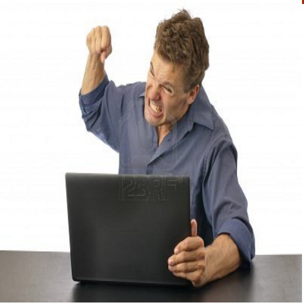
Towards resilient Krylov solvers

1st CNPq/INRIA meeting
Sophia Antipolis, France.

Emmanuel AGULLO, Luc GIRAUD
Abdou GUERMOUCHE, Jean ROMAN
Mawussi ZOUNON

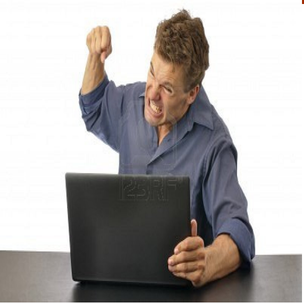
PROJECT-TEAM
HiePACS
Joint lab INRIA-CERFACS
FRANCE

Agence Nationale de la Recherche



- Iterative methods in parallel distributed environment.
- If one Processor fails, all its data are lost.
- Impossible to continue iterations.

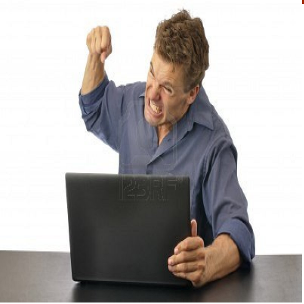
Introduction



- Iterative methods in parallel distributed environment.
- If one Processor fails, all its data are lost.
- Impossible to continue iterations.

Resilience: Ability to compute a correct output in presence of faults.

Introduction



- ⋆ Iterative methods in parallel distributed environment.
- ⋆ If one Processor fails, all its data are lost.
- ⋆ Impossible to continue iterations.

Resilience: Ability to compute a correct output in presence of faults.

- ⋆ Goal: Keep converging in presence of fault.
- ⋆ Method: Re-generate lost data <u>without</u> Checkpoint/Restart strategy.
- ⋆ Approach: Numerical algorithm.
- ⋆ Context: Krylov solvers.

Outline

1. Faults in HPC Systems
2. Iterative methods for sparse linear systems
3. Our model assumptions
4. Interpolation methods
5. Concluding remarks and perspectives

Outline

Framework

Forecast for exascale systems

- Mean Time Between Failure (MTBF): less the one hour.
- Checkpoint overhead:
  - 30 minutes per checkpoint.
  - 1 Terabyte/second.

- Limitation of classical checkpointing.
- Explore fault-tolerant schemes with less/no overhead.
- Numerical algorithms to deal with overhead issue.

Faults in this presentation

- Invalid processor (memory, caches, network connections, . . .)

Framework

Forecast for exascale systems

- Mean Time Between Failure (MTBF): less the one hour.
- Checkpoint overhead:
  - 30 minutes per checkpoint.
  - 1 Terabyte/second.

- Limitation of classical checkpointing.
- Explore fault-tolerant schemes with less/no overhead.
- Numerical algorithms to deal with overhead issue.

Faults in this presentation

- Invalid processor (memory, caches, network connections, ...)

Framework

Forecast for exascale systems

- Mean Time Between Failure (MTBF): less the one hour.
- Checkpoint overhead:
  - 30 minutes per checkpoint.
  - 1 Terabyte/second.

- Limitation of classical checkpointing.
- Explore fault-tolerant schemes with less/no overhead.
- Numerical algorithms to deal with overhead issue.

Faults in this presentation

- Invalid processor (memory, caches, network connections, . . .)

Outline

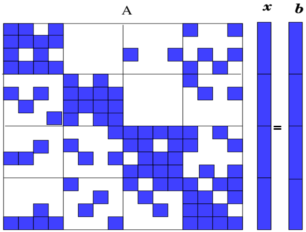
1. Faults in HPC Systems

2. Iterative methods for sparse linear systems

3. Our model assumptions

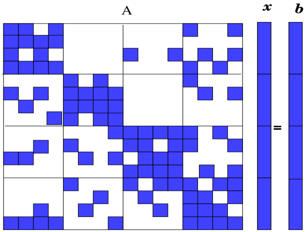
4. Interpolation methods

5. Concluding remarks and perspectives



$$Ax = b.$$

We have to design fault tolerant solver for sparse linear system.

