Parallel solution of the generalized eigenvalue problem given in a factored form

Edoardo di Napoli[1], Vedran Novaković[2], Gayatri Čaklović[3], Sanja Singer[4]

[1] Jülich Supercomputing Centre, and RWTH Aachen, Germany

[2] Universitat Jaume I, Castellón de la Plana, Spain

[3] M.S. student at Faculty of Science, Department of Mathematics, University of Zagreb, Croatia

[4] Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia

10th International Workshop on Parallel Matrix Algorithms and Applications
PMAA18,
June 27–29, 2018, Zürich, Switzerland

This work has been supported in part by
Croatian Science Foundation under the project IP–2014–09–3670.

Introduction

Outline of the talk:

- description of the problem,
- brief description of the two-sided Hari–Zimmermann (HZ) algorithm for the GEP,
- implementation details of the parallel algorithm,
- partial results of numerical testing.

DFT – Density Functional Theory

Density Functional Theory framework

- is used in simulation of the physical properties of complex quantum mechanical systems made of few dozens up to few hundreds of atoms
- the core of the method relies on the simultaneous solution of a set of Schrödinger–like equations also known as Kohn–Sham equations
- there exists a wide variety of approaches that can be used to "translate" the DFT mathematical layout into a computational tool.

FLAPW method

Full-potential Linearized Augmented Plane Wave (FLAPW) method

- FLAPW method is one od the most accurate methods — particular discretization of the DFT fundamental equations
- FLAPW is all-electron method — it explicitly describes all of the (potentially large number of) electrons in the material with a much larger number of basis function
- it is a quite computationally expensive method.

FLAPW method

Full-potential Linearized Augmented Plane Wave (FLAPW) method

- the discretization in FLAPW method leads to the solution of the generalized eigenvalue problem for matrices (H, S), where

$$H = \sum_{a=1}^{N_A} (A_a^* T^{[AA]} A_a + A_a^* T^{[AB]} B_a + B_a^* T^{[BA]} A_a + B_a^* T^{[BB]} B_a)$$

$$S = \sum_{a=1}^{N_A} (A_a^* A_a + B_a^* U_a^* U_a B_a),$$

where $A_a, B_a \in \mathbb{C}^{N_L \times N_G}$, $T_a^{[\cdots]} \in \mathbb{C}^{N_L \times N_L}$, $U \in \mathbb{C}^{N_L \times N_L}$ is a diagonal matrix, while $(T^{[AA]})^* = T^{[AA]}$, $(T^{[BB]})^* = T^{[BB]}$, and $(T^{[AB]})^* = T^{[BA]}$.

Problem sizes

Typical matrix sizes

- $N_A = \mathcal{O}(100)$, $N_G = \mathcal{O}(1000)$–$\mathcal{O}(10000)$, and $N_L = \mathcal{O}(100)$
- test examples NaCl – $N_A = 512$, $N_L = 49$
  - $N_G = 2256$, $N_G = 3893$, $N_G = 6217$, $N_G = 9273$
- test examples AuAg – $N_A = 128$, $N_L = 121$
  - $N_G = 3275$, $N_G = 5638$, $N_G = 8970$, $N_G = 13379$.

Computation of H and S

Proposed by Fabregat–Traver at al.

- write H as $H = H_{AA} + H_{AB+BA+BB}$

$$H_{AA} = \sum_{a=1}^{N_A} A_a^* T^{[AA]} A_a$$

$$H_{AB+BA+BB} = \sum_{a=1}^{N_A} \left(B_a^* T^{[BA]} A_a + A_a^* T^{[AB]} B_a + B_a^* T^{[BB]} B_a \right)$$

$$= \sum_{a=1}^{N_A} (B_a^* Z_a + Z_a^* B_a)$$

(`ZHER2K`s!), where

$$Z_a = T^{[BA]} A_a + \frac{1}{2} T^{[BB]} B_a.$$

Modification?

Why

- the algorithm proposed by Fabregat–Traver at al. computes in parallel only H and S — then use any GEVD,
- intention to keep matrices in a factored form – ideal for parallelization of the GEVD
- usage of one-sided methods — faster than the two-sided methods — columnwise action
- such approach usually computes small eigenvalues more accurately
- similar algorithm for the (real) generalized SVD is approximately 125 times faster than the LAPACK routine with theaded MKL.

Transform the problem!

Transformed problem

- by using the properties of matrices $T^{[\cdots]}$ it is obvious that the problem can be written as

$$H = \sum_{a=1}^{N_A} \begin{bmatrix} A_a^* & B_a^* \end{bmatrix} \begin{bmatrix} T^{[AA]} & T^{[AB]} \\ (T^{[AB]})^* & T^{[BB]} \end{bmatrix} \begin{bmatrix} A_a \\ B_a \end{bmatrix} := \sum_{k=1}^{n} H_k^* T_k H_k,$$

$$S = \sum_{a=1}^{N_A} \begin{bmatrix} A_a^* & B_a^* U_a^* \end{bmatrix} \begin{bmatrix} A_a \\ U_a B_a \end{bmatrix} := \sum_{k=1}^{n} S_k^* S_k.$$

Transform the problem!

... or as products of three (two) matices

$$H = \begin{bmatrix} H_1^* & \cdots & H_n^* \end{bmatrix} \begin{bmatrix} T_1 & & \\ & \ddots & \\ & & T_n \end{bmatrix} \begin{bmatrix} H_1 \\ \vdots \\ H_n \end{bmatrix} := \widetilde{F}^* T \widetilde{F}$$

$$S = \begin{bmatrix} S_1^* & \cdots & S_n^* \end{bmatrix} \begin{bmatrix} S_1 \\ \vdots \\ S_n \end{bmatrix} := G^* G.$$

Matrix sizes

- $H_k, S_k \in \mathbb{C}^{(2N_L) \times N_G}$, $T_k \in \mathbb{C}^{(2N_L) \times (2N_L)}$,
- $F, G \in \mathbb{C}^{(2N_A N_L) \times N_G}$, $T \in \mathbb{C}^{(2N_A N_L) \times (2N_A N_L)}$.

Transform the problem!

Make T simpler

- the method can be applied even on already described matrices $\widetilde{F}$, G and T implicitely, but multiplication by T is slow
- T should be either factored (for example by using somewhat modified Hermitian indefinite factorization), or diagonalized (simultaneous diagonalization of T_ks) — diagonalization is too slow
- therefore, H is written as

$$H := F^* J F, \qquad J = \mathrm{diag}(\pm 1).$$

The complex Hari–Zimmermann method for the GEP

One-sided vs. two-sided method

- the original Hari–Zimmerman method works from both sides on the Hermitian matrix pair
- the modified method works from one side on the factors of the Hermitian matrix pair
- idea: think two-sided, act one-sided
- transformations will be computed from the pivot submatrices H_{pq} of H and S_{pq} of S

$$H_{pq} = \begin{bmatrix} F_p^* J F_p & F_p^* J F_q \\ & F_q^* J F_q \end{bmatrix}, \quad S_{pq} = \begin{bmatrix} G_p^* G_p & G_p^* G_q \\ & G_q^* G_q \end{bmatrix}.$$

The complex Hari–Zimmermann method for the GEP

The method consists of 3 transformations (Hari)

- as a preprocessing step H and S can be scaled by the diagonal matrix D such that $\operatorname{diag}(DSD) = I$

$$H_0 := DHD, \quad S_0 := DSD,$$
$$D = \operatorname{diag}\left(\frac{1}{\sqrt{s_{11}}}, \frac{1}{\sqrt{s_{22}}}, \ldots, \frac{1}{\sqrt{s_{nn}}}\right)$$

