A Projected Preconditioned Conjugate Gradient algorithm for computing a large invariant subspace of a Hermitian matrix

Eugene Vecharynski
Lawrence Berkeley National Laboratory

joint work with
Chao Yang (LBNL) and John E.Pask (LLNL)

Scientific Computing and Matrix Computations Seminar
UC Berkeley, October 22, 2014

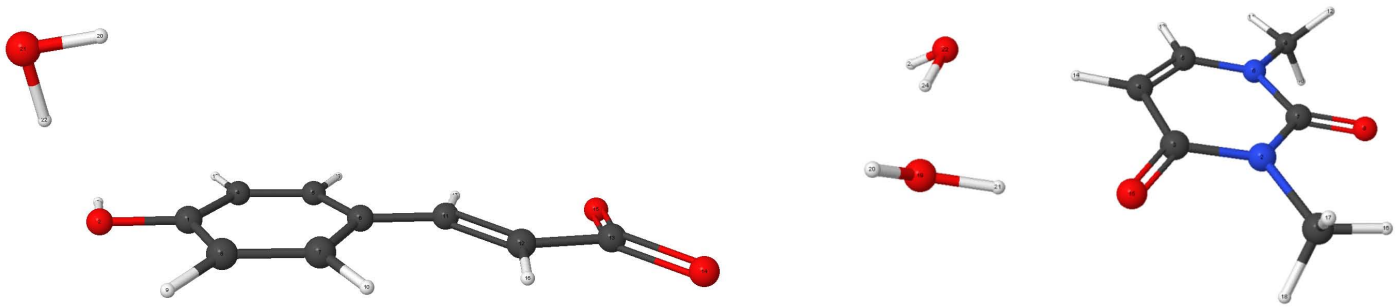
Outline

- Motivation, the problem, existing approaches
- The Projected Preconditioned Conjugate Gradient (PPCG) algorithm for computing LARGE invariant subspaces
- Computational examples
- Conclusions and future work

Work supported by DOE SciDAC projects.

Motivation

Extreme-scale electronic structure calculations



- Ground and excited states of a quantum many-body system
- Density functional theory based electronic structure calculations requires several lowest eigenpairs
- Density functional perturbation theory often requires many more lowest eigenpairs

HUGE eigenvalue problems

Find a subset of lowest eigenpairs (λ, x) of

$$Ax = \lambda x, \quad A = A^*.$$

- The problem size n is well beyond 10^6
- A small fraction of lowest eigenpairs wanted ($\approx 1\%$)
- The number of targeted eigenpairs can be on the order of 10^3–10^5
- Solutions of the eigenvalue problem are computed repeatedly after slight perturbations of A (SCF loop)

HUGE eigenvalue problems

Find a subset of lowest eigenpairs (λ, x) of

$$Ax = \lambda x, \quad A = A^*.$$

- The problem size n is well beyond 10^6
- A small fraction of lowest eigenpairs wanted ($\approx 1\%$)
- The number of targeted eigenpairs can be on the order of 10^3–10^5
- Solutions of the eigenvalue problem are computed repeatedly after slight perturbations of A (SCF loop)

Need powerful eigensolvers that compute LARGE eigenspaces!

Modern eigensolvers

Desirable features:

- Block iterations
- Preconditioning
- Low memory requirement
- Simplicity of implementation

State-of-the art methods:

- Block Davidson ('75)
- LOBPCG (Knyazev '01)
- Jacobi-Davidson (Sleijpen-V. der Vorst '96)
- TRACEMIN (Sameh-Wisniewski '82)

Modern eigensolvers

Desirable features:

- Block iterations
- Preconditioning
- Low memory requirement
- Simplicity of implementation

State-of-the art methods:

- Block Davidson ('75)
- LOBPCG (Knyazev '01)
- Jacobi-Davidson (Sleijpen-V. der Vorst '96)
- TRACEMIN (Sameh-Wisniewski '82)

Davidson-Liu (Preconditioned Steepest Descent) algorithm

Given an initial guess $X^{(0)}$ and a preconditioner M, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}$
- **While** convergence not reached
  - $W \leftarrow M(AX - X(X^*AX))$
  - $S \leftarrow [X, W]$
  - Find eigenvectors C associated with the k smallest eigenvalues of $(S^*AS)C = (S^*S)C\Omega$, $C^*(S^*S)C = I$
  - $X \leftarrow SC$
- **EndWhile**

Davidson-Liu (Preconditioned Steepest Descent) algorithm

Given an initial guess $X^{(0)}$ and a preconditioner M, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}$
- **While** convergence not reached
  - $W \leftarrow M(AX - X(X^*AX))$
  - $S \leftarrow [X, W]$
  - Find eigenvectors C associated with the k smallest eigenvalues of $(S^*AS)C = (S^*S)C\Omega,\ C^*(S^*S)C = I$
  - $X \leftarrow SC$
- **EndWhile**

The Rayleigh–Ritz (RR) procedure

At each step, RR leads to the projected dense eigenproblem

$$(S^*AS)C = (S^*S)C\Omega, \quad C^*(S^*S)C = I$$

- Size of the projected problem is proportional to k ($2k$ for Davidson-Liu, or $3k$ for LOBPCG)

The Rayleigh–Ritz (RR) procedure

At each step, RR leads to the projected dense eigenproblem

$$(S^*AS)C = (S^*S)C\Omega, \quad C^*(S^*S)C = I$$

- Size of the projected problem is proportional to k ($2k$ for Davidson-Liu, or $3k$ for LOBPCG)
- Complexity of dense eigensolver scales as k^3

The Rayleigh–Ritz (RR) procedure

At each step, RR leads to the projected dense eigenproblem

$$(S^*AS)C = (S^*S)C\Omega, \quad C^*(S^*S)C = I$$

- Size of the projected problem is proportional to k ($2k$ for Davidson-Liu, or $3k$ for LOBPCG)
- Complexity of dense eigensolver scales as k^3
- Limited parallel scalability of existing kernels for dense eigenproblems (e.g., ScaLAPACK)

The Rayleigh–Ritz (RR) procedure

At each step, RR leads to the projected dense eigenproblem

$$(S^*AS)C = (S^*S)C\Omega, \quad C^*(S^*S)C = I$$

- Size of the projected problem is proportional to k ($2k$ for Davidson-Liu, or $3k$ for LOBPCG)
- Complexity of dense eigensolver scales as k^3
- Limited parallel scalability of existing kernels for dense eigenproblems (e.g., ScaLAPACK)
- The RR becomes a computational bottleneck in practice

The Rayleigh–Ritz (RR) procedure

At each step, RR leads to the projected dense eigenproblem

$$(S^*AS)C = (S^*S)C\Omega, \quad C^*(S^*S)C = I$$

- Size of the projected problem is proportional to k ($2k$ for Davidson-Liu, or $3k$ for LOBPCG)
- Complexity of dense eigensolver scales as k^3
- Limited parallel scalability of existing kernels for dense eigenproblems (e.g., ScaLAPACK)
- The RR becomes a computational bottleneck in practice
- Our idea: Entirely remove or reduce the number of the RR calculations!

