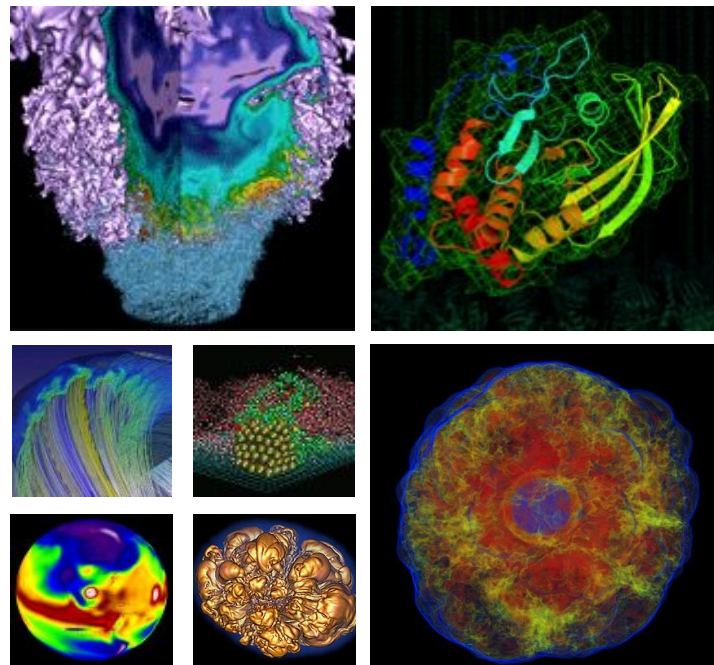


# EigenSolver Performance Comparison on Cray XC Systems



Brandon Cook, Thorsten Kurth, Jack Deslippe

CRAY® Pierre Carrier, Nick Hill, Nathan Wichmann

- Evaluate the general performance of two important eigenvalue solvers: ELPA and SCALAPACK
- Determine rules for optimal parameters
  - Automate the parameter search
- Evaluate OpenMP
- Compare Hardware (Xeon, Xeon-Phi)
- Offer a benchmarking suite (to be continued...)

# ELPA and ScaLAPACK (PZHEEVD)



Quantum chemistry  
Density functional Theory  
(DFT),  
Time-Dependent DFT

Engineering  
Noise Vibration, and  
harshness (NVH)

NXN  
dense  
matrices



$$Ac = Bc\lambda$$

Machine learning  
Singular Value  
Decomposition (SVD)



# Block cyclic distribution



### Global view of array

|                   |                   |                   |                   |                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| (0,0)<br>0:[0,0]  | (0,1)<br>1:[0,1]  | (0,2)<br>2:[0,2]  | (0,3)<br>3:[0,3]  | (0,4)<br>0:[0,0]  | (0,5)<br>1:[0,1]  | (0,6)<br>2:[0,2]  | (0,7)<br>3:[0,3]  |
| (1,0)<br>4:[1,0]  | (1,1)<br>5:[1,1]  | (1,2)<br>6:[1,2]  | (1,3)<br>7:[1,3]  | (1,4)<br>4:[1,0]  | (1,5)<br>5:[1,1]  | (1,6)<br>6:[1,2]  | (1,7)<br>7:[1,3]  |
| (2,0)<br>8:[2,0]  | (2,1)<br>9:[2,1]  | (2,2)<br>10:[2,2] | (2,3)<br>11:[2,3] | (2,4)<br>8:[2,0]  | (2,5)<br>9:[2,1]  | (2,6)<br>10:[2,2] | (2,7)<br>11:[2,3] |
| (3,0)<br>12:[3,0] | (3,1)<br>13:[3,1] | (3,2)<br>14:[3,2] | (3,3)<br>15:[3,3] | (3,4)<br>12:[3,0] | (3,5)<br>13:[3,1] | (3,6)<br>14:[3,2] | (3,7)<br>15:[3,3] |
| (4,0)<br>0:[0,0]  | (4,1)<br>1:[0,1]  | (4,2)<br>2:[0,2]  | (4,3)<br>3:[0,3]  | (4,4)<br>0:[0,0]  | (4,5)<br>1:[0,1]  | (4,6)<br>2:[0,2]  | (4,7)<br>3:[0,3]  |
| (5,0)<br>4:[1,0]  | (5,1)<br>5:[1,1]  | (5,2)<br>6:[1,2]  | (5,3)<br>7:[1,3]  | (5,4)<br>4:[1,0]  | (5,5)<br>5:[1,1]  | (5,6)<br>6:[1,2]  | (5,7)<br>7:[1,3]  |
| (6,0)<br>8:[2,0]  | (6,1)<br>9:[2,1]  | (6,2)<br>10:[2,2] | (6,3)<br>11:[2,3] | (6,4)<br>8:[2,0]  | (6,5)<br>9:[2,1]  | (6,6)<br>10:[2,2] | (6,7)<br>11:[2,3] |
| (7,0)<br>12:[3,0] | (7,1)<br>13:[3,1] | (7,2)<br>14:[3,2] | (7,3)<br>15:[3,3] | (7,4)<br>12:[3,0] | (7,5)<br>13:[3,1] | (7,6)<br>14:[3,2] | (7,7)<br>15:[3,3] |