- in the first step the pivot submatrix $\widehat{S}_0$ of S_0 is diagonalized by the complex rotation

$$\widehat{R}_1 = \begin{bmatrix} \cos\varphi_1 & e^{i\beta_1}\sin\varphi_1 \\ -e^{-i\beta_1}\sin\varphi_1 & \cos\varphi_1 \end{bmatrix},$$

The complex Hari–Zimmermann method for the GEP

The transformations

- the first transformation is

$$H_1 = R_1^* H_0 R_1, \quad S_1 = R_1^* S_0 R_1,$$

  $R_1 = I$ except at the pivot positions, where $R_1 = \widehat{R}_1$.
- if H and S are preprocessed, then $\varphi_1 = -\frac{\pi}{4}$
- in the second step – the diagonal of S_1 is rescaled to I
- this transformation is similar to the preprocessing step

$$H_2 := D_2 H_1 D_2, \quad S_2 := D_2 S_1 D_2.$$

The transformations

- in the third step the pivot submatrix $\widehat{H}_2$ of H_2 is diagonalized by the complex rotation

$$\widehat{R}_3 = \begin{bmatrix} \cos\varphi_3 & e^{i\alpha_3}\sin\varphi_3 \\ -e^{-i\alpha_3}\sin\varphi_3 & \cos\varphi_3 \end{bmatrix},$$

- the third transformation is

$$H_3 = R_3^* H_2 R_3, \quad S_3 = R_3^* S_2 R_3,$$

  $R_3 = I$ except at the pivot positions, where $R_3 = \widehat{R}_3$.
- if H and S are preprocessed, then $\varphi_3 = \vartheta + \frac{\pi}{4}$.

The transformations

- note that after the first three steps, the pivot submatrix $\widehat{S}_3$ is still diagonal (in fact identity)

$$\widehat{S}_3 = \widehat{Z}^* \widehat{S} \widehat{Z}, \quad \widehat{Z} = \widehat{R}_1 \widehat{D}_2 \widehat{R}_3$$

- if H and S are preprocessed, the fourth step is only formal — it helps in coupling together all the transformations

$$H_4 = \Phi_4^* H_3 \Phi_4, \quad S_4 = \Phi_4^* S_3 \Phi_4, \quad \widehat{\Phi}_4 = \mathrm{diag}(e^{-i\sigma_p}, e^{-i\sigma_q}).$$

The complex Hari–Zimmermann method for the GEP

The coupled transformation Z ...

- looks similar to an ordinary plane rotation: it is the identity matrix, except for its (p, q)-restriction $\widehat{Z}$, where

$$\widehat{Z} = \frac{1}{\sqrt{1 - \left(|s_{pq}|\right)^2}} \begin{bmatrix} \cos\varphi & e^{i\alpha}\sin\varphi \\ -e^{-i\beta}\sin\psi & \cos\psi \end{bmatrix},$$

- φ and ψ are determined so that the transformations diagonalize the pivot submatrices $\widehat{H}$ and $\widehat{S}$
- the transformation keeps the diagonal elements of S intact
- if $S = I$ then Z is the ordinary rotation, the method is the ordinary Jacobi method for a single matrix.

The Hari–Zimmermann method for the GEP

Computation of the elements of $\widehat{Z}$

- let

$$s = |s_{pq}|, \quad t = \sqrt{1 - s^2}, \quad r = s_{qq} - s_{pp},$$

$$\sigma = \begin{cases} 1 & e \geq 0 \\ -1 & e < 0, \end{cases}, \quad u + iv = e^{-i \arg(s_{pq})} h_{pq},$$

- then if $\gamma = \alpha - \beta$

$$\tan(\gamma) = 2\frac{v}{r}, \qquad -\frac{\pi}{2} \leq \gamma \leq \frac{\pi}{2}$$

$$\tan(2\vartheta) = \sigma \frac{2u - (h_{pp} + h_{qq})s}{\sqrt{e^2 + 4v^2} \cdot t}, \qquad -\frac{\pi}{4} < \vartheta \leq \frac{\pi}{4}$$

Computation of the elements of $\widehat{Z}$

- and

$$2\cos^2\varphi = 1 + s\sin(2\vartheta) + t\cos(2\vartheta)\cos(\gamma), \quad 0 \le \varphi < \frac{\pi}{2}$$

$$2\cos^2\psi = 1 - s\sin(2\vartheta) + t\cos(2\vartheta)\cos(\gamma), \quad 0 \le \psi < \frac{\pi}{2}$$

$$e^{i\alpha}\sin(\varphi) = \frac{(\sin(2\vartheta) - s) + i\sqrt{1-s^2}\sin(\gamma)\cos(2\vartheta)}{1 - s\sin(2\vartheta) + \sqrt{1-s^2}\cos(\gamma)\cos(2\vartheta)}$$

$$e^{-i\beta}\sin(\psi) = \frac{(\sin(2\vartheta) + s) - i\sqrt{1-s^2}\sin(\gamma)\cos(2\vartheta)}{1 + s\sin(2\vartheta) + \sqrt{1-s^2}\cos(\gamma)\cos(2\vartheta)}.$$

The pointwise algorithm

The implicit HZ algorithm

$Z = I$; $\quad it = 0$

`repeat` // sweep loop

$it = it + 1$

`for` all pairs (p, q), $1 \le p < q \le k$

compute

$$\widehat{H} = \begin{bmatrix} f_p^* J f_p & f_p^* J f_q \\ & f_q^* J f_q \end{bmatrix}; \qquad \widehat{S} = \begin{bmatrix} g_p^* g_p & g_p^* g_q \\ & g_q^* g_q \end{bmatrix}$$

compute the elements of $\widehat{Z}$

// transform F, G and Z

$[f_p, f_q] = [f_p, f_q] \cdot \widehat{Z}$

$[g_p, g_q] = [g_p, g_q] \cdot \widehat{Z}$

$[z_p, z_q] = [z_p, z_q] \cdot \widehat{Z}$

`until` (no transf. in this sweep) `or` ($it \ge maxcyc$)

Hardware platform

Developer Edition of the Intel Xeon Phi 7210 (KNL) processor

- 96 GB of RAM per node,
- 64 cores per node,
- clock 1.30 GHz (Turbo Boost off),
- Intel AVX-512 (Advanced Vector Extensions) instruction set
- presence of two vector processing units (VPUs) per core — each VPU operates independently on 512-bit vector registers – suitable for simultaneous processing of 16 `single precision` or 8 `double precision` numbers.

The first step

Hermitian indefinite factorization of all T_k's

- `for all` T_k`'s do in parallel`

$$T_k = P_k^T R_k^* D_k R_k P_k,$$

P_k is a permutation matrix – formed as in LAPACK
(as a sequence of partial permutations),
D_k is block diagonal, with 1×1 or 2×2 diagonal blocks,
R_k is upper triangular

- `diagonalize all` D_k`'s in parallel`

$$D_k = U_k^* \Delta_k U_k = U_k^* \sqrt{|\Delta_k|} J_k \sqrt{|\Delta_k|} U_k,$$

Δ_k diagonal, U_k block-diagonal, unitary, $J_k = \mathrm{diag}(\pm 1)$,

The first step (cnt'd)

- for all J_k repermute them in parallel

$$J_k := \widetilde{P}_k^T \operatorname{diag}(I, -I)\widetilde{P}_k$$

  $\widetilde{P}_k$ is a permutation,

- multiply rows of all R_k and repermute them

$$R_k = \widetilde{P}_k \sqrt{|\Delta_k|} U_k$$

- repremute columns of all R_k according to pemutations stored in P_k

$$R_k := R_k P_k.$$

The first step (cnt'd)