The RR cost

Computation	Time (sec.)
GEMM	11
AX	6
RR	66

Table: The timing profiles for Davidson–Liu to compute $\approx 2,000$ eigenpairs on 480 cores.

Existing "RR-avoiding" techniques

- Spectrum slicing (Schofield–Chelikowsky–Saad '12)
  - Break the spectrum into subintervals
  - Use polynomial filters to compute interior eigenvalues in different subintervals simultaneously
  - Orthogonalize whenever necessary
  - Available in PARSEC software

Dividing spectrum is not trivial. Preconditioning is not used.

Existing "RR-avoiding" techniques

- Spectrum slicing (Schofield–Chelikowsky–Saad '12)
  - Break the spectrum into subintervals
  - Use polynomial filters to compute interior eigenvalues in different subintervals simultaneously
  - Orthogonalize whenever necessary
  - Available in PARSEC software

  Dividing spectrum is not trivial. Preconditioning is not used.
- Penalized trace minimization (Wen–Yang–Liu-Zhang '13)
  - Based on the unconstrained formulation

$$\min_X \operatorname{trace}(X^* A X) + \mu \|X^* X - I\|_F^2$$

  - Steepest descent with Barzilai-Borwein line search
  - RR, orthonormalization every once a while

  The penalty parameter μ may be hard to estimate.

Projected gradient methods: the general framework

- Elegant approach for solving constrained optimization problems: Rosen '60-61, Goldstein '64, Levitin–Polyak '66

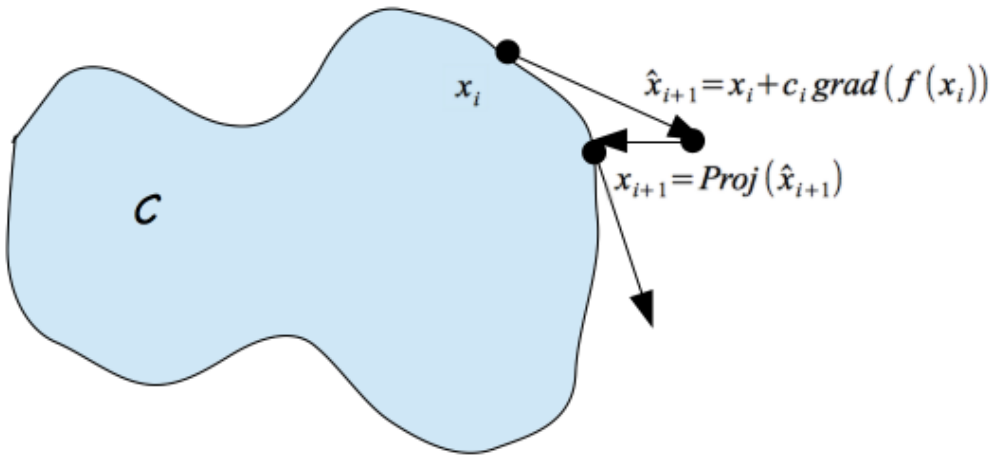
$$\min_x f(x), \quad x \in \mathcal{C}$$

Projected gradient methods: the general framework

- Elegant approach for solving constrained optimization problems: Rosen '60-61, Goldstein '64, Levitin–Polyak '66

$$\min_x f(x), \quad x \in \mathcal{C}$$

- In a nutshell: Skip the constraint, make a step in the gradient direction, project onto the constraints



Convergence theory exists for classes of constraints

Projected gradient methods: the eigenvalue problem

- Block Rayleigh Quotient

$$f(X) = \text{trace}(X^*X)^{-1}(X^*AX), \quad X \in \mathcal{C}^{n\times k}$$

Projected gradient methods: the eigenvalue problem

- Block Rayleigh Quotient

$$f(X) = \text{trace}(X^*X)^{-1}(X^*AX), \quad X \in \mathcal{C}^{n\times k}$$

- The minimization problem

$$\min_X f(X), \quad X \in \mathcal{C} = \{X \ : \ X^*X = I\}$$

Projected gradient methods: the eigenvalue problem

- Block Rayleigh Quotient

$$f(X) = \text{trace}(X^*X)^{-1}(X^*AX), \quad X \in \mathcal{C}^{n\times k}$$

- The minimization problem

$$\min_X f(X), \quad X \in \mathcal{C} = \{X \ : \ X^*X = I\}$$

- The preconditioned gradient direction for $f(X)$

$$W = M(AX - X(X^*AX)), \quad X \in \mathcal{C}$$

Projected gradient methods: the eigenvalue problem

- Block Rayleigh Quotient

$$f(X) = \text{trace}(X^*X)^{-1}(X^*AX), \quad X \in \mathcal{C}^{n\times k}$$

- The minimization problem

$$\min_X f(X), \quad X \in \mathcal{C} = \{X \ : \ X^*X = I\}$$

- The preconditioned gradient direction for $f(X)$

$$W = M(AX - X(X^*AX)), \quad X \in \mathcal{C}$$

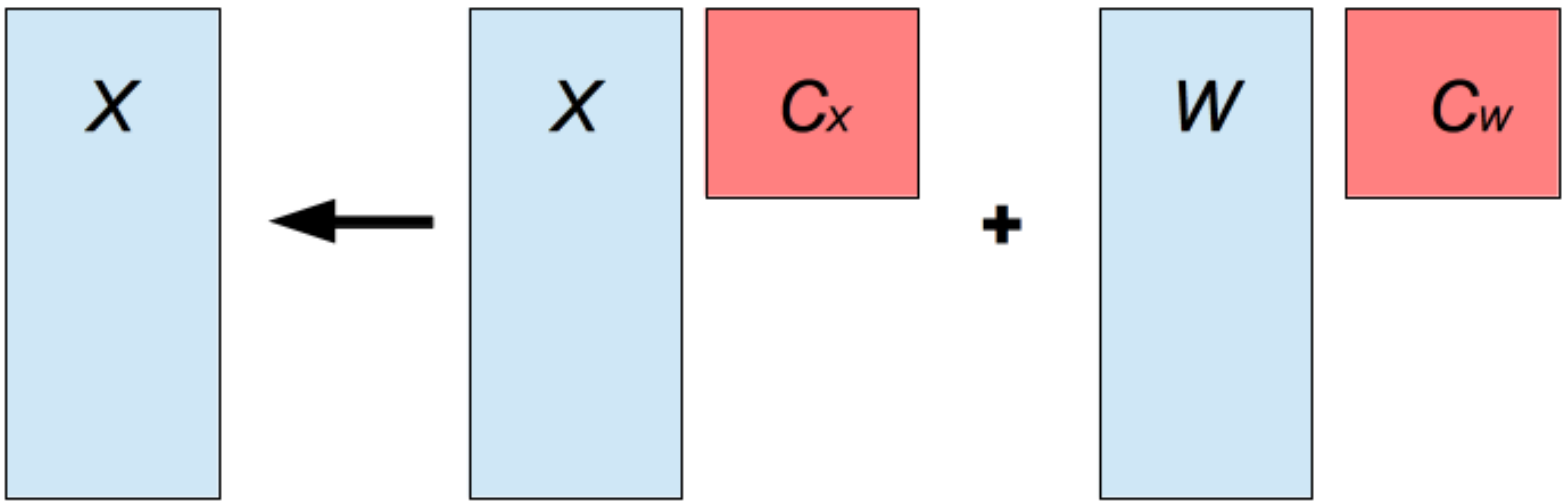
- The Preconditioned Steepest Descent (PSD) or Davidson–Liu

$$X \leftarrow XC_X + WC_W,$$

where C_X and C_W are k-by-k iteration coefficient matrices.