blocksize  
nb

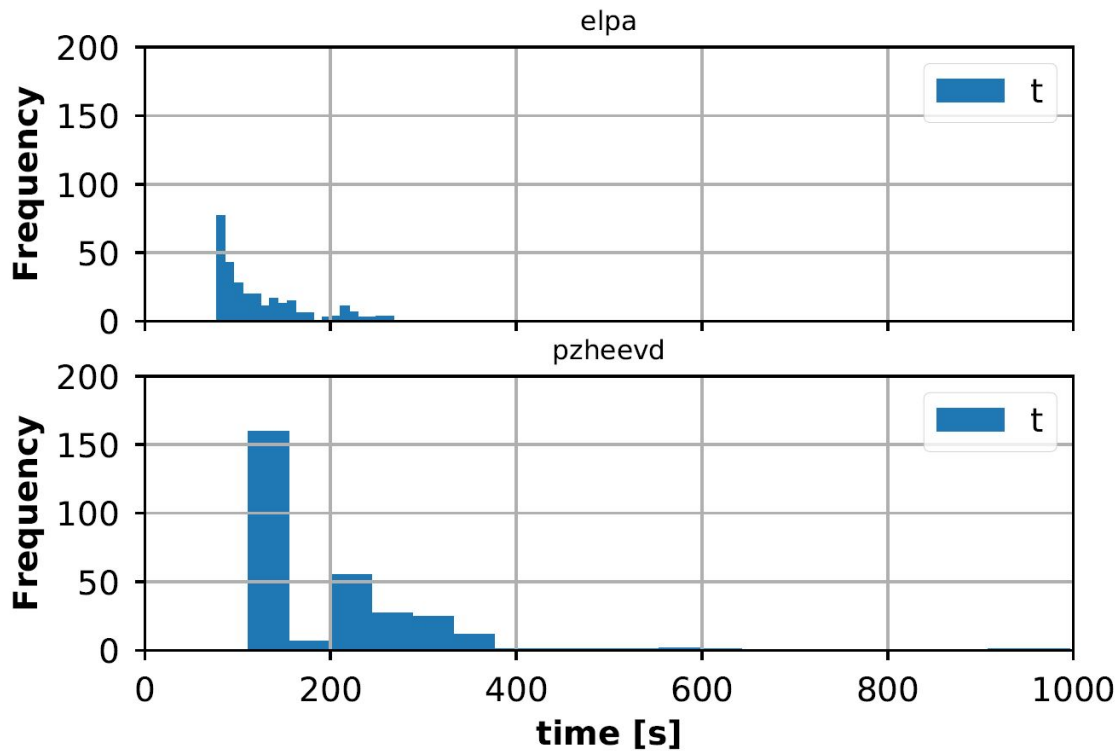
### Process view of array

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| (0,0) | (0,4) | (0,1) | (0,5) | (0,2) | (0,6) | (0,3) | (0,7) |
| (4,0) | (4,4) | (4,1) | (4,5) | (4,2) | (4,6) | (4,3) | (4,7) |
| (1,0) | (1,4) | (1,1) | (1,5) | (1,2) | (1,6) | (1,3) | (1,7) |
| (5,0) | (5,4) | (5,1) | (5,5) | (5,2) | (5,6) | (5,3) | (5,7) |
| (2,0) | (2,4) | (2,1) | (2,5) | (2,2) | (2,6) | (2,3) | (2,7) |
| (6,0) | (6,4) | (6,1) | (6,5) | (6,2) | (6,6) | (6,3) | (6,7) |
| (3,0) | (3,4) | (3,1) | (3,5) | (3,2) | (3,6) | (3,3) | (3,7) |
| (7,0) | (7,4) | (7,1) | (7,5) | (7,2) | (7,6) | (7,3) | (7,7) |