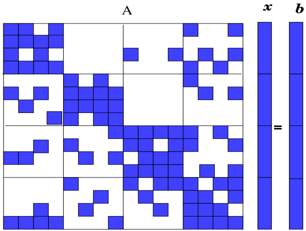


$$Ax = b.$$

We have to design fault tolerant solver for sparse linear system.

Two classes of iterative methods

- Stationary methods (Jacobi, Gauss-Seidel, ...).
- Krylov subspace methods (GMRES, CG, Bi-CGStab, ...).



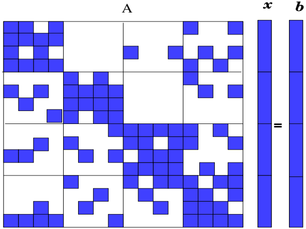
$$Ax = b.$$

We have to design fault tolerant solver for sparse linear system.

Two classes of iterative methods

- ⋆ Stationary methods (Jacobi, Gauss-Seidel, ...).
- ⋆ Krylov subspace methods (GMRES, CG, Bi-CGStab, ...).

⋆ Krylov methods have attractive potential for resilience.



$$Ax = b.$$

We have to design fault tolerant solver for sparse linear system.

Two classes of iterative methods

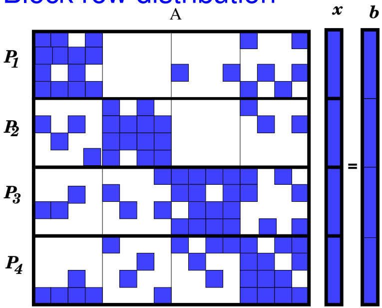
- Stationary methods (Jacobi, Gauss-Seidel, ...).
- Krylov subspace methods (GMRES, CG, Bi-CGStab, ...).

- Krylov methods have attractive potential for resilience.
- They combine two main advantages:
  - Numerical robustness.
  - Converging in presence of fault when clever recovering schemes are employed.

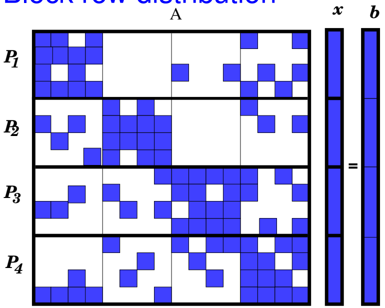
Outline

1. Faults in HPC Systems
2. Iterative methods for sparse linear systems
3. Our model assumptions
4. Interpolation methods
5. Concluding remarks and perspectives

Block row distribution



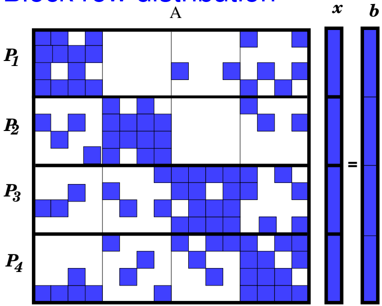
Block row distribution



Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

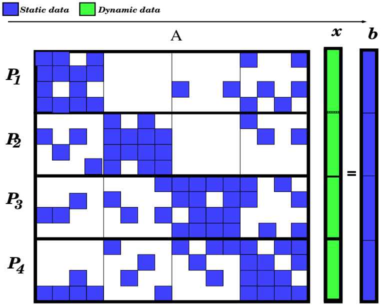
- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.

Block row distribution



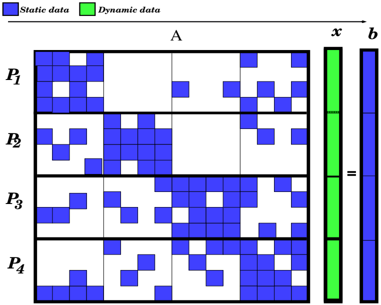
Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.



Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

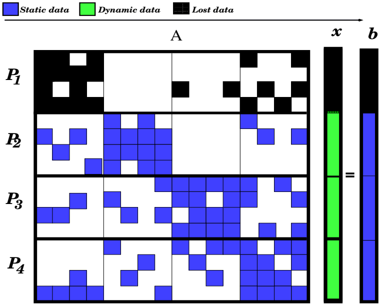
- Computational environment.
- Static data.
- Dynamic data.



Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.

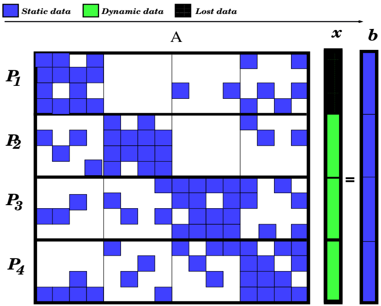
Let's Assume that P_1 fails.



Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.

Let's Assume that P_1 fails.

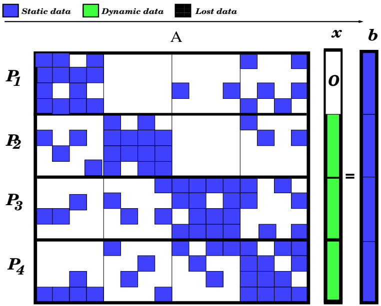


Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.

Let's Assume that P_1 fails.

- ★ Failed processor is replaced.
- ★ Static data are recovered.

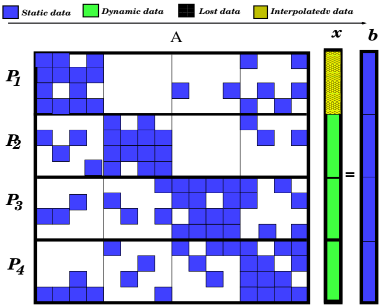


Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

- ★ Computational environment.
- ★ Static data.
- ★ Dynamic data.

Let's Assume that P_1 fails.

- ★ Failed processor is replaced.
- ★ Static data are recovered.
- ★ Reset: Set (x_1) to zero.



Three categories of lost data [Julien Langou et al, SIAM J. Sci, 2007]

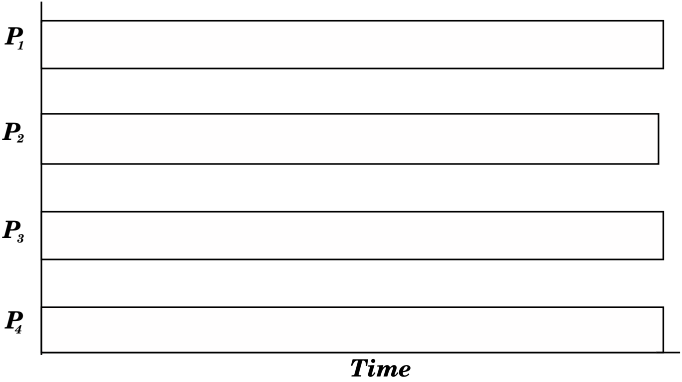
- Computational environment.
- Static data.
- Dynamic data.

Let's Assume that P_1 fails.

- Failed processor is replaced.
- Static data are recovered.
- Reset: Set (x_1) to zero.

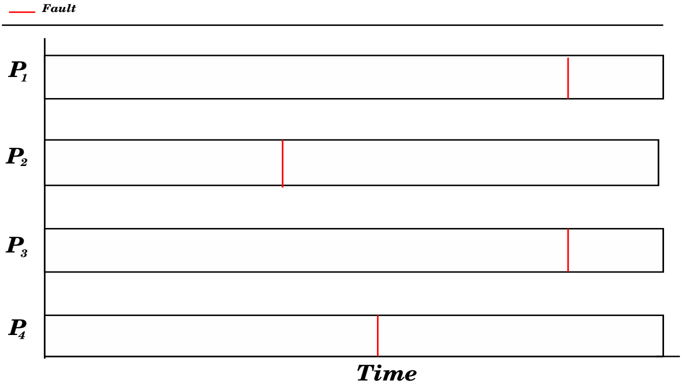
Our algorithms aim at recovering x_1.