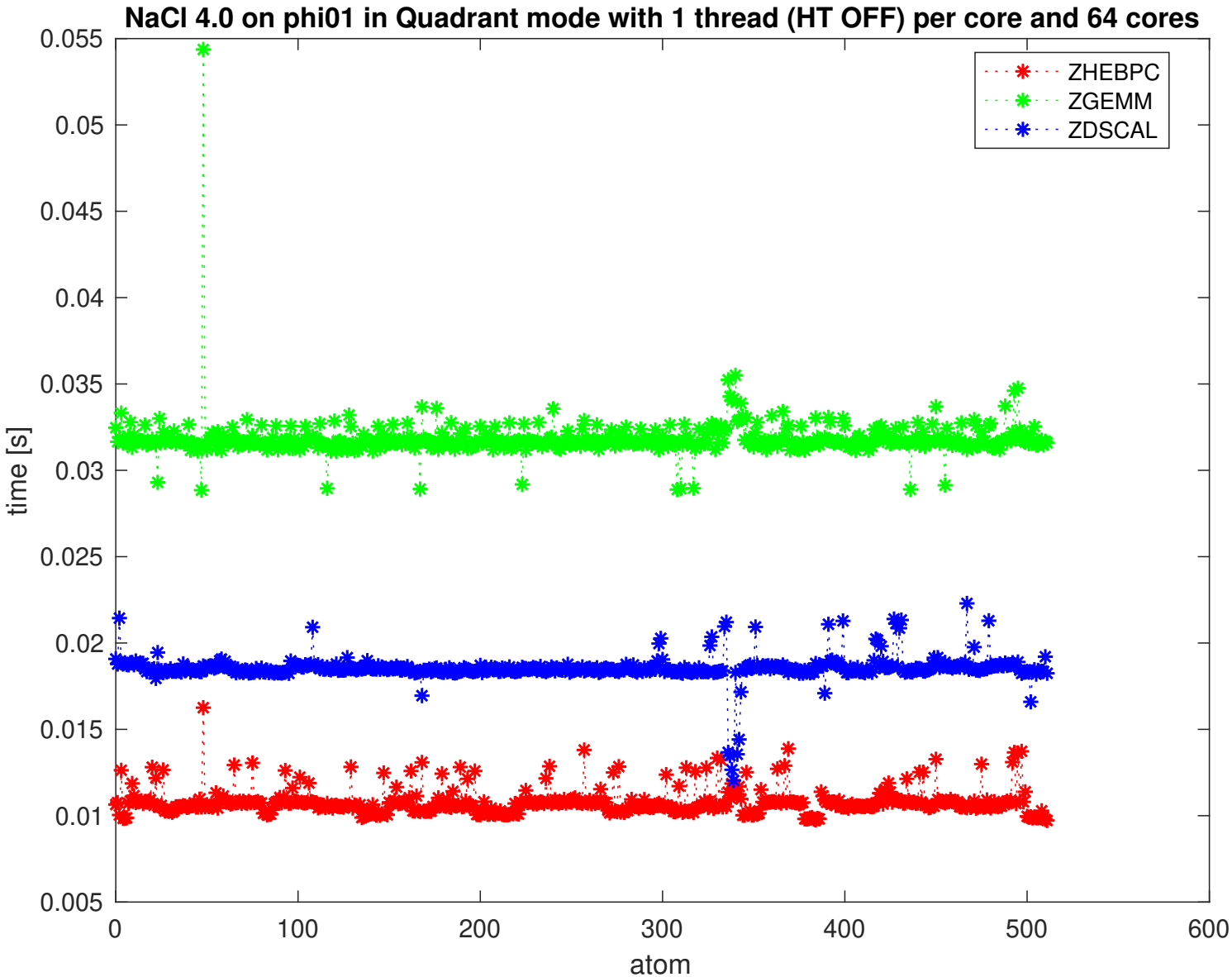
Final state

$$T_k = R_k^* \,\mathrm{diag}(I, -I) R_k, \quad k = 1, \ldots, n.$$

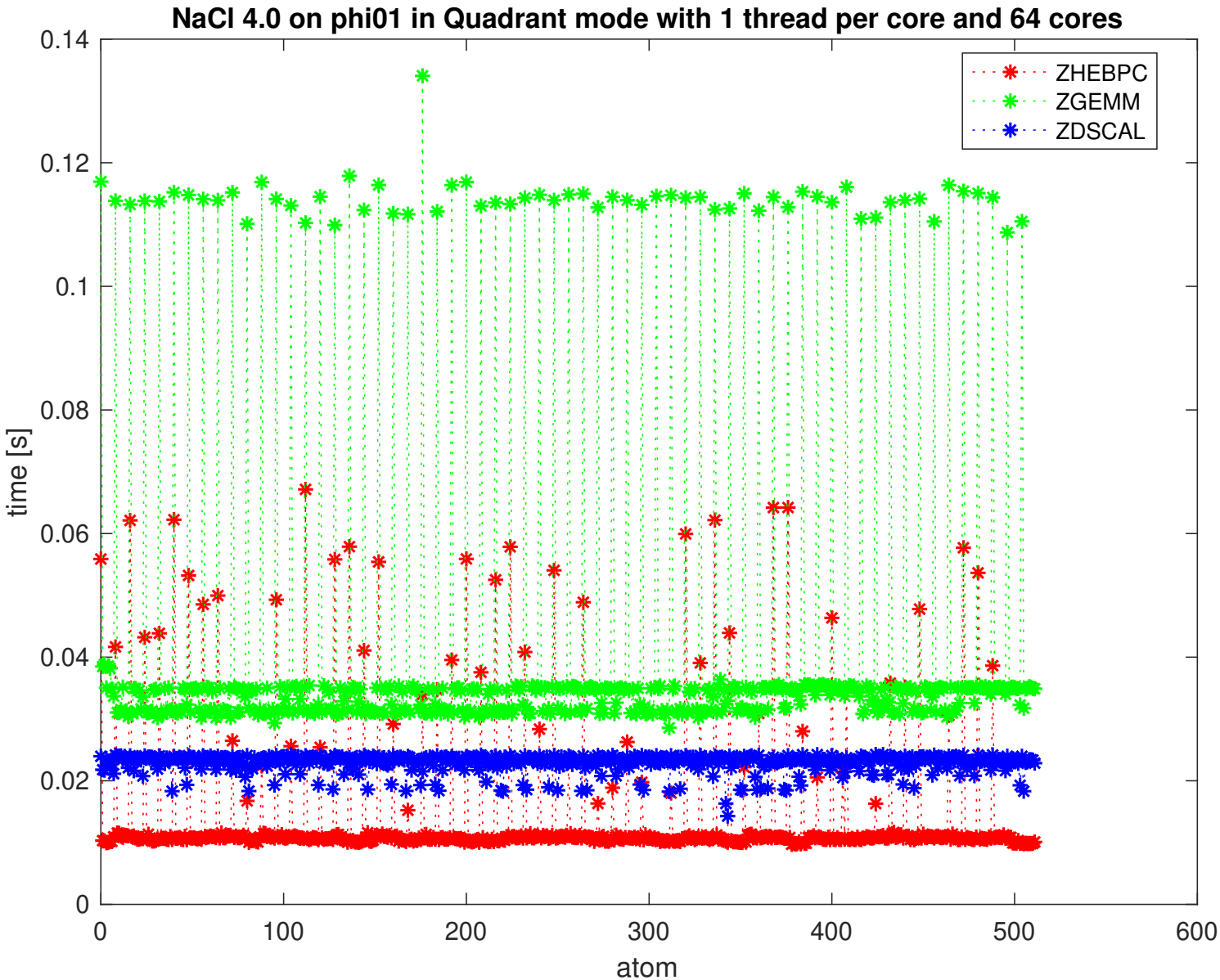
Comments

- in the first step, the factorization is sequential for each T_k
- each physical core of the Xeon Phi deals with one or more T_k in turn (OpenMP `parallel do` over all T_k)
- each core can use its own 1–4 hyperthreads in a call of the threaded BLAS routines – therefore even per core algoritm is somewhat parallel – but do not use hyperthreading, since...