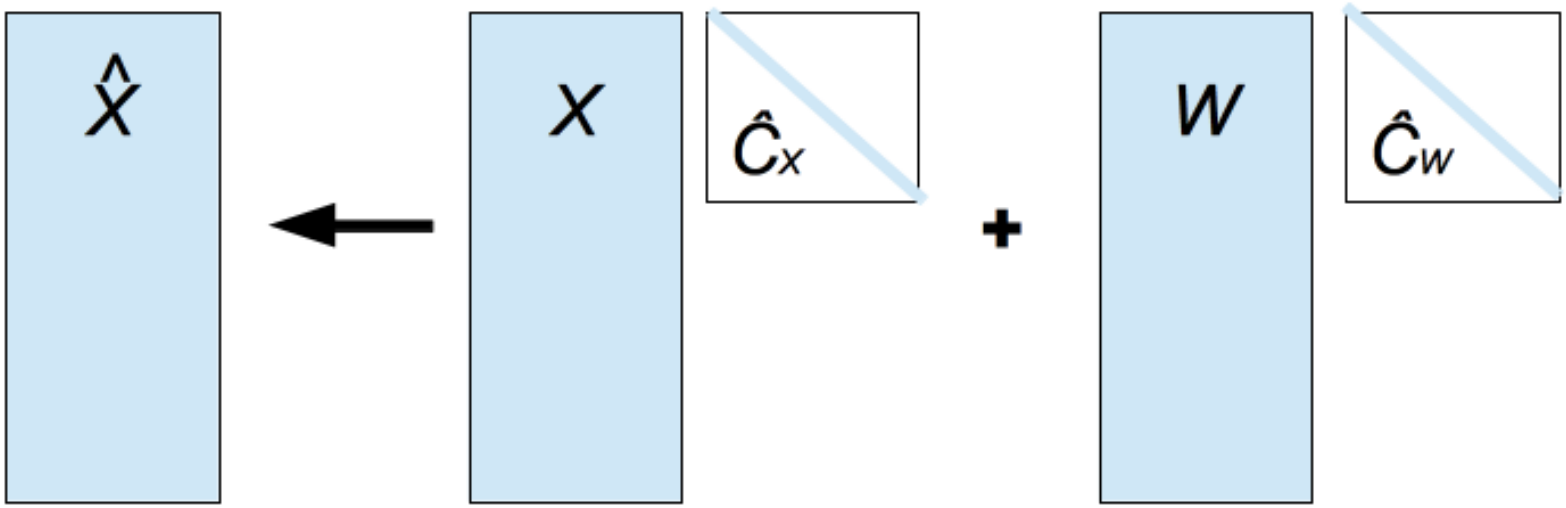
PSD

$$X \leftarrow X C_X + W C_W,$$



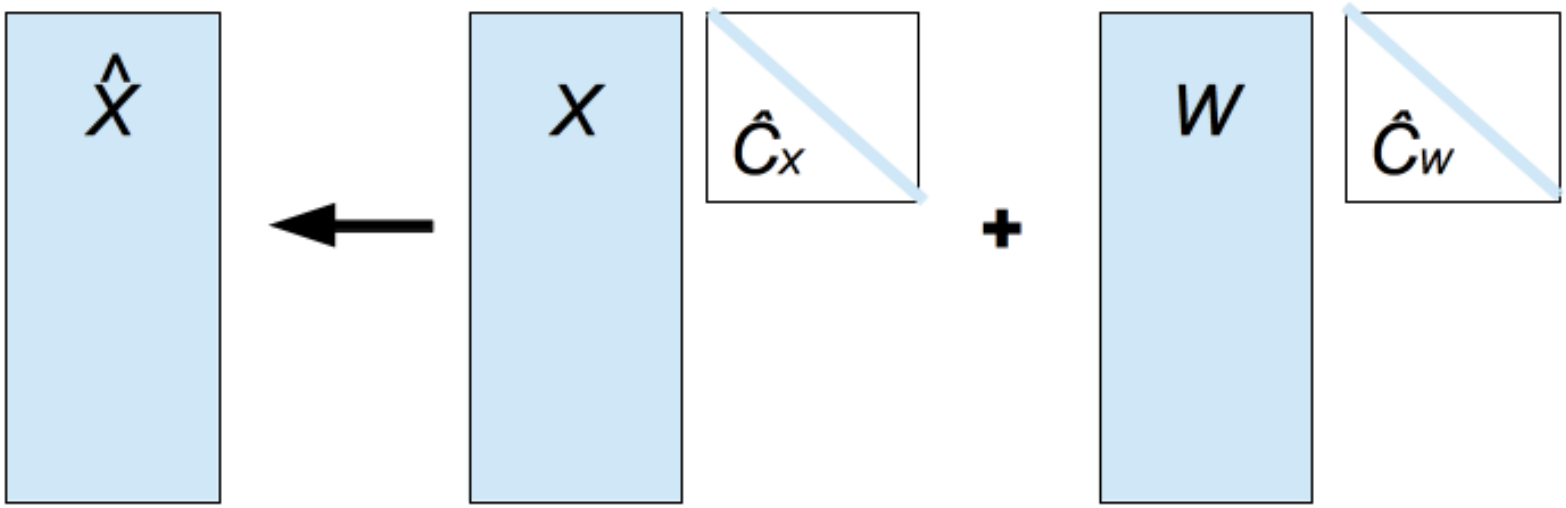
A projected PSD

$$\hat{X} \leftarrow X\hat{C}_X + W\hat{C}_W,$$



A projected PSD

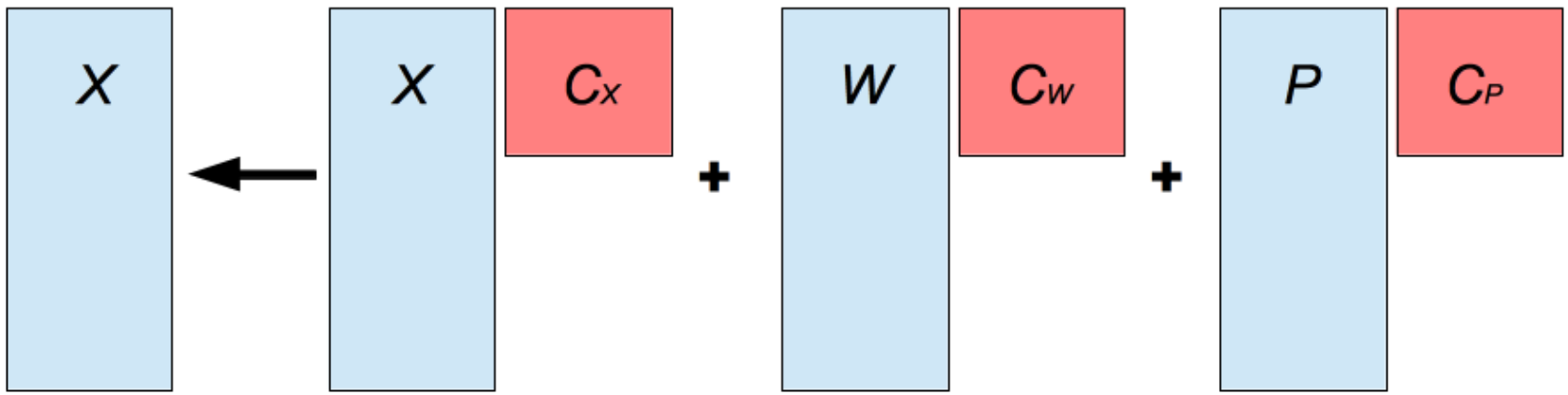
$$\hat{X} \leftarrow X\hat{C}_X + W\hat{C}_W,$$