Overview of our fault tolerant algorithm



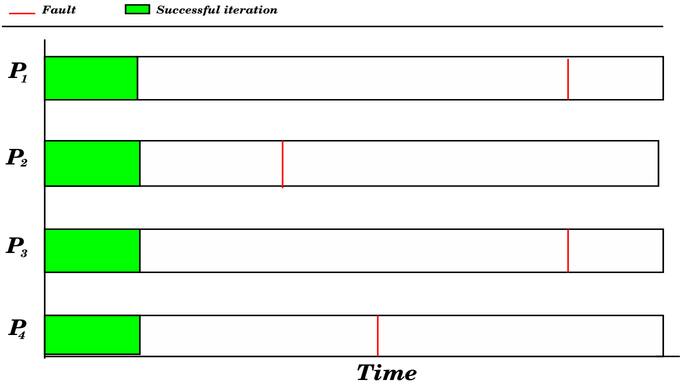
- Matlab prototype.
- Simulation of parallel environment.

Overview of our fault tolerant algorithm



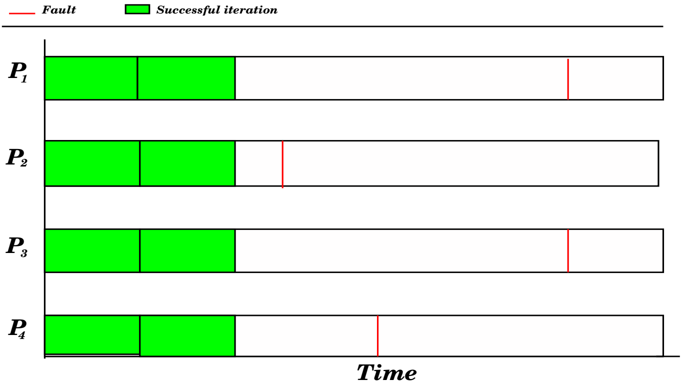
- Matlab prototype.
- Simulation of parallel environment.
- Generation of fault trace.
- Realistic probability distribution.

Overview of our fault tolerant algorithm



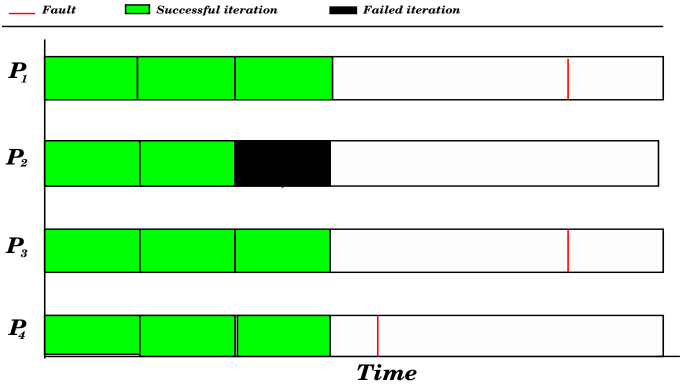
- Matlab prototype.
- Simulation of parallel environment.
- Generation of fault trace.
- Realistic probability distribution.

Overview of our fault tolerant algorithm



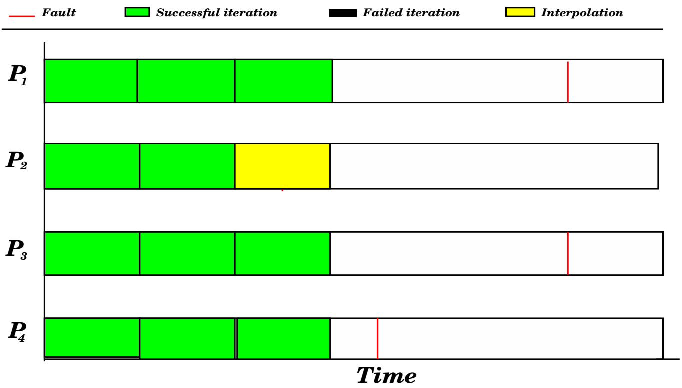
- ★ Matlab prototype.
- ★ Simulation of parallel environment.
- ★ Generation of fault trace.
- ★ Realistic probability distribution.