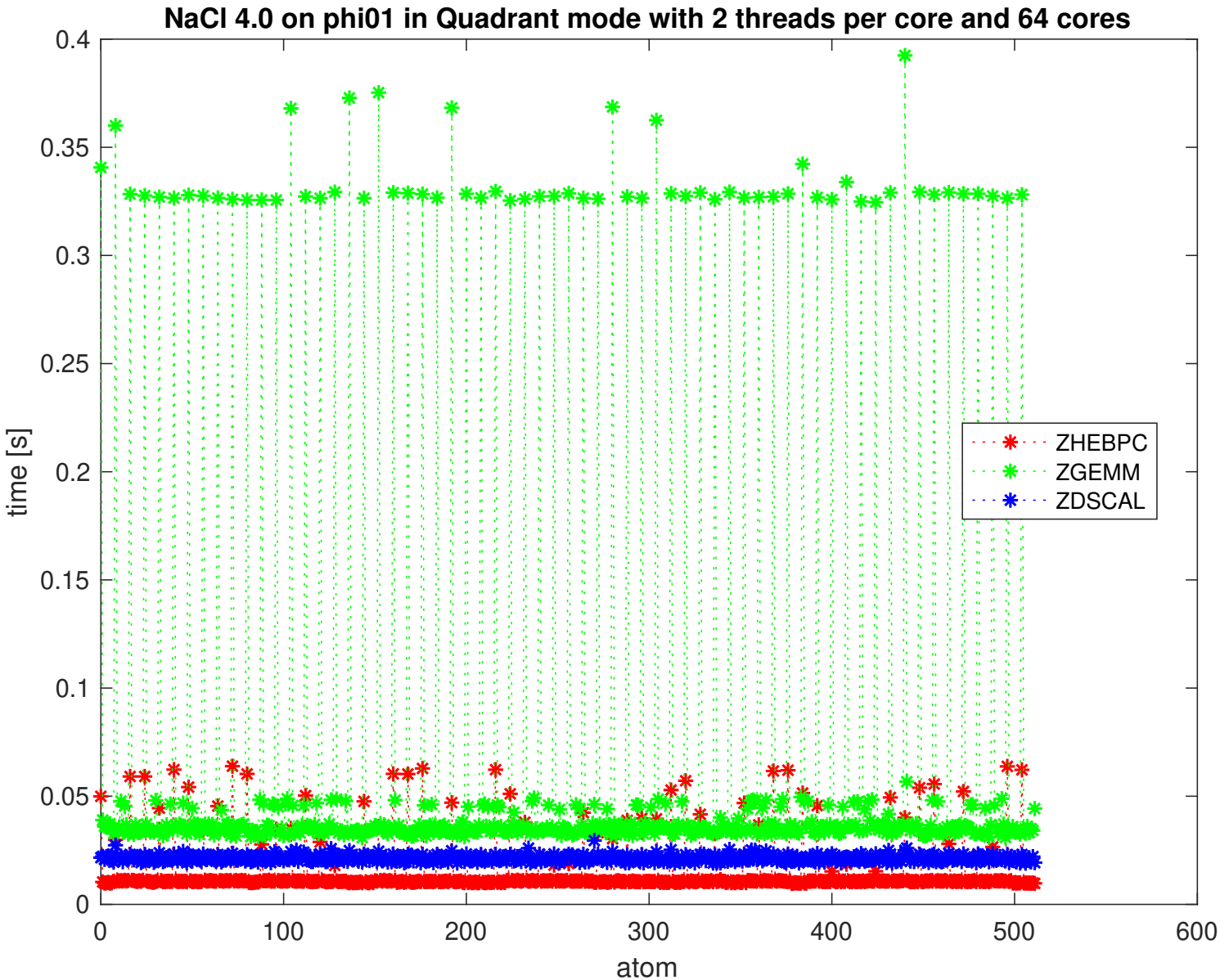
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



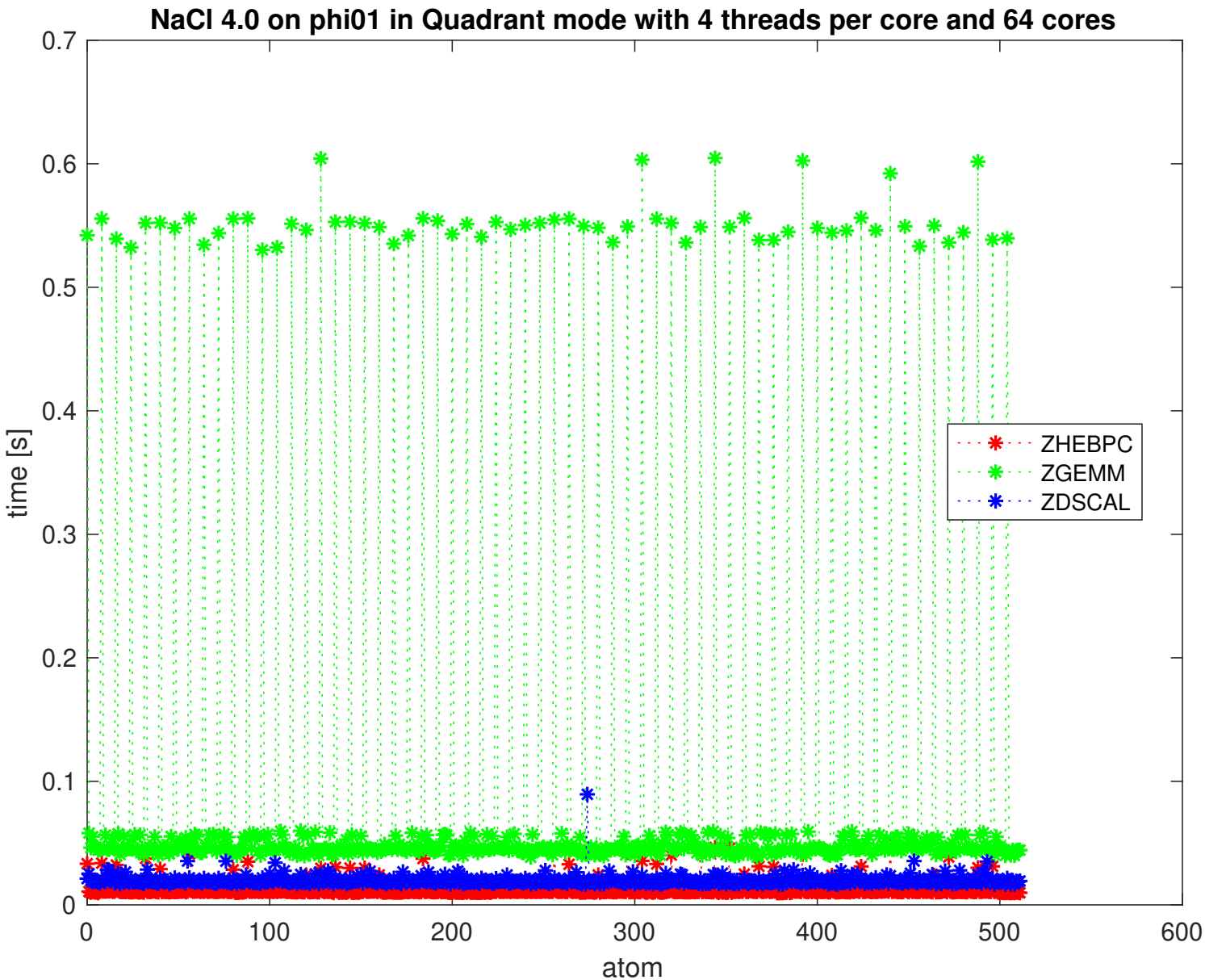
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