$$X \leftarrow \text{orth}(\hat{X}), \quad X^*X = I$$

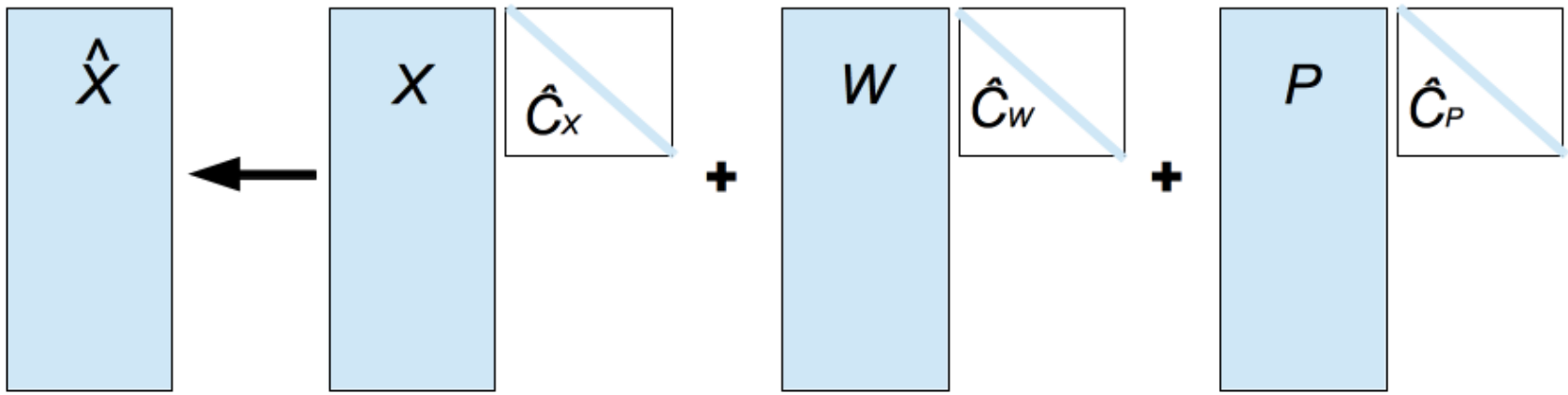
Locally Optimal Block PCG (LOBPCG)

$$X \leftarrow X C_X + W C_W + P C_P, \quad P \in \mathrm{col}\{X, X_{\mathrm{prev}}\}$$



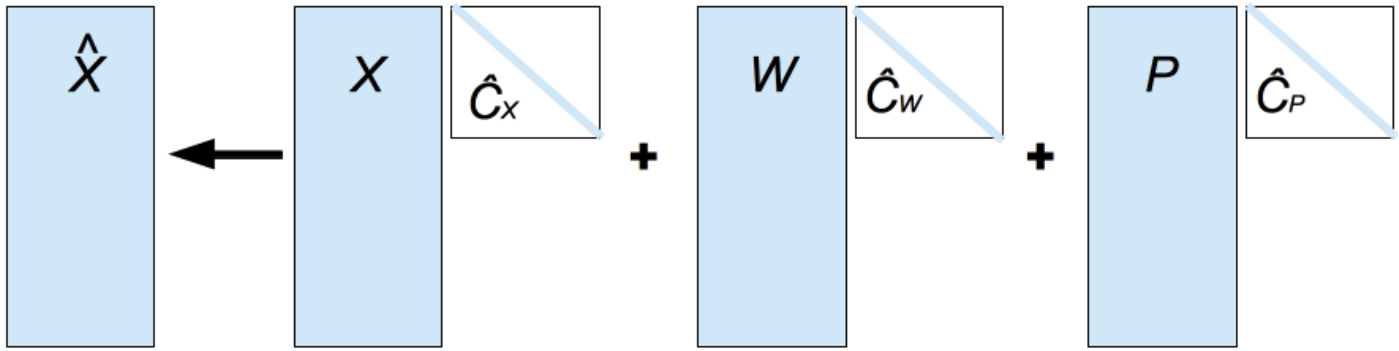
A projected PCG (PPCG)

$$\hat{X} \leftarrow X\hat{C}_X + W\hat{C}_W + P\hat{C}_P, \quad P \in \mathrm{col}\{X, X_{\mathrm{prev}}\}$$



$$X \leftarrow \mathrm{orth}(\hat{X}), \quad X^*X = I$$

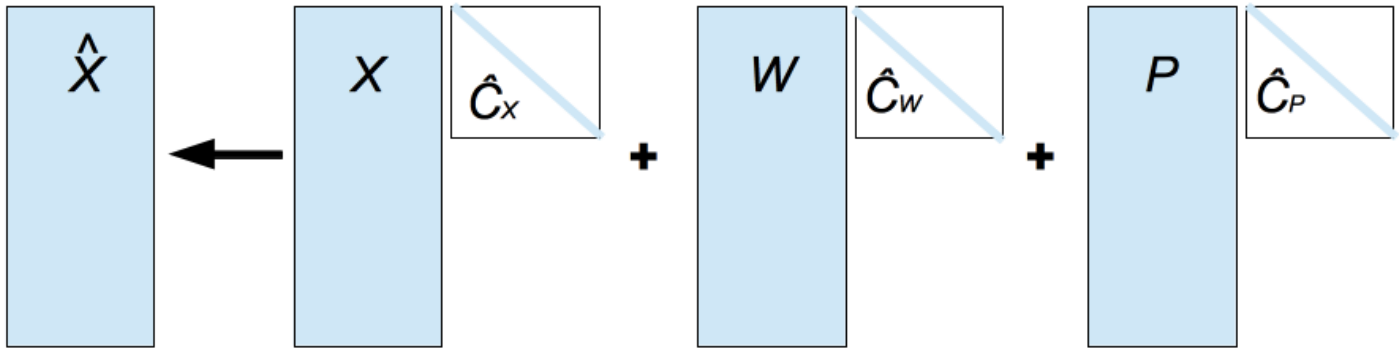
A projected PCG (PPCG)



- Block iteration decouples into single-vector updates

$$\hat{x}_j \leftarrow \alpha_j x_j + \beta_j w_j + \gamma_j p_j, \quad j = 1, \ldots, k;$$

A projected PCG (PPCG)