Overview of our fault tolerant algorithm



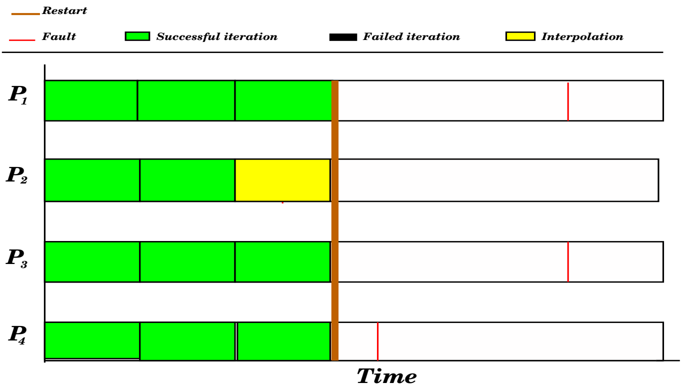
- ⋆ Matlab prototype.
- ⋆ Simulation of parallel environment.
- ⋆ Generation of fault trace.
- ⋆ Realistic probability distribution.

Overview of our fault tolerant algorithm



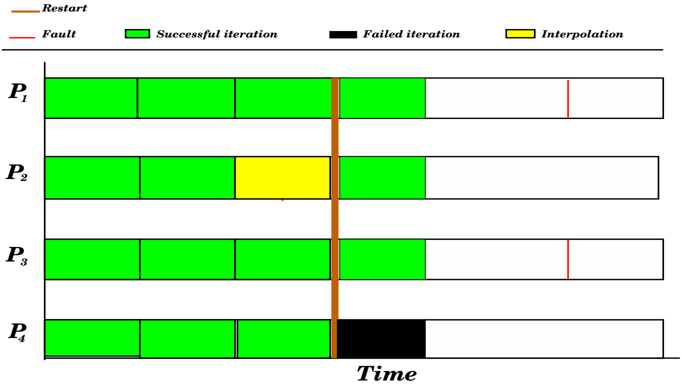
- ★ Matlab prototype.
- ★ Simulation of parallel environment.
- ★ Generation of fault trace.
- ★ Realistic probability distribution.

Overview of our fault tolerant algorithm



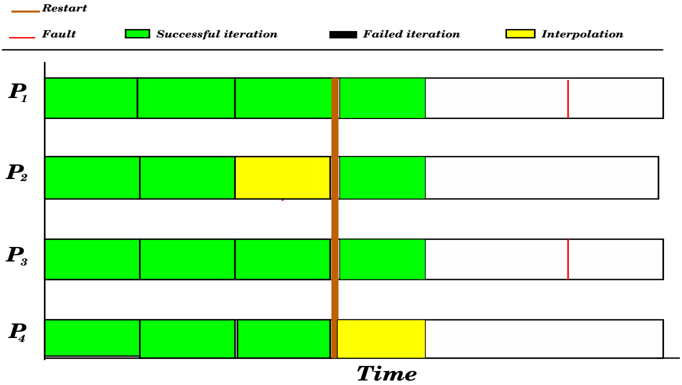
- ★ Matlab prototype.
- ★ Simulation of parallel environment.
- ★ Generation of fault trace.
- ★ Realistic probability distribution.

Overview of our fault tolerant algorithm



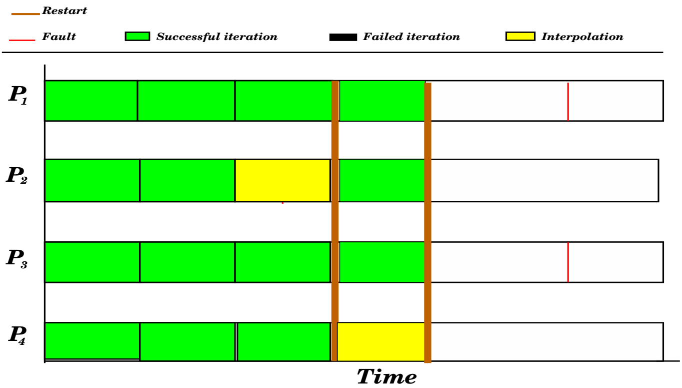
- Matlab prototype.
- Simulation of parallel environment.
- Generation of fault trace.
- Realistic probability distribution.