ny = 4

nx = 4

# Distribution of performance

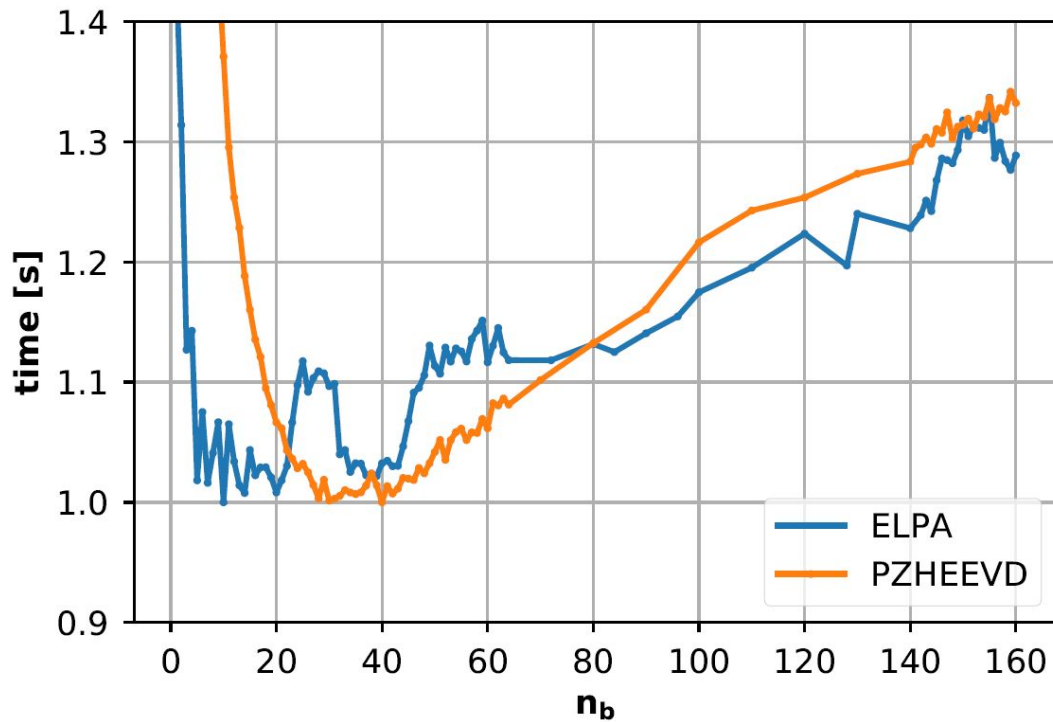


|              |         |
|--------------|---------|
| (nx,ny)      | ALL     |
| nb           | 1...100 |
| N            | 20000   |
| Architecture | KNL     |
| Nodes        | 4       |