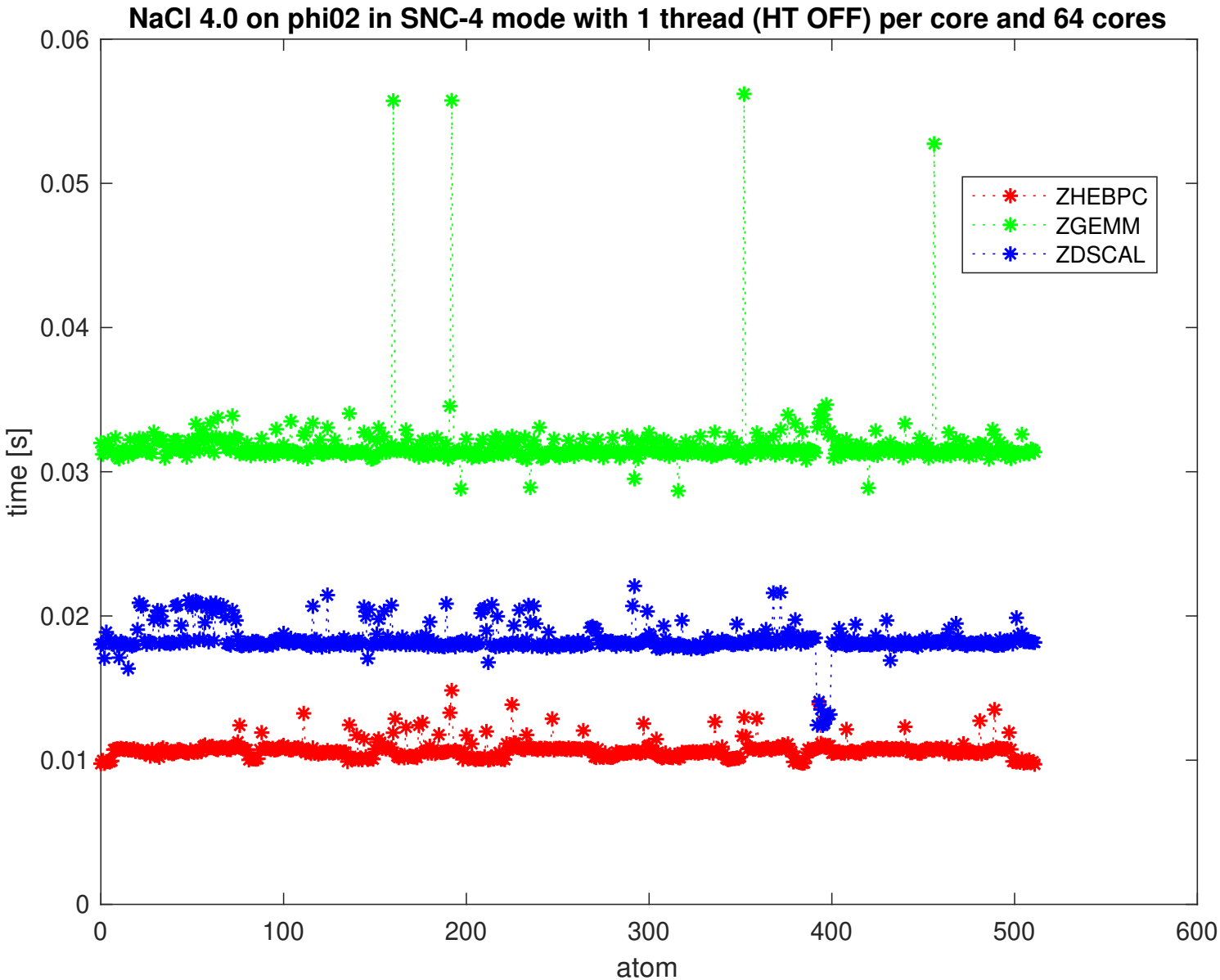
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



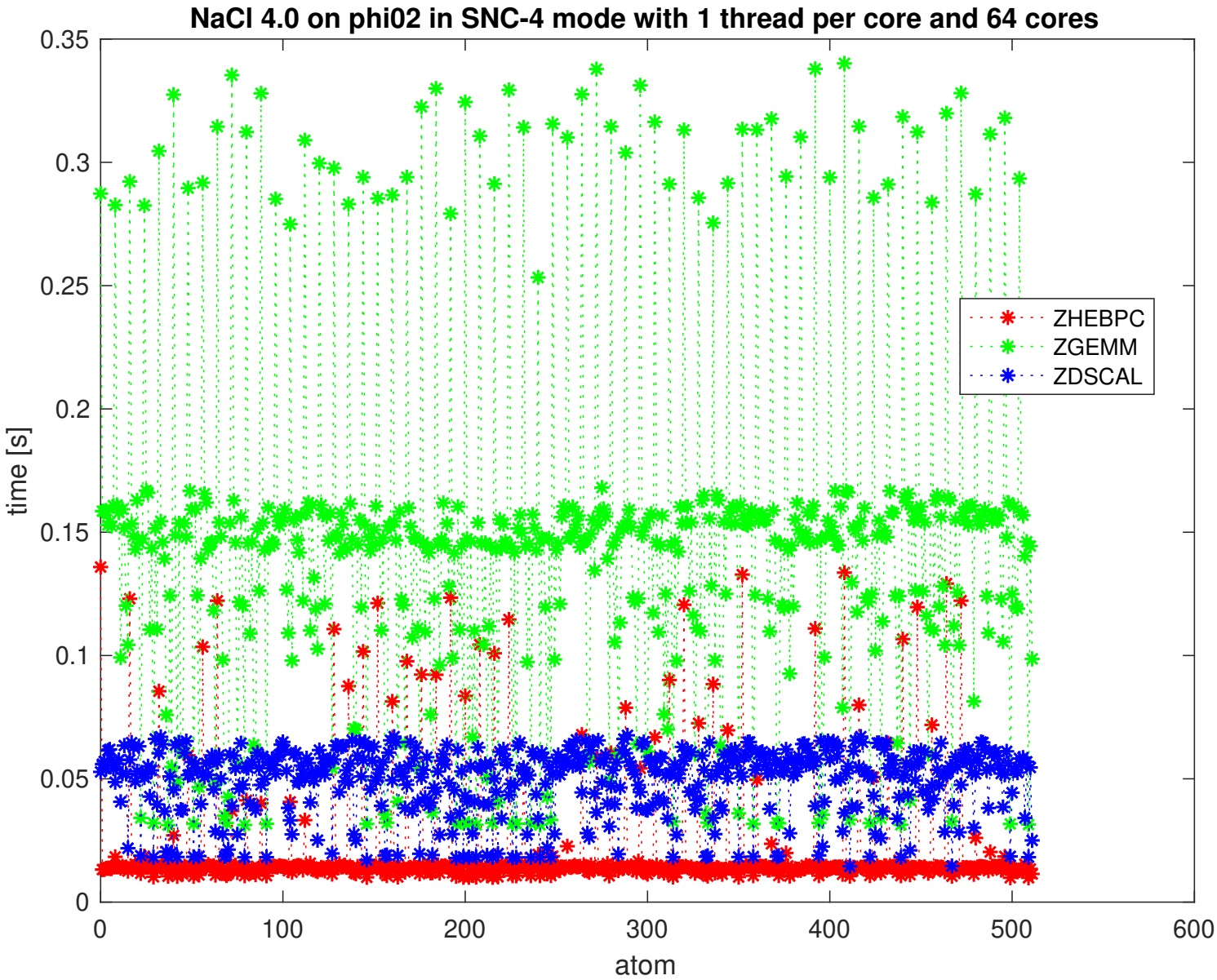
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