Overview of our fault tolerant algorithm



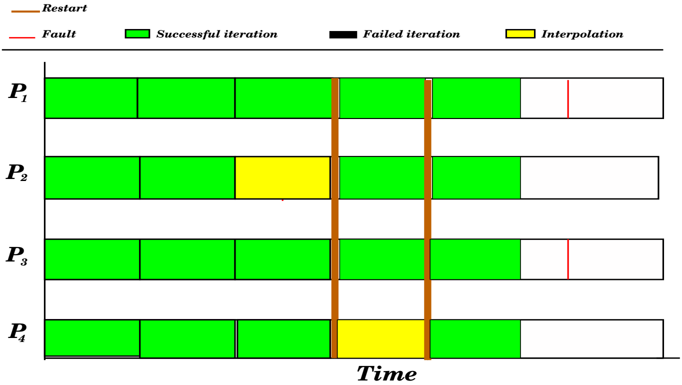
- Matlab prototype.
- Simulation of parallel environment.
- Generation of fault trace.
- Realistic probability distribution.

Overview of our fault tolerant algorithm



- ⋆ Matlab prototype.
- ⋆ Simulation of parallel environment.
- ⋆ Generation of fault trace.
- ⋆ Realistic probability distribution.

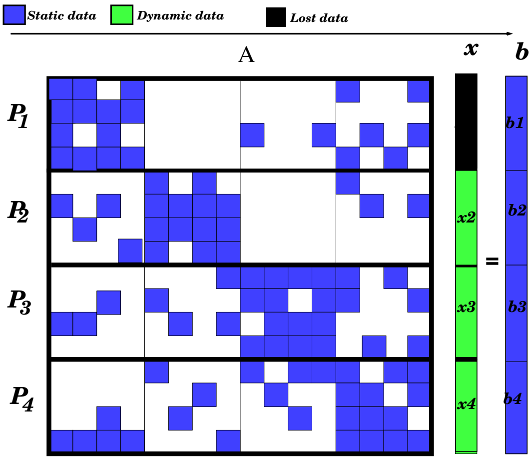
Overview of our fault tolerant algorithm



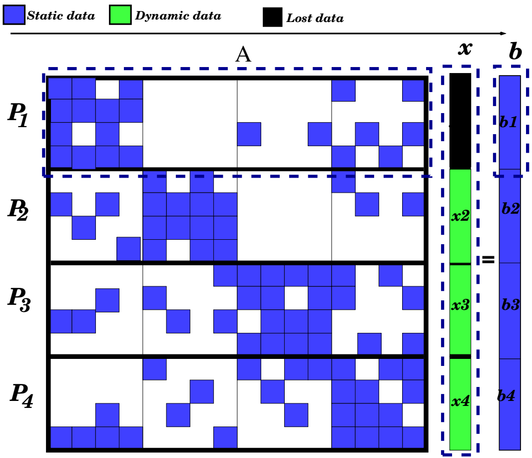
- Matlab prototype.
- Simulation of parallel environment.
- Generation of fault trace.
- Realistic probability distribution.

