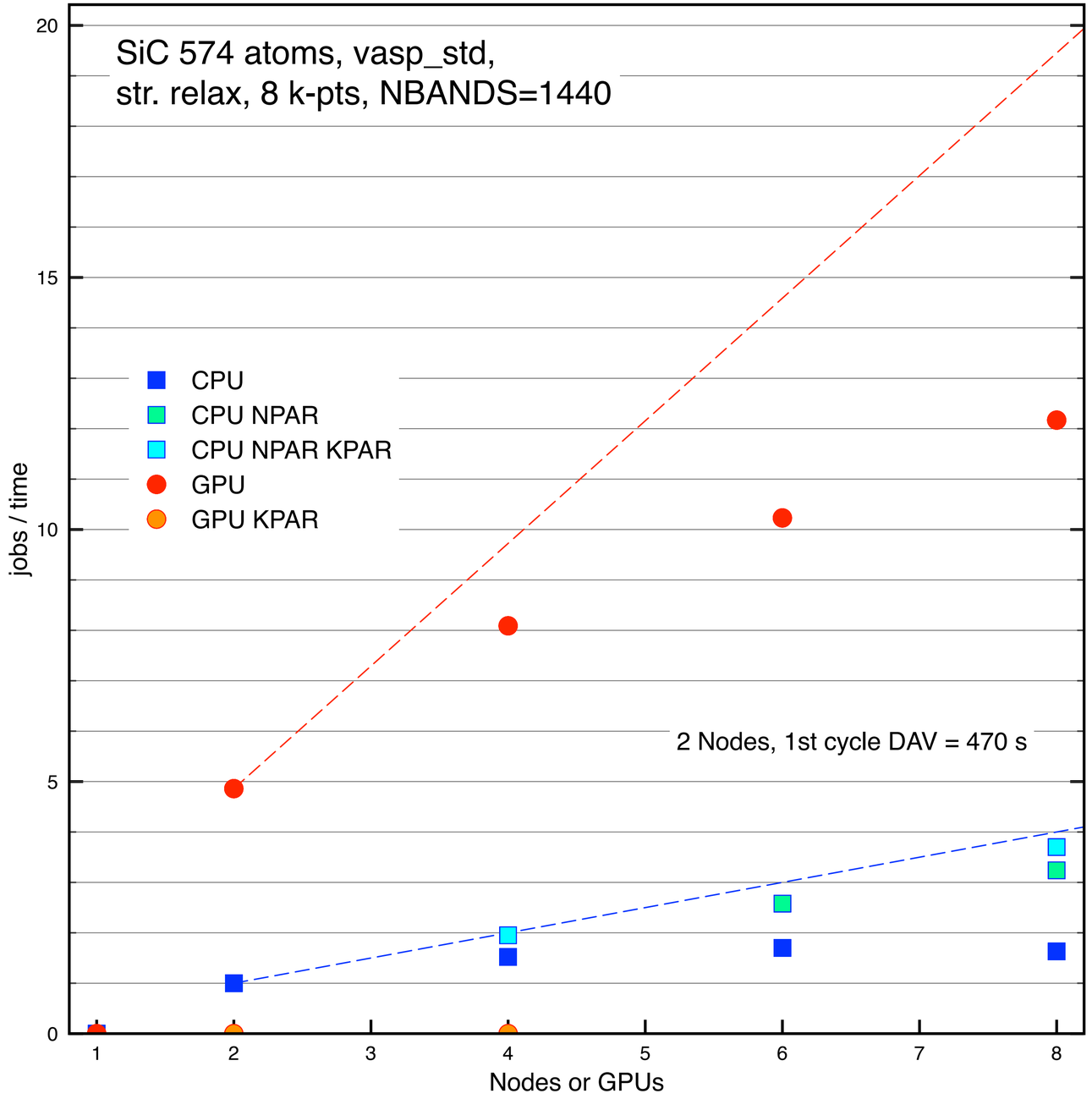
vasp_ncl, non-collinear MD run
128 atoms
8 k-points
XC = PBE
NBANDS = 1536
ISPIN = 2



Notes:

"Heavy" HSE06 job
Many k-points (14)
Excellent scaling CPU & GPU
Time from 1st CGA step, 1st cycle
Also runs on 1/2 GPU

vasp_std, structure relaxation
126 atoms
14 k-points
XC = HSE06
NBANDS = 340
ISPIN = 2



Notes:

Memory heavy

GPU with KPAR out-of-memory

CPU with KPAR works well

Average DAV time, 1st cycle

vasp_std, structure relaxation

574 atoms

8 k-points

XC = PBE

NBANDS = 1440

ISPIN = 2

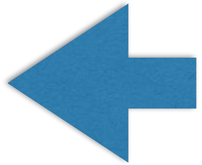
Summary: GPU runs

- Tetralith CPU - note “old” system
- Berzelius GPU - Nvidia A100
- 3.7 - 4.9 jobs/time, `vasp_std`
- 9.8 jobs/time, `vasp_ncl`
- “Big enough” jobs for using several GPUs
- KPAR, efficient split between GPUs
- Check for new Arrhenius system!

Problem solving & Summary

Different problems and their (possible) solution

Possible problems

- Input related  many kinds
- Memory (too little)
- **Job size vs. allocation** (mismatch)
  - Inefficient use (wasting core-hours)
- “Difficult” calculations (too costly, not possible)
- **Bugs** (compilers, old versions, …)

  - sometimes from choice of compiler flags which in theory ought to be OK…

Memory (RAM) issues

starting with regular PBE → running HSE06, GW

- changing **type** of calcs.

2x2x2 k-mesh → 4x4x4 k-mesh x 8 k-points

- increasing the **k-mesh**

ENCUT = 400 eV → ENCUT = 600 eV x 1.8

- increasing **energy cutoff**

$$n_{pw} = \propto \texttt{ENCUT}^{3/2}$$

Memory (RAM) issues …solutions

- Reduce cores/node, e.g. 24c/node, 16c/node

```
#SBATCH --ntasks-per-node=16
#SBATCH --mem=0
```

- More nodes (and reduce cores)

```
INCAR: NCORE=16
```

- @Tetralith: use “fat” memory nodes

```
#SBATCH -C fat
```

- Reduce k-mesh, ENCUT?
- Simplify system?
- @Tetralith: only use `--mem=0` (don’t set limit!)

Memory (RAM) issues …OpenMP?