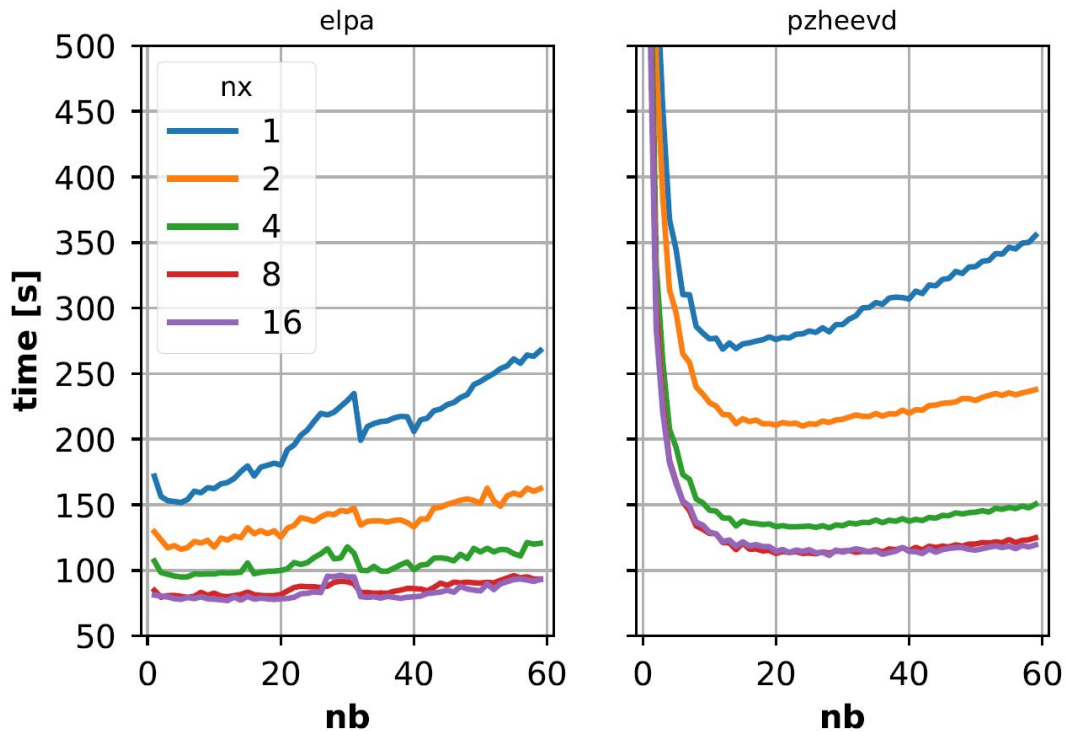
# Parameter landscape - nb



|              |          |
|--------------|----------|
| (nx,ny)      | (10, 32) |
| N            | 40000    |
| Architecture | SKX      |
| Nodes        | 8        |



# Parameter landscape

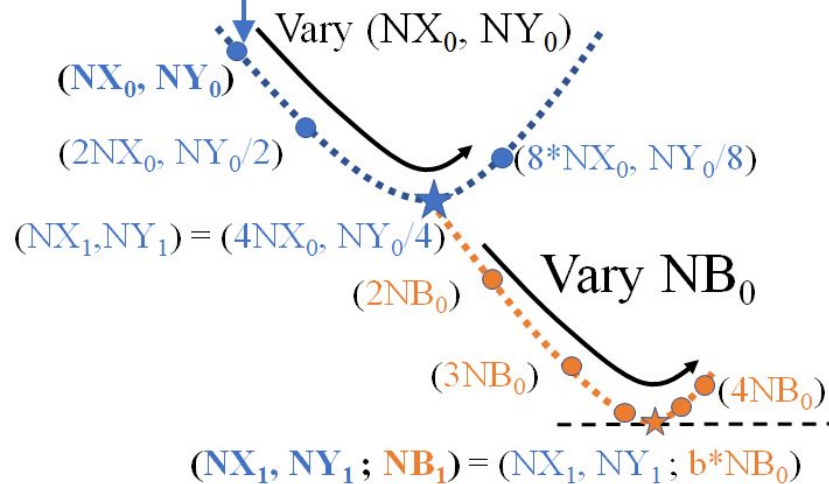


|              |       |
|--------------|-------|
| N            | 20000 |
| Architecture | KNL   |
| Nodes        | 4     |

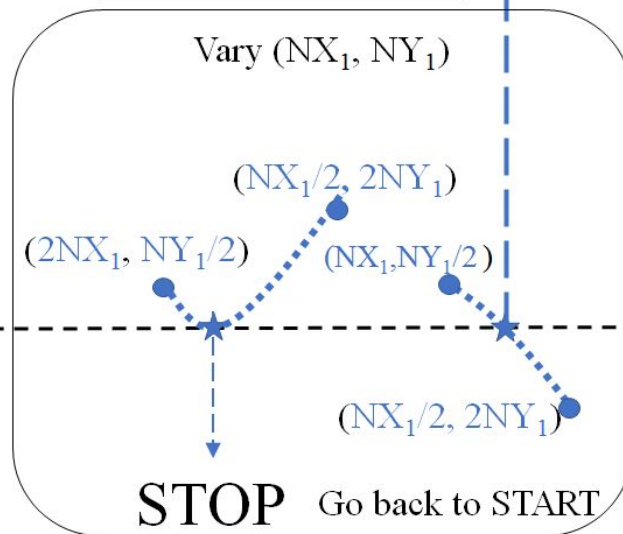
# Manual optimization



START:

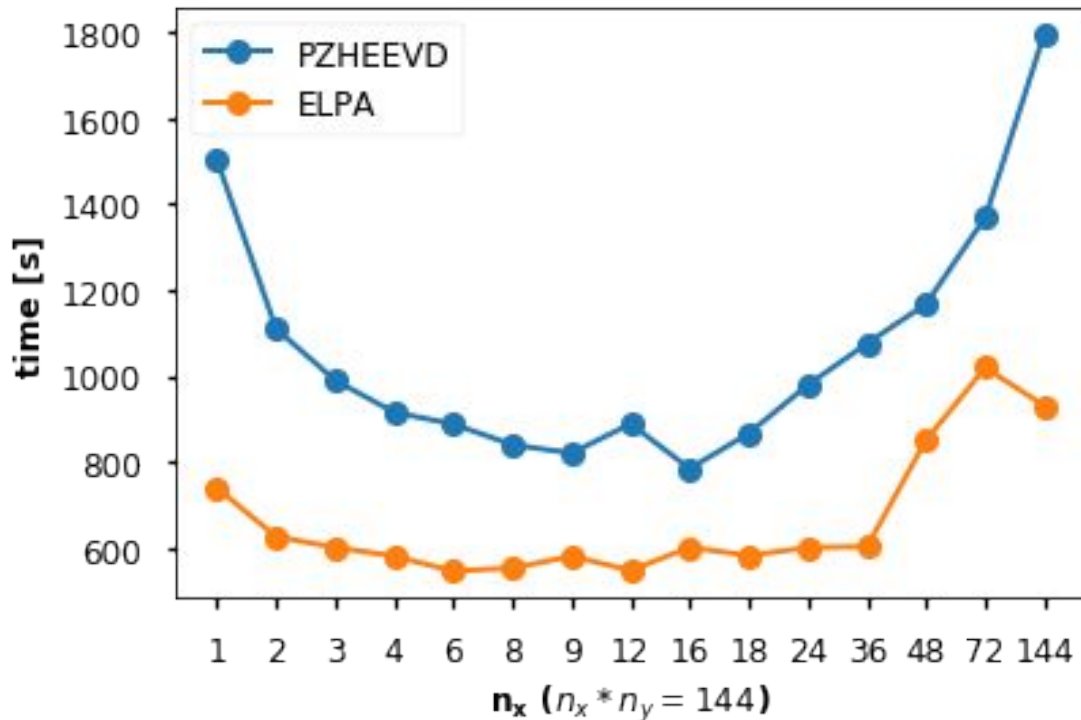


2 situations:





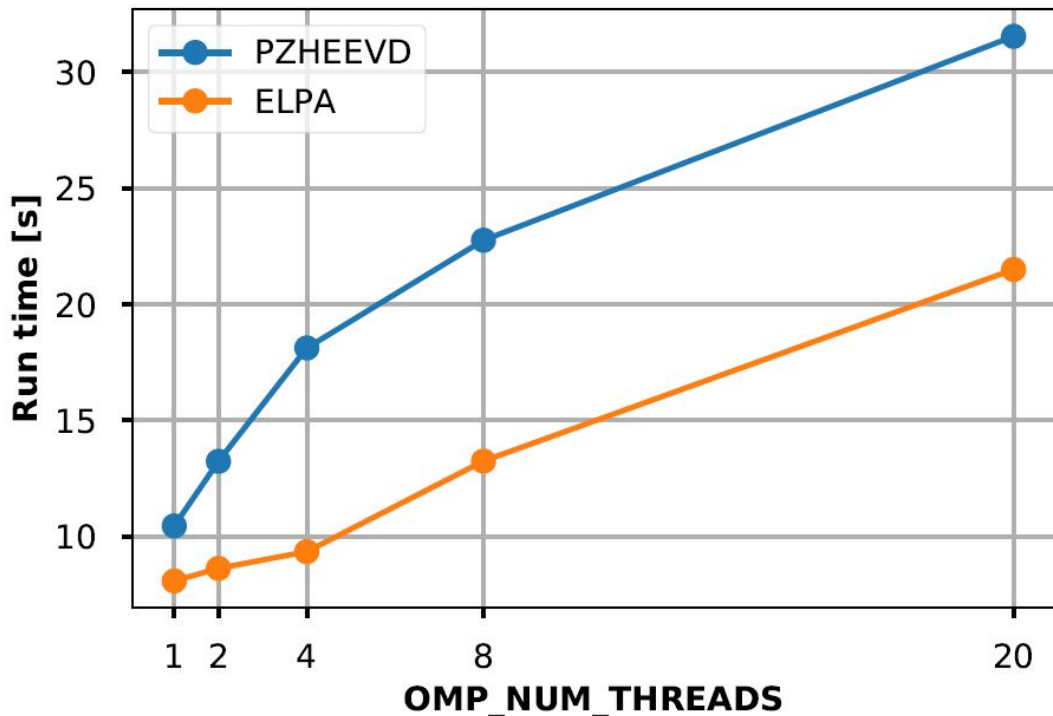
# Optimal (nx,ny)