Outline

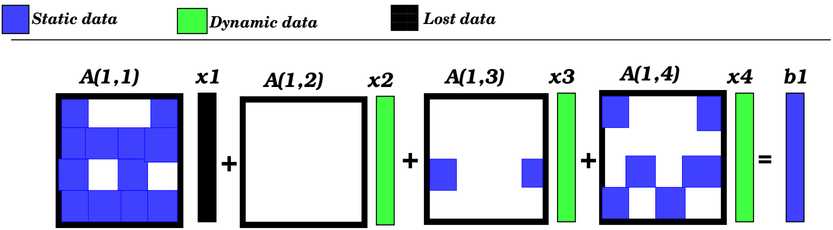
Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]

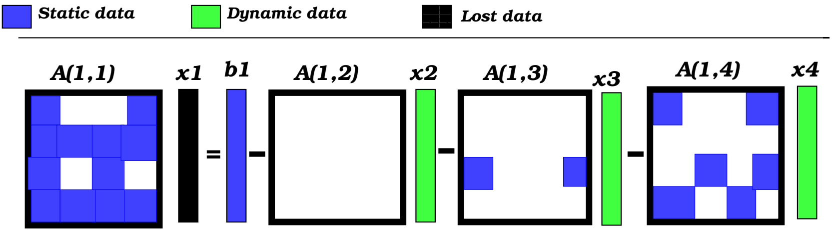


Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



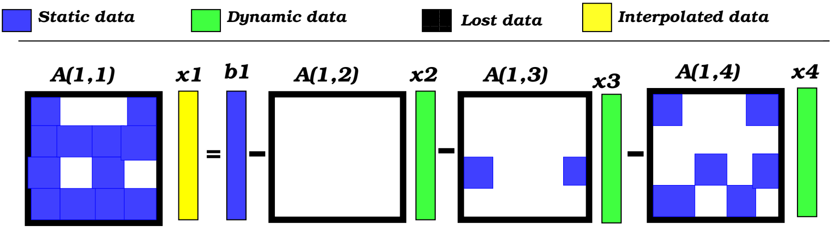
$$A_{(1,1)}x_1 + A_{(1,2)}x_2 + A_{(1,3)}x_3 + A_{(1,4)}x_4 = b_1.$$

Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



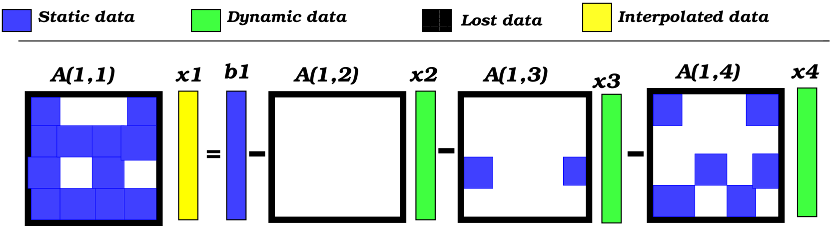
$$A_{(1,1)}x_1 = b_1 - A_{(1,2)}x_2 - A_{(1,3)}x_3 - A_{(1,4)}x_4.$$

Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



$$A_{(1,1)}x_1 = b_1 - A_{(1,2)}x_2 - A_{(1,3)}x_3 - A_{(1,4)}x_4.$$

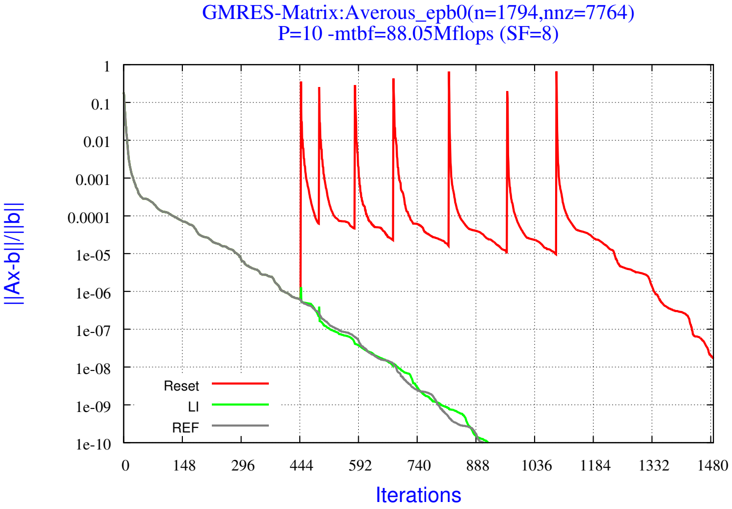
Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