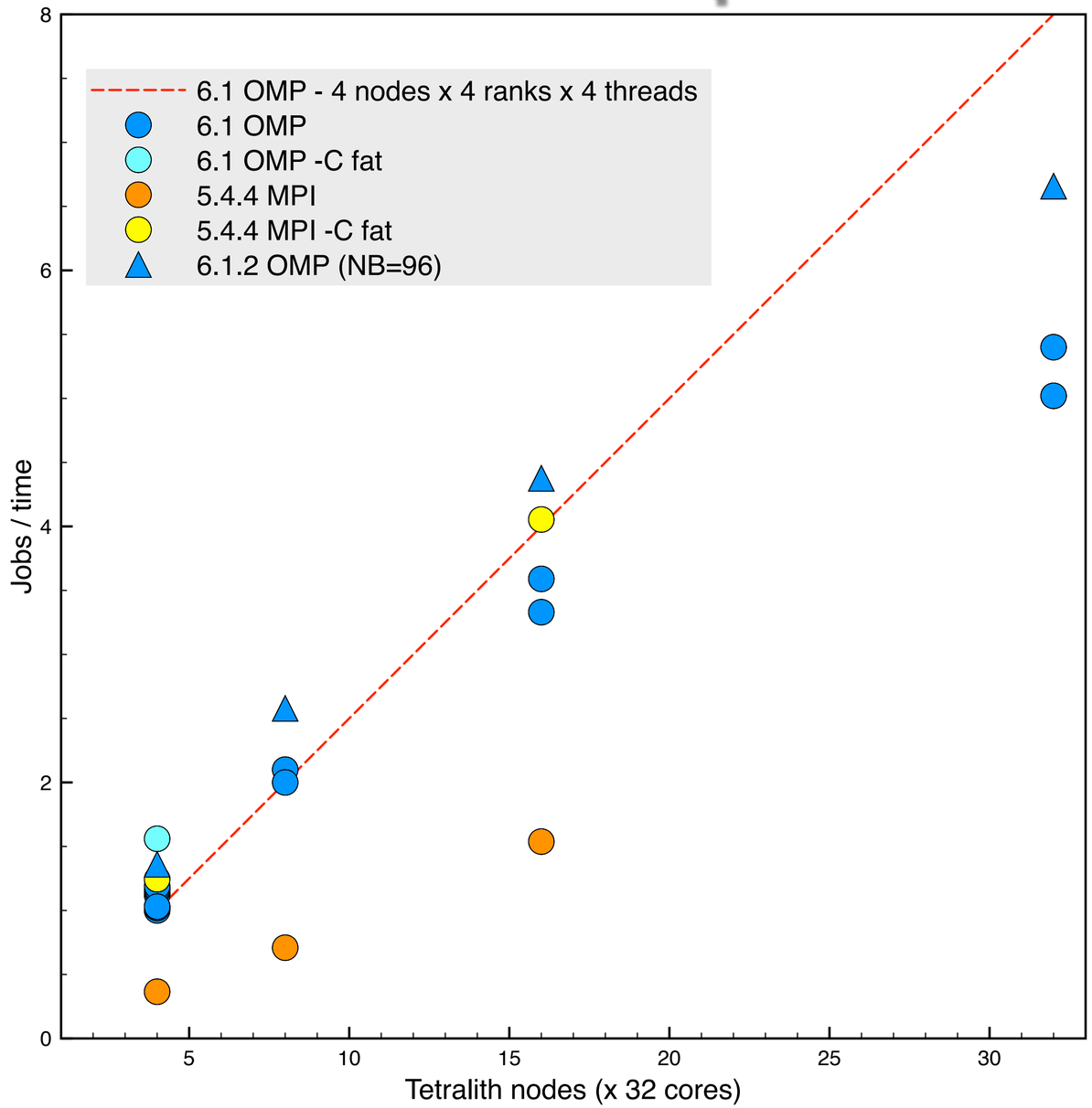
- OpenMP availble from VASP6
- May combine with “fat” nodes, reducing cores, …
- Test no. of threads

```
#!/bin/bash
#SBATCH -J vasp
#SBATCH -N 4
#SBATCH --ntasks-per-node=8
#SBATCH -t 4:00:00
#SBATCH -A naiss-xxx-yyy

export OMP_NUM_THREADS=4
export OMP_STACKSIZE=512m

module add VASP/6.4.0.14022023-omp-nsc1-intel-2018a-eb
mpprun vasp_std
```

Example: Fe 2000 atoms MD



Notes:

Fe 2000 atoms PBE MD

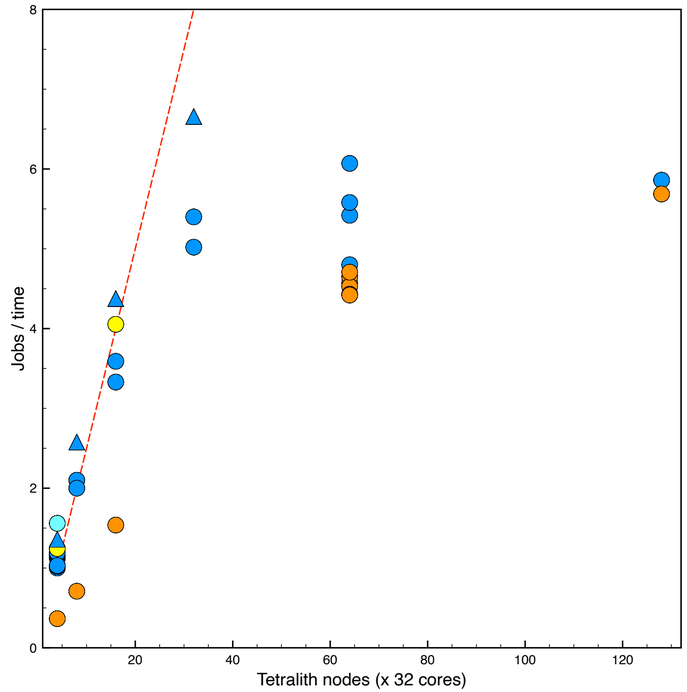
Memory heavy

VASP6 vs VASP5

OpenMP+MPI more efficient

“Fat” node -> 4x 96 GiB RAM

test with NB=96 in src/scala.F



Example: ScNZr 1000 atoms band str.