- $n_x=n_y$  is good, but not the best
- $(n_x, n_y) \neq (n_y, n_x)$
- $n_x | n_y = 1$  is always bad

|              |       |
|--------------|-------|
| N            | 20000 |
| Architecture | BDW   |
| Nodes        | 4     |
| MPI ranks    | 144   |

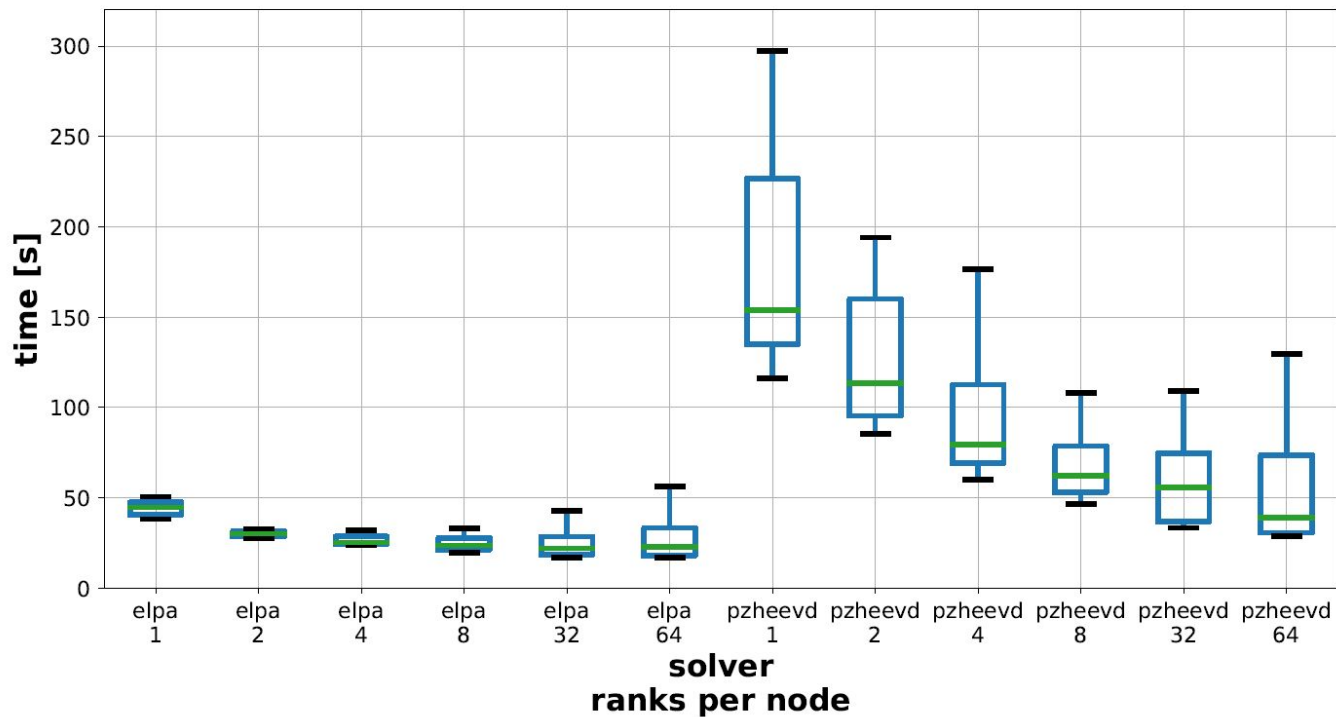
# OpenMP scalability



- OMP\_NUM\_THREADS \* MPI\_Ranks kept constant
- No OpenMP penalty would be a horizontal line
- ELPA is OK up to 4 threads
- Useful for e.g. Quantum Espresso Exact Exchange simulations

|              |      |
|--------------|------|
| (nx,ny)      | best |
| nb           | best |
| Architecture | SKX  |
| nodes        | 4    |