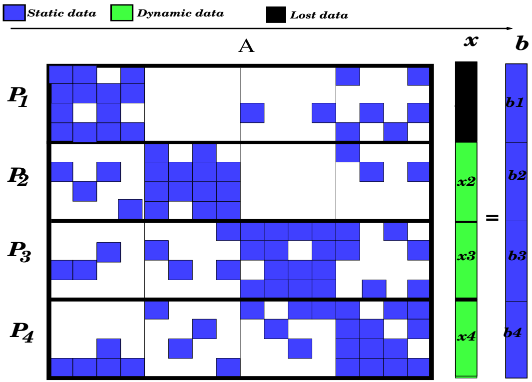
$$A_{(1,1)} x_1 = b_1 - A_{(1,2)} x_2 - A_{(1,3)} x_3 - A_{(1,4)} x_4.$$

$$A_{(i,i)} x_i^{(new)} = b_i - \sum_{i \neq j} A_{(i,j)} x_j.$$

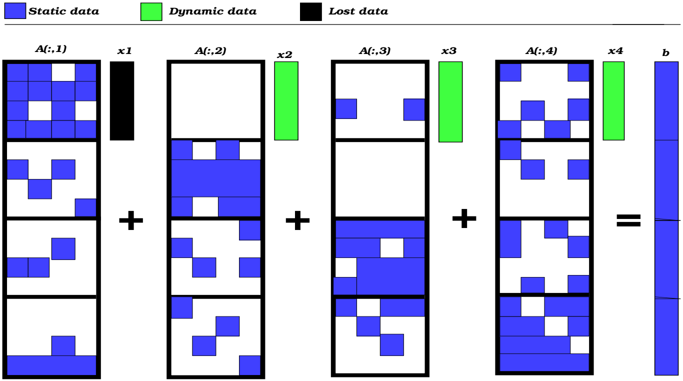
Linear Interpolation (LI) [J. Langou et al, SIAM J. Sci, 2007]



Least squares interpolation (LSI)

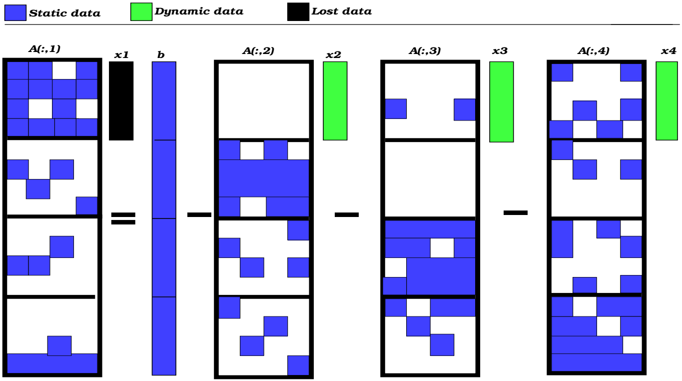


Least squares interpolation (LSI)



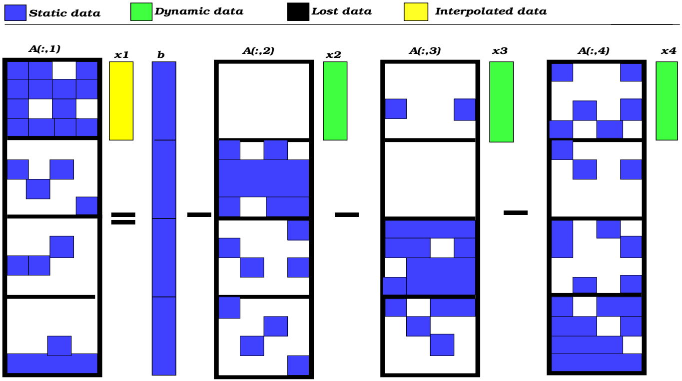
$$A_{(:,1)}x_1 + A_{(:,2)}x_2 + A_{(:,3)}x_3 + A_{(:,4)}x_4 = b.$$

Least squares interpolation (LSI)