- ScNZr 1000 atoms PBE LDA+U, band str. (ICHARG=11)
- 428 k-points -> split into 1 node jobs @Tetralith
- Cost (core-hours) ≈ walltime x jobs x 32
  ≈ iteration time x iterations
- PREC=Accurate, LREAL=.FALSE., ENCUT=400 (600)
  ~527 631 core-hours, using 16 cores for more RAM -> 1 k-pt/job
- PREC=Normal, LREAL=Auto, ENCUT=400
  ~14 761 core-hours -> **35.7x**, less RAM needed -> 3 k-pts/job
  - NCORE=16 (somewhat better than 32)
  - NSIM=32 (much better than default 4, 8)
- Other ways (reduce k-points, wannier90, …)?

Warning/advice output

Check stdout (slurm-***.out) for warnings!

```
 -----------------------------------------------------------------------------
|                                                                             |
|           W    W    AA    RRRRR   N    N  II  N    N   GGGG   !!!          |
|           W    W   A  A   R    R  NN   N  II  NN   N  G    G  !!!          |
|           W    W  A    A  R    R  N N  N  II  N N  N  G       !!!          |
|           W WW W  AAAAAA  RRRRR   N  N N  II  N  N N  G  GGG   !           |
|           WW  WW  A    A  R   R   N   NN  II  N   NN  G    G               |
|           W    W  A    A  R    R  N    N  II  N    N   GGGG   !!!          |
|                                                                             |
|      ALGO=A and IALGO=5X tend to fail with the tetrahedron method           |
|      (e.g. Bloechls method ISMEAR=-5 is not variational)                    |
|      please switch to IMSEAR=0-n, except for DOS calculations               |
|      For DOS calculations use IALGO=53 after preconverging with ISMEAR>=0   |
|          I HOPE YOU KNOW, WHAT YOU ARE  DOING                               |
|                                                                             |
 -----------------------------------------------------------------------------
```

Common support cases

- complicated INCAR…
- structure (POSCAR)
- k-mesh (KPOINTS)
- NCORE/NPAR, KPAR
- VASP version
- cores
- memory

Common support cases

- complicated INCAR… simplify & try again! ALGO=N
- structure (POSCAR) reasonable/correct?
- k-mesh (KPOINTS) Γ-centered?
- **NCORE**/~~NPAR~~, KPAR simplify (possibly remove)!
- VASP version try latest (possibly "vanilla" version)!

```
$ module add VASP/5.4.4.16052018-nsc1-intel-2018a-eb
```

- cores too few/many?
- memory
  - ENCUT
  - k-mesh

See previous discussion

larger memory nodes:

```
#SBATCH -C fat
```

reduce cores/node:

```
#SBATCH --ntasks-per-node=16
```

```
INCAR: NCORE=16
```

Notes / Reminders

- Same NBANDS when comparing E_{tot}?
- Large enough NBANDS? e.g. increase for higher temp.
- Sufficient accuracy for your case?
- Use vasp_gam for 1 k-point calcs.
- LWAVE=.FALSE. WAVECAR might grow very large, don't output if not needed

Summary “rules of thumb”

- job size (max): **total cores ≈ NBANDS / 8**
- **NSIM = 4** (default), test increasing it
- **NCORE = cores/node**
- PREC = Accurate - if forces important
- ENCUT = ENMAX x1.5 - “max setting”
- KPAR = min (nodes, k-points) **HSE06, especially useful**
- In general, INCAR default settings OK
- GPU: important to check NSIM

Resources

- Wiki and Manual
  Check in detail!
- Examples, tutorials
- Presentations
- Forum

Find all the links:
https://vasp.at/

- Also other resources, materials and tools for VASP
- Peter Larsson's old blog at NSC: https://www.nsc.liu.se/~pla/
- Tetralith installations: https://www.nsc.liu.se/software/docs/vasp/
- Dardel installations: https://support.pdc.kth.se/doc/applications/vasp/

Questions / trouble @NAISS clusters? https://supr.naiss.se/support/