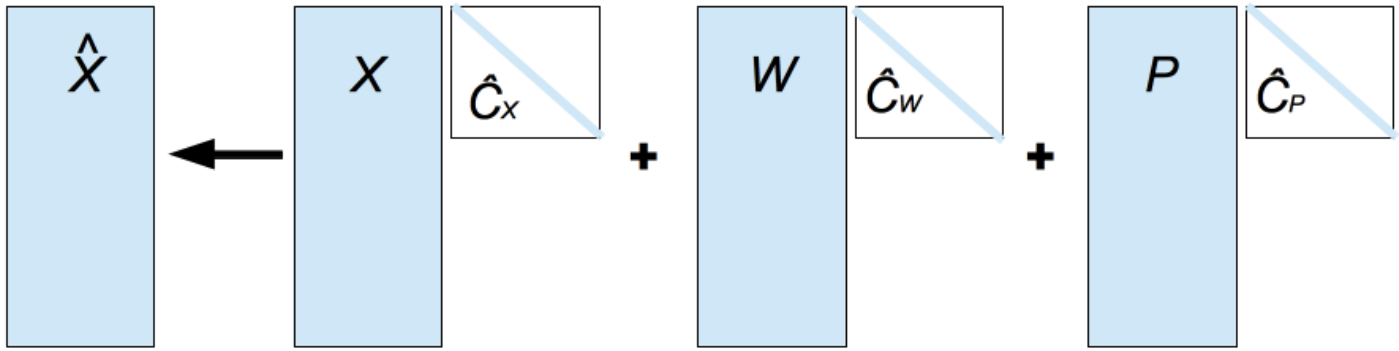


- Block iteration decouples into single-vector updates

$$\hat{x}_j \leftarrow \alpha_j x_j + \beta_j w_j + \gamma_j p_j, \quad j = 1, \ldots, k;$$

- Choose α_j, β_j, and γ_j to minimize $x^* A x$ over span$\{x_j, w_j, p_j\}$
  $\Rightarrow$ solve k 3-by-3 eigenproblems

A projected PCG (PPCG)



- Block iteration decouples into single-vector updates

$$\hat{x}_j \leftarrow \alpha_j x_j + \beta_j w_j + \gamma_j p_j, \quad j = 1, \ldots, k;$$

- Choose α_j, β_j, and γ_j to minimize $x^* A x$ over $\text{span}\{x_j, w_j, p_j\}$
  $\Rightarrow$ solve k 3-by-3 eigenproblems
- $X \leftarrow \text{orth}(\hat{X}), \quad \hat{X} = [\hat{x}_1, \ldots, \hat{x}_k]$

The PPCG algorithm

Given an initial guess $X^{(0)}$ and a preconditioner T, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}, \; P \leftarrow [\,]$
- **While** convergence not reached
  - $W \leftarrow (I - XX^*)T(I - XX^*)(AX - X(X^*AX))$
  - **For** $j = 1, \ldots, k$ **Do**
    - $S \leftarrow [x_j, w_j, p_j]$
    - Find eigenvector $c = (\alpha_j, \beta_j, \gamma_j)^T$ of $S^*ASc = \theta(S^*S)c$ corresponding to the smallest θ
    - $p_j \leftarrow \beta_j w_j + \gamma_j p_j$
    - $\hat{x}_j \leftarrow \alpha_j x_j + p_j$
  - **EndFor**
  - $\hat{X} \leftarrow [\hat{x}_1, \ldots, \hat{x}_k], \; P \leftarrow [p_1, \ldots, p_k]$
  - $X \leftarrow \mathrm{orth}(\hat{X})$
- **EndWhile**

The PPCG algorithm

Given an initial guess $X^{(0)}$ and a preconditioner T, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}, \ P \leftarrow [\,]$
- **While** convergence not reached
  - $W \leftarrow \underbrace{(I - XX^*)T(I - XX^*)}_{M}(AX - X(X^*AX))$
  - **For** $j = 1, \ldots, k$ **Do**
    - $S \leftarrow [x_j, w_j, p_j]$
    - Find eigenvector $c = (\alpha_j, \beta_j, \gamma_j)^T$ of $S^*ASc = \theta(S^*S)c$ corresponding to the smallest θ
    - $p_j \leftarrow \beta_j w_j + \gamma_j p_j$
    - $\hat{x}_j \leftarrow \alpha_j x_j + p_j$
  - **EndFor**
  - $\hat{X} \leftarrow [\hat{x}_1, \ldots, \hat{x}_k], \ P \leftarrow [p_1, \ldots, p_k]$
  - $X \leftarrow \mathrm{orth}(\hat{X})$
- **EndWhile**

The PPCG algorithm

Given an initial guess $X^{(0)}$ and a preconditioner T, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}, \; P \leftarrow [\,]$
- **While** convergence not reached
  - $W \leftarrow \underbrace{(I - XX^*)T(I - XX^*)}_{M}(AX - X(X^*AX))$
  - **For** $j = 1, \ldots, k$ **Do**
    - $S \leftarrow [x_j, w_j, p_j]$
    - Find eigenvector $c = (\alpha_j, \beta_j, \gamma_j)^T$ of $S^*ASc = \theta(S^*S)c$ corresponding to the smallest θ
    - $p_j \leftarrow \beta_j w_j + \gamma_j p_j$
    - $\hat{x}_j \leftarrow \alpha_j x_j + p_j$
  - **EndFor**
  - $\hat{X} \leftarrow [\hat{x}_1, \ldots, \hat{x}_k], \; P \leftarrow [p_1, \ldots, p_k]$
  - $X \leftarrow \mathrm{orth}(\hat{X})$
  - Once in a while, $X \leftarrow \mathrm{RR}(X)$
- **EndWhile**

The PPCG algorithm

Given an initial guess $X^{(0)}$ and a preconditioner T, compute k lowest eigenpairs of A:

- $X \leftarrow X^{(0)}, \; P \leftarrow [\,]$
- **While** convergence not reached
  - $W \leftarrow \underbrace{(I - XX^*)T(I - XX^*)}_{M}(AX - X(X^*AX))$
  - **For** $j = 1, \ldots, k$ **Do** $\Leftarrow$ **can be done for subblocks!**
    - $S \leftarrow [x_j, w_j, p_j]$
    - Find eigenvector $c = (\alpha_j, \beta_j, \gamma_j)^T$ of $S^*ASc = \theta(S^*S)c$ corresponding to the smallest θ
    - $p_j \leftarrow \beta_j w_j + \gamma_j p_j$
    - $\hat{x}_j \leftarrow \alpha_j x_j + p_j$
  - **EndFor**
  - $\hat{X} \leftarrow [\hat{x}_1, \ldots, \hat{x}_k], \; P \leftarrow [p_1, \ldots, p_k]$
  - $X \leftarrow \mathrm{orth}(\hat{X})$
  - Once in a while, $X \leftarrow \mathrm{RR}(X)$
- **EndWhile**

The PPCG algorithm: practical aspects

1. Partition X, W, and P into subblocks, as opposed to columns

The PPCG algorithm: practical aspects

1. Partition X, W, and P into subblocks, as opposed to columns
2. Periodic RR needed (every 5-10 steps) to re-position columns of X into approximate eigenvectors