# OpenMP performance

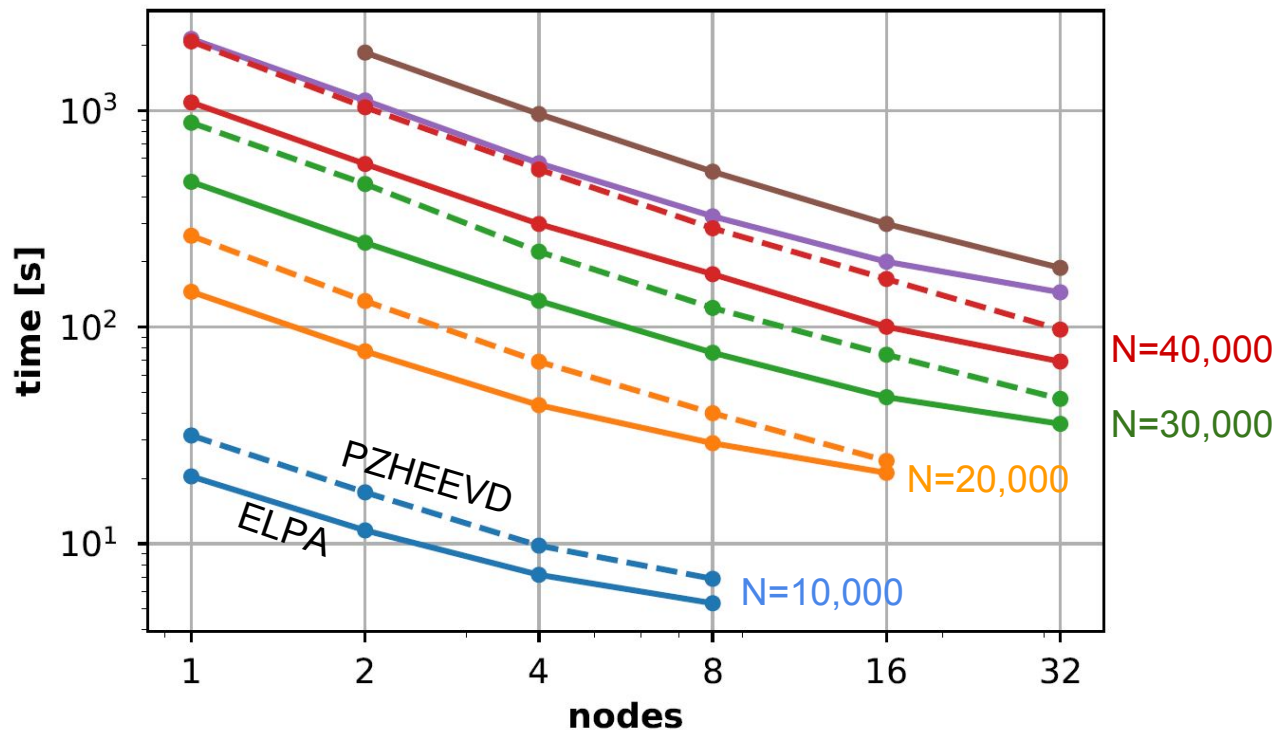


|              |       |
|--------------|-------|
| N            | 10000 |
| Architecture | KNL   |
| Nodes        | 4     |

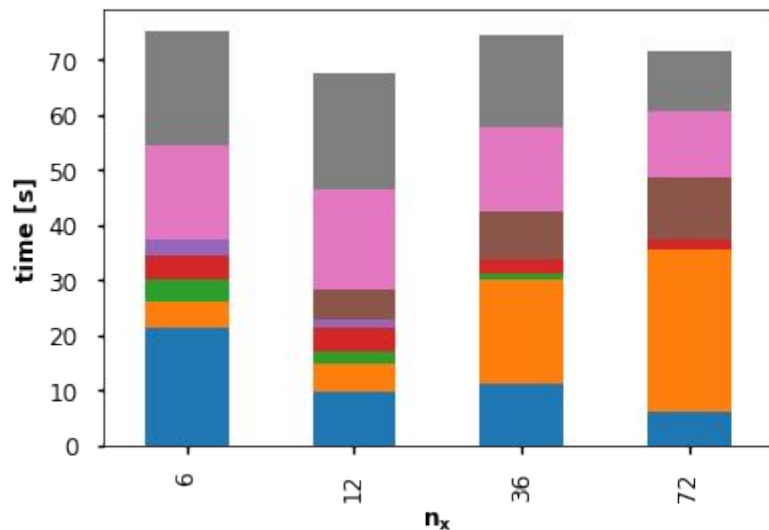
# Scaling of ELPA and PZHEEVD



|              |      |
|--------------|------|
| (nx,ny)      | best |
| nb           | best |
| Architecture | SKX  |