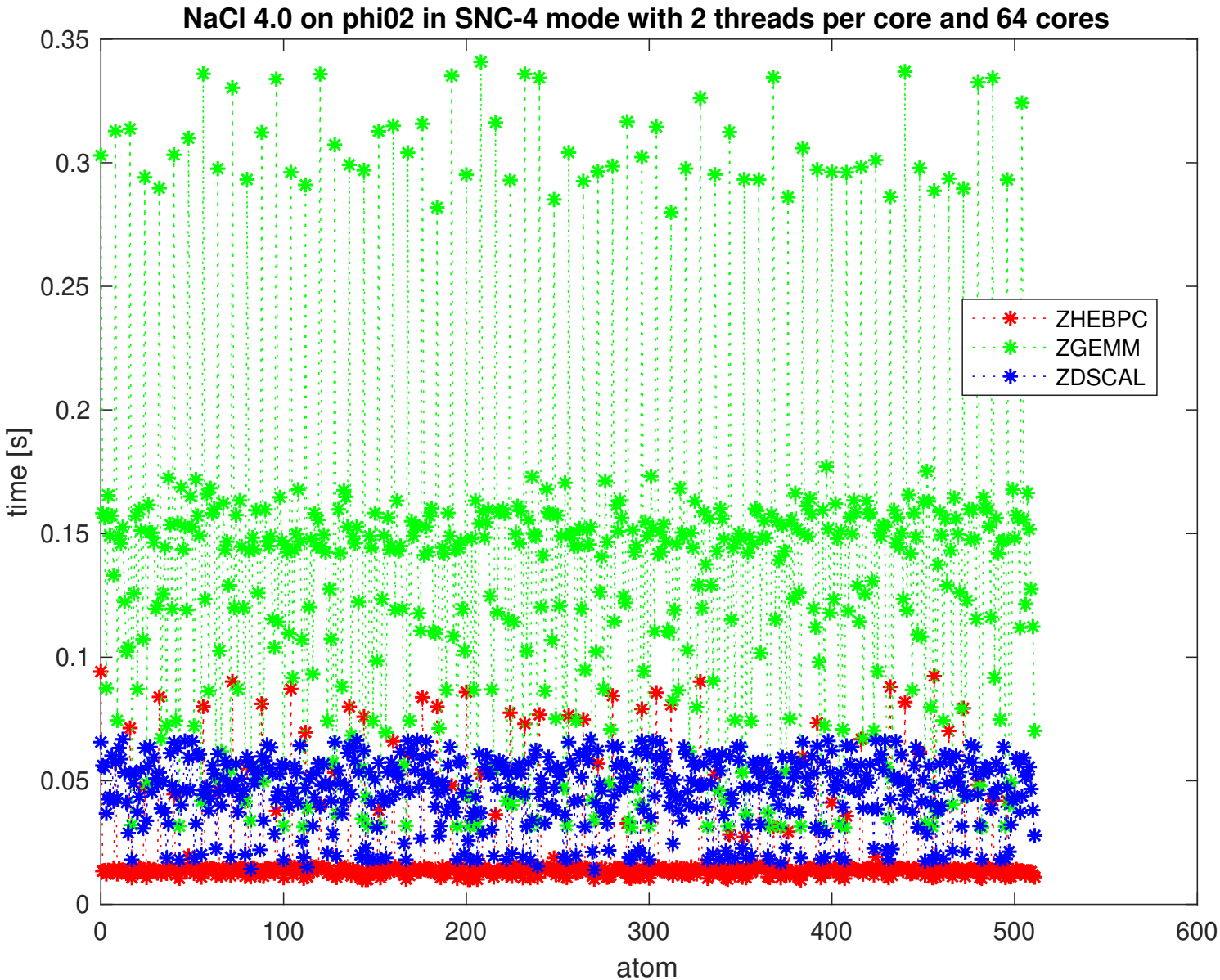
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



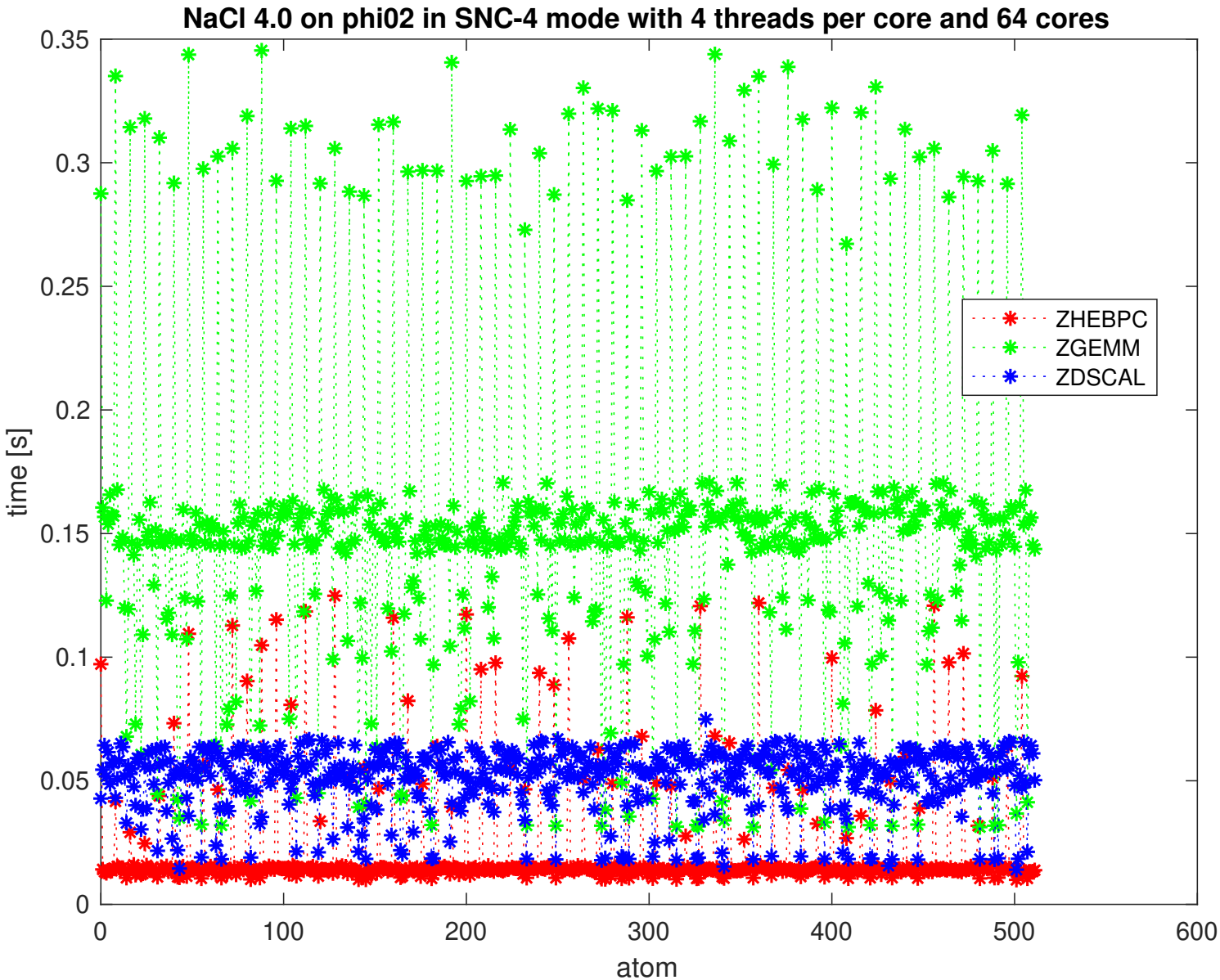
1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