The PPCG algorithm: practical aspects

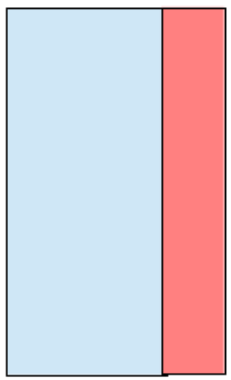
1. Partition X, W, and P into subblocks, as opposed to columns
2. Periodic RR needed (every 5-10 steps) to re-position columns of X into approximate eigenvectors
3. Locking converged eigenpairs

The PPCG algorithm: practical aspects

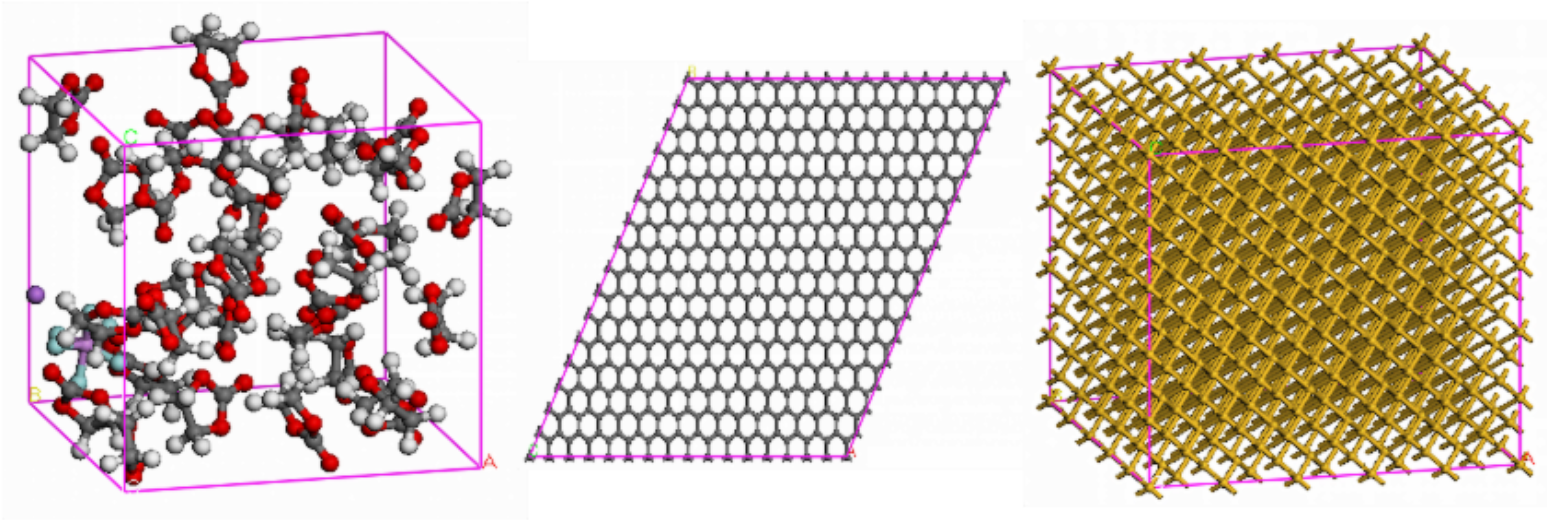
1. Partition X, W, and P into subblocks, as opposed to columns
2. Periodic RR needed (every 5-10 steps) to re-position columns of X into approximate eigenvectors
3. Locking converged eigenpairs
4. QR factorization of $\hat{X}$ can be skipped every few iterations

The PPCG algorithm: practical aspects

1. Partition X, W, and P into subblocks, as opposed to columns
2. Periodic RR needed (every 5-10 steps) to re-position columns of X into approximate eigenvectors
3. Locking converged eigenpairs
4. QR factorization of $\hat{X}$ can be skipped every few iterations
5. "Buffer"



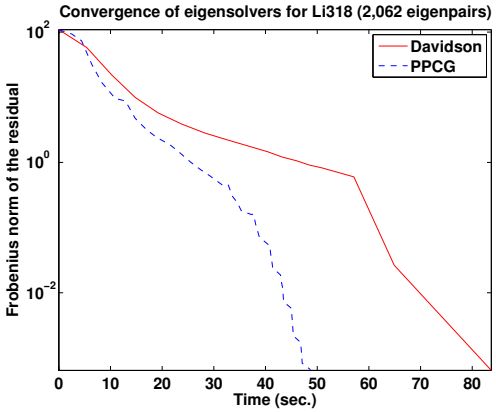
Test problems



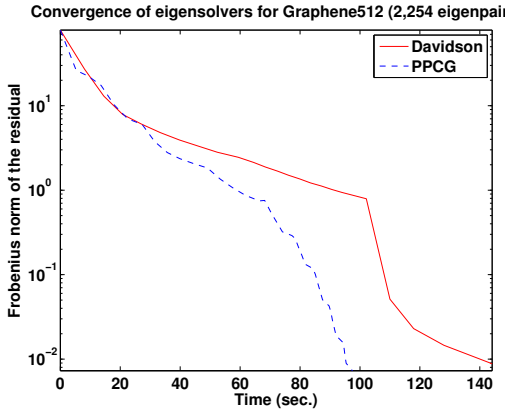
Problem	n_a	ecut (Ryd)	n_G	k
Li318	318	25	206,691	2,062
Graphene	512	25	538,034	2,254
bulk Si	1000	35	1,896,173	2,550

Table: Test problems

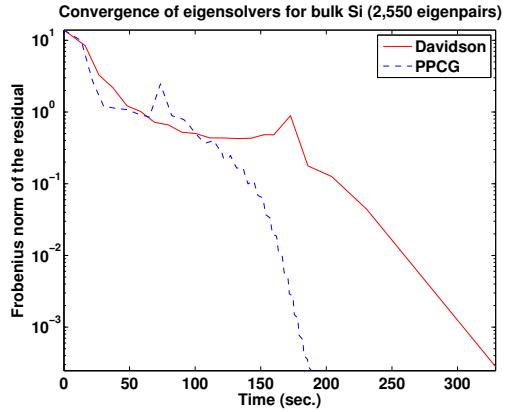
Numerical experiments: execution time



(a) Li318



(b) Graphene512



(c) bulk Si

Figure: Convergence of the Davidson–Liu and PPCG algorithms.

- Implementation in Quantum Espresso
- # of cores: 480 (Li318), 576 (Graphene512), and 2,400 (Si)
- RR performed once every 5 iterations
- Subblock size is 5 (Li318) and 50 (Graphene512 and Si)
- "Buffer size" is 50

Numerical experiments: timing profiles

Computation	PPCG	Davidson
GEMM	16	11
AX	10	6
RR	13	66
CholQR	8	0

Table: The timing profiles (in sec.) for PPCG and Davidson to compute 2,062 eigenpairs of the Li318 problem on 480 cores.

Computation	PPCG	Davidson
GEMM	27	41
AX	94	96
RR	40	191
CholQR	19	0

Table: The timing profiles (in sec.) for PPCG and Davidson to compute 2,550 eigenpairs of the bulk Si problem on 2,400 cores.