# ELPA profile

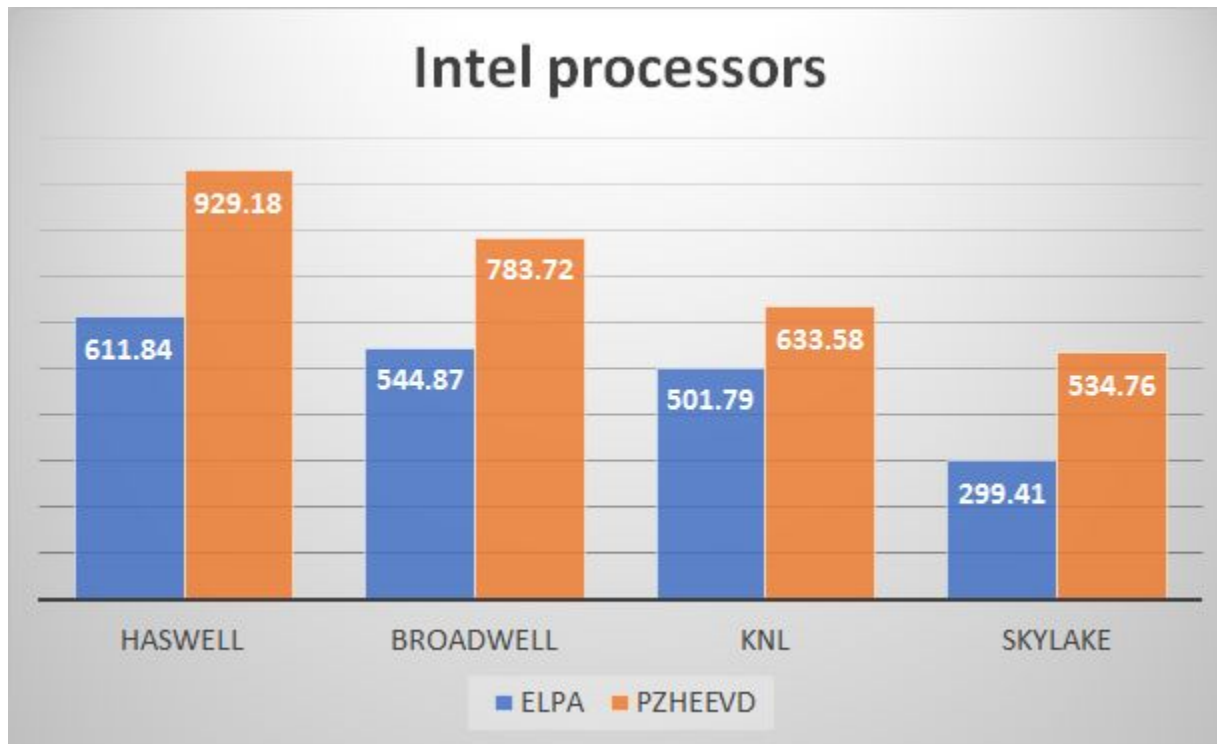


- obtained with Cray perftools
- MPI tradeoff
- $n_x=12$  is best overall time

# Processor evolution



solution  
time on 4  
nodes



# Benchmarking programs



## APPENDIX A. ELPA BENCHMARK

```
program benchmark_elpa
use elpa
use elpa_driver
...
call blacs_get(...)
call blacs_gridinit(...)
call blacs_gridinfo(...)
...
e => elpa_allocate()
call e%set("na", ln, err)
...
call e%set("complex_kernel",
ELPA_2STAGE_COMPLEX_AVX512_BLOCK1, err)
call e%eigenvectors(a_sym, w, z, err)
end program benchmark_elpa
```

## APPENDIX B. PZHEEVD BENCHMARK

```
program benchmark_pzheevd
...

call blacs_get(...)
call blacs_gridinit(...)
call blacs_gridinfo(...)
...

call pzheevd(..., temp, ...)
...

call pzheevd(...,work, ...)
end program benchmark_elpa
```

# Conclusions



- In general, ELPA is 20-30% faster than ScaLAPACK. When possible, choose ELPA (Quantum Espresso, VASP, etc).
- NX close to NY, is an optimum, but not always.
- NB: SCALAPACK choose a value in the 50s. ELPA choose a value in the 10s.
- The search space of ELPA contains multiple minima.
- The optimum (NX,NY) in ELPA is an interplay between MPI\_Bcast and MPI\_Allreduce.
- Threading is better in ELPA than in ScaLAPACK. More work needs to be done in that direction.
- Looking to expand this eigenvalue benchmarking suite:
  - FEAST
  - ELEMENTAL
  - ...



# Extras

---



# Spearmint optimization