1, 2 or 4 threads, NaCl, $N_L = 49$, $N_A = 512$, $N_G = 9273$



The second and the optional step

Form J, F and G

- `store` $J = \mathrm{diag}(J_1, \ldots, J_n)$,
- `multiply` $F = \mathrm{diag}(R_1, \ldots, R_n)F$, `each` R_k `in parallel`
- `scale` B_k `by` U_k `in parallel and store` G

Optional step — make J, F and G square

- square matrix – faster HZ algorithm
- this step is the hyperbolic QR factorization on F and the QR factorization on G – both algorithms moderately parallel
- pivoting strategy – partial pivoting?, threshold pivoting?
- usage of (block)-reflectors or (block)-rotations?
- do it or not – depends on the ratio (number of rows) / (number of columns) of F and G

Driver level of the implicit HZ algorithm

Details of the level-2 algorithm

- algorithm is Generalized Hyperbolic SVD of (F, G) with respect to J
- matrices F and G are divided in even number of block-columns

$$F = [F_1, \ldots, F_{2b}], \quad G = [G_1, \ldots, G_{2b}]$$

- number of block-columns depend on the number of physical cores of the processor (our case: 64 cores = maximum 128 blocks, no hyperthreading)
- each thread is connected to one physical core.

Driver level of the implicit HZ algorithm

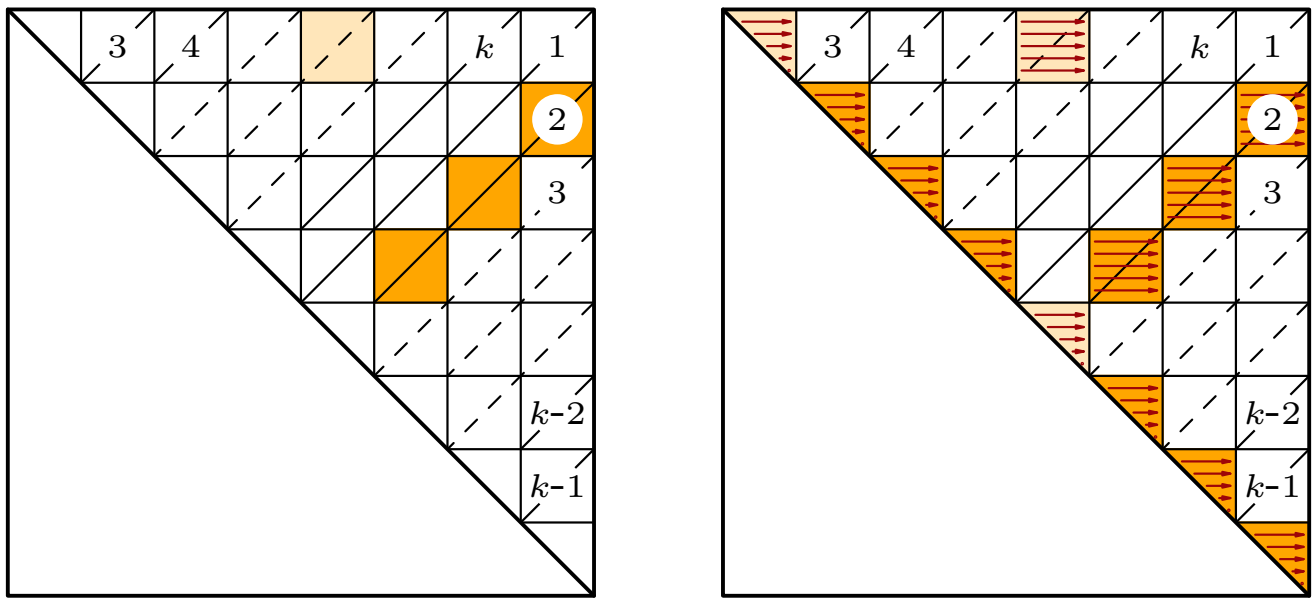
Each thread . . .

- works on a pair of block-columns of each matrix given by some parallel pivot strategy
- allocates storage for $[F_p, F_q]$, $[G_p, G_q]$, their "shadow" counterparts, and for the part of the transformation matrix
- "shadow" memory — used for scaling by J_k and data exchange
- since architecture is NUMA (Non Uniform Memory Access), columns are also physically copied to "shadow" memory (alternative: reassignment of pointers)
- allocates square space in fast MCDRAM for computation of the transformation Z_{pq} and the pivot block submatrices H_{pq} and S_{pq}.

Pivoting strategy

Parallel pivoting strategy

- Choose pivot blocks independently in each step, for example, by using (block)-modulus strategy (not optimal!)



- stopping criterion
  - skip a transformation if cosines are 1
  - final stop — all transformations are skipped.

Driver level of the implicit HZ algorithm

Each thread . . .

- actually computes H_{pq} and S_{pq} (`ZGEMM` and `ZHERK`)
- factorizes H_{pq} and S_{pq} by the Hermitian indefinite factorization (test of definiteness of S_{pq})

$$H_{pq} = F_{pq}^{*} J_{pq} F_{pq}, \quad S_{pq} = G_{pq}^{*} G_{pq},$$

  where F_{pq}, G_{pq}, and J_{pq} are square
- calls level-1 (non-blocked routine) on the triplet (F_{pq}, G_{pq}, J_{pq})
- applies transformation matrix to original F_{pq}, G_{pq}, and Z_{pq} (`ZGEMM`s)
- transfers one triplet of $(F_{\ell}, G_{\ell}, Z_{\ell})$, $\ell \in \{p, q\}$ to the next "owner" (thread) into its "shadow" memory.

Computational level of the implicit HZ algorithm

Details of the level-1 algorithm

- single-threaded (including BLAS calls) SIMD-parallel code,
- the main loop — `sweep` iterations (1, m, until convergence)
- parallel pivot strategy determines maximal number of independent pivot pairs — stage of the algorithm
- in each stage — pairs are divided into groups of 8 pairs (AVX-512 instructions)
- compute 6 dot products (vectorized + reductions) with only 4 accesses of f_p, f_q, g_p, and g_q:

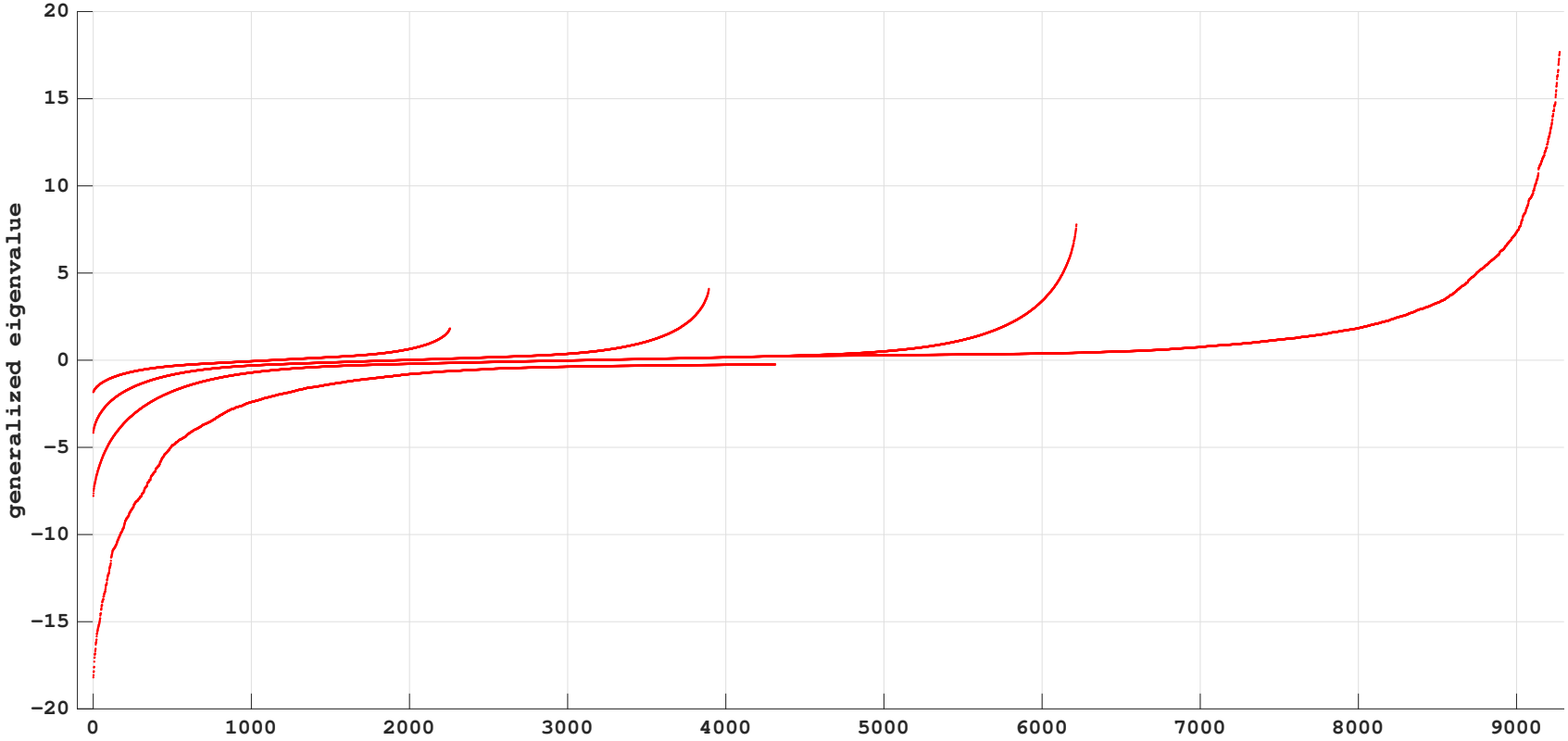
$$\widehat{H}_{pq} = \begin{bmatrix} f_p^* J_p f_p & f_p^* J_q f_q \\ & f_q^* J_q f_q \end{bmatrix}, \qquad \widehat{S}_{pq} = \begin{bmatrix} g_p^* g_p & g_p^* g_q \\ & g_q^* g_q \end{bmatrix},$$

Computational level of the implicit HZ algorithm

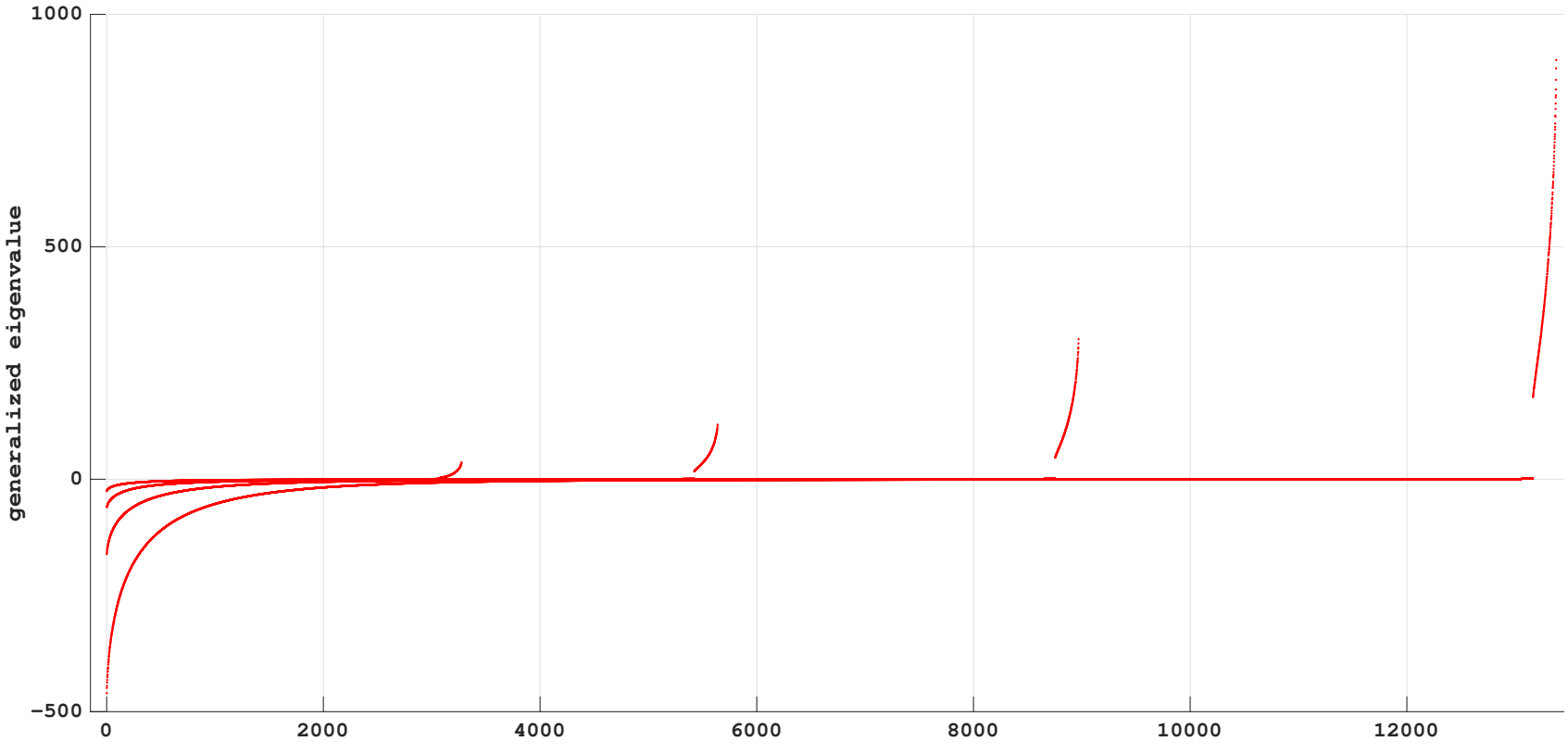
Details of the level-1 algorithm

- an example: the dot products are computed without BLAS to avoid function calls (slow!)
- computing transformation matrices for 8 pairs simultaneously
- transformations to 8 column pairs (f_p, f_q), (g_p, g_q), (z_p, z_q) are applied sequentially for each pair (cache!)

Distribution of eigenvalues – NaCl



Distribution of eigenvalues – AuAg



Timings

Example	Problem size	Number of cores 64	32
NaCl 2.5	50176 × 2256	800.70	556.95
NaCl 3.0	50176 × 3893	1973.64	1465.68
NaCl 3.5	50176 × 6217	2810.50	3660.44
NaCl 4.0	50176 × 9273	4846.98	7028.50
AuAg 2.5	26136 × 3275	724.20	587.23
AuAg 3.0	26136 × 5638	1549.92	1715.60
AuAg 3.5	26136 × 8970	3152.78	4711.65
AuAg 4.0	26136 × 13379	6544.16	11955.74

Conclusion

On a particular hardware testing space is enormous

- use Quadrant or SNC-4 clustering mode?
- in a single step — transform columns only once (block-oriented algorithm) or fully diagonalize them (full block algorithm)
- best pivoting strategy?
- is there need to shorten the columns by the hyperbolic QR factorization, and is there a switching point (use them or not)

  ...

Work in progress

- only lower 20% of the eigenvalues are needed
- is there any sufficiently parallel algorithm to compute them (without multiplication of the factors)?