Numerical experiments: scaling (bulk Si)

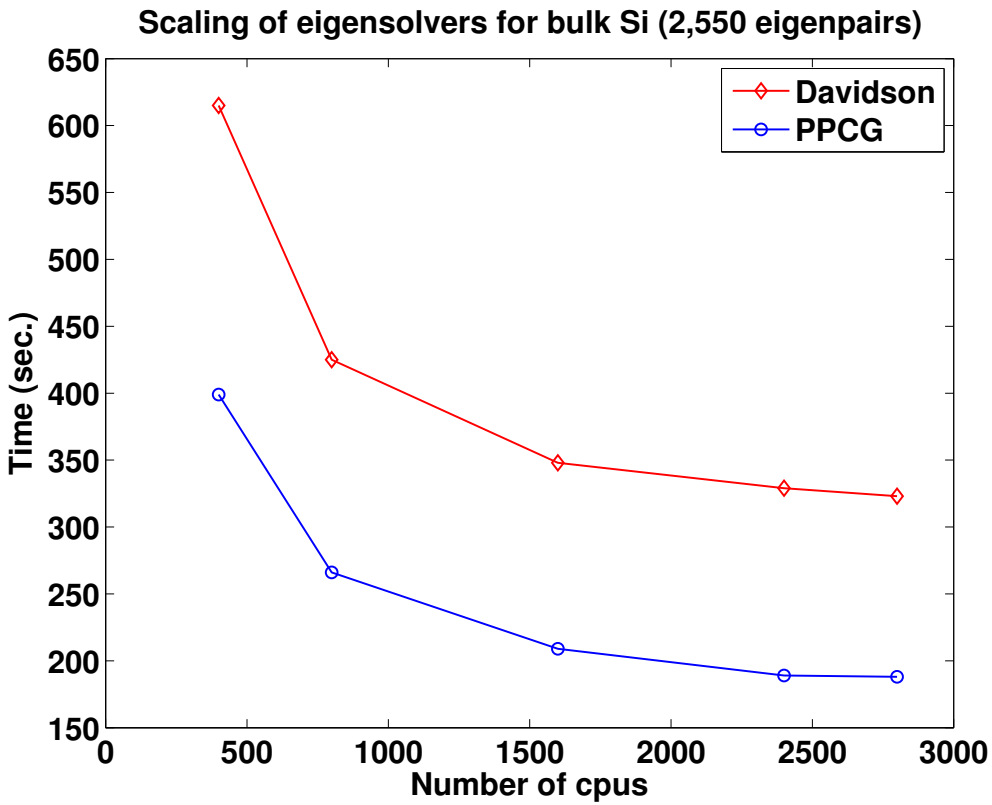
	ncpus					
Computation	200	400	800	1,600	2,400	2,800
GEMM	202	104	57	35	27	26
AX	247	165	129	106	94	92
RR	142	77	48	40	40	41
CholQR	66	91	21	18	19	21
Total	685	399	266	209	189	188

Table: Scaling of different computational components of PPCG.

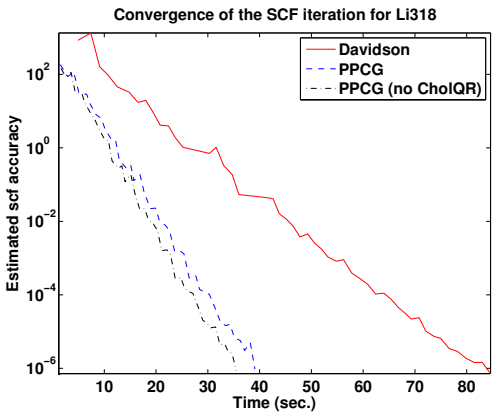
	ncpus					
Computation	200	400	800	1,600	2,400	2,800
GEMM	248	138	76	47	41	38
AX	253	169	133	111	96	96
RR	474	303	214	189	191	189
Total	986	615	425	348	329	323

Table: Scaling of different computational components of Davidson.

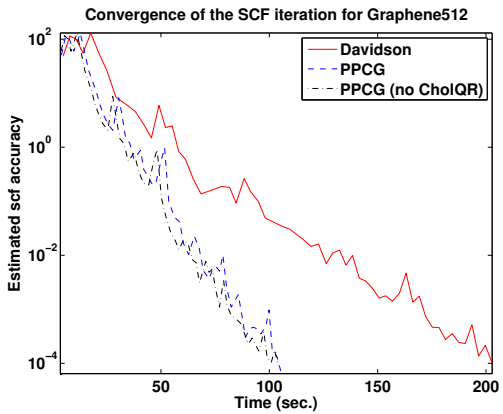
Numerical experiments: scaling (bulk Si)



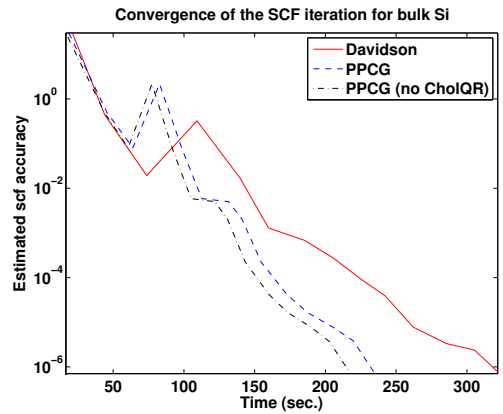
Numerical experiments: the SCF loop



(a) Li318



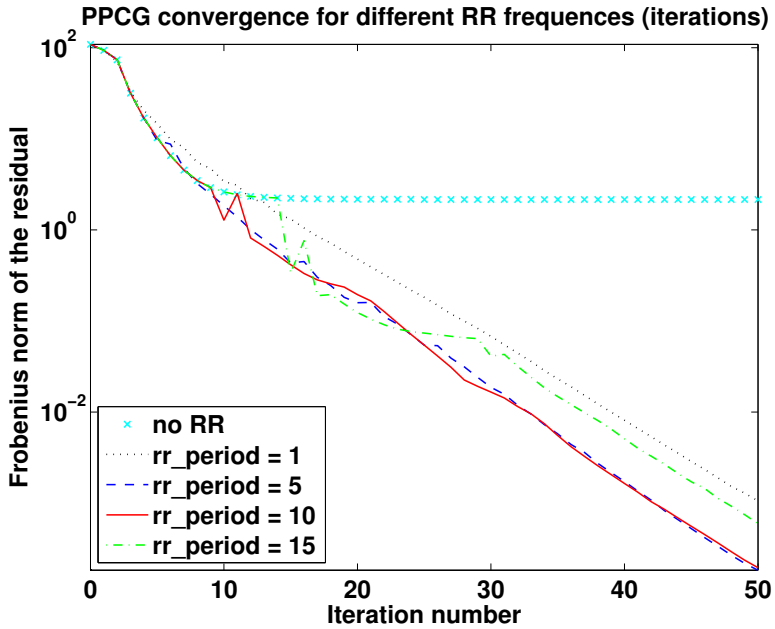
(b) Graphene512



(c) bulk Si

Figure: The convergence of the SCF iteration with Davidson and PPCG.

Numerical experiments: effects of RR (Li318)



Numerical experiments: effects of the subblock size (Li318)

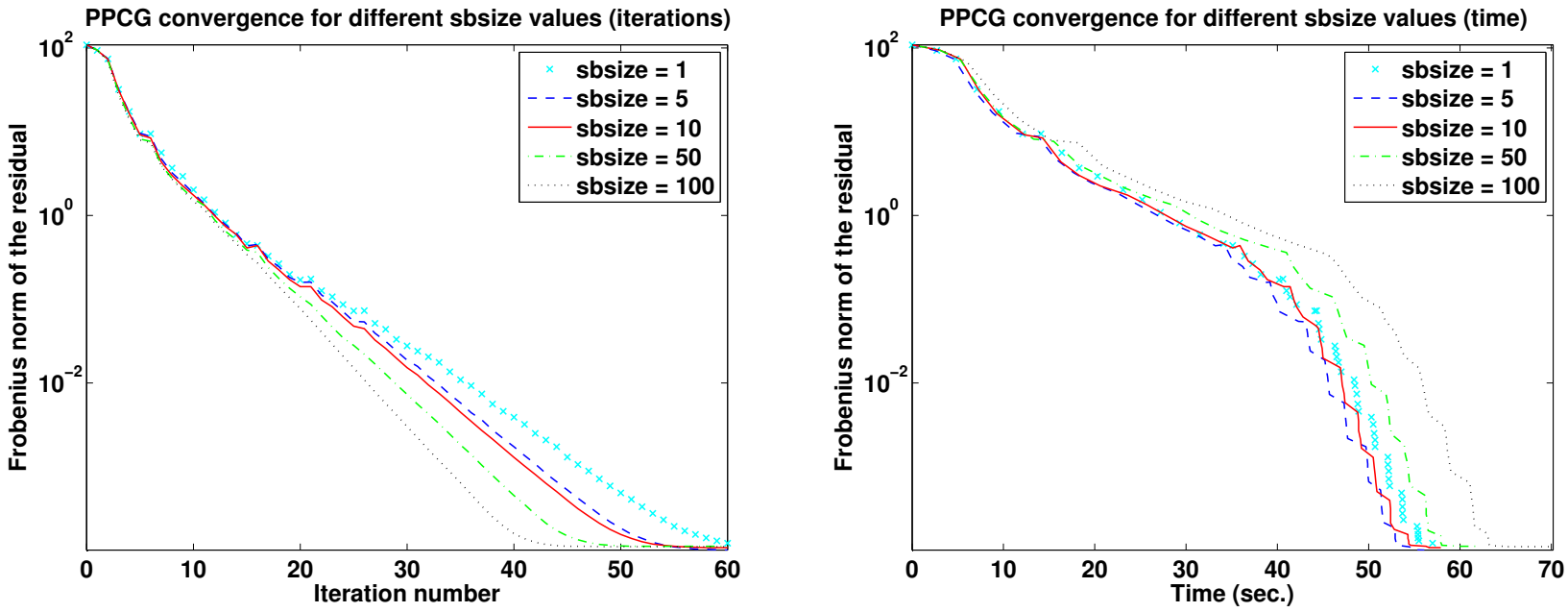


Figure: Convergence rate with respect to **iteration count** (left) and **time** (right)

Conclusions

- The PPCG iteration reduces the amount of RR computations
- Up to 2x speedup demonstrated
- Pilot versions implemented in Quantum Espresso and Qbox

Current and future work

- Better theoretical understanding (domain decomposition framework?)
- Integration into molecular dynamics simulation codes, with application to the chemistry and dynamics of lithium-ion batteries
- Extension to non-linear, structured, and interior eigenproblems

Questions?

Thank you!

- E. Vecharynski, C. Yang, and J. E. Pask: *"A Projected Preconditioned Conjugate Gradient Algorithm for Computing a Large Invariant Subspace of a Hermitian Matrix"*, available at http://arxiv.org/abs/1407.7506

QE process grid for a dense ScaLAPACK eigensolver

ncpus	Processor grid
200	10x10
400	14x14
800	20x20
1,600	28x28
2,400	34x34
3,000	38x38

Table: The default ScaLAPACK processor grid configurations used by QE for different total core counts.

Numerical experiments: effects of RR (Li318)

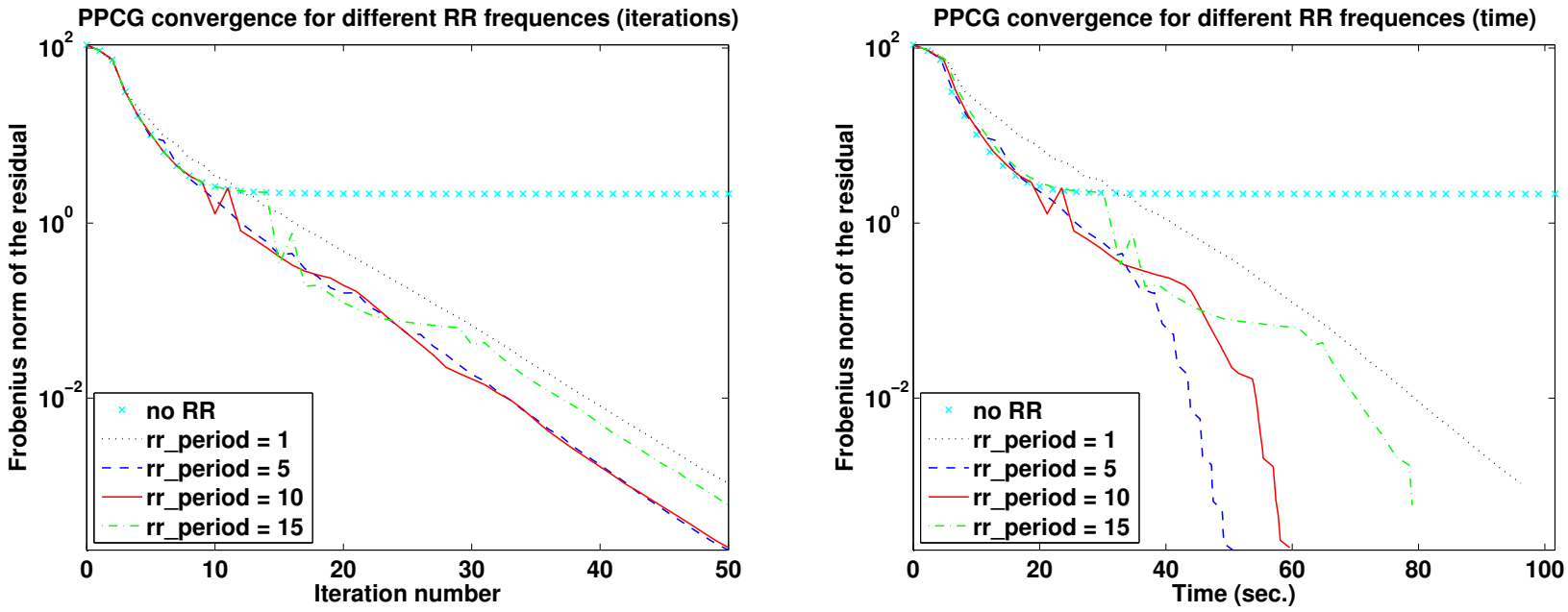


Figure: Convergence rate with respect to **iteration count** (left) and **time** (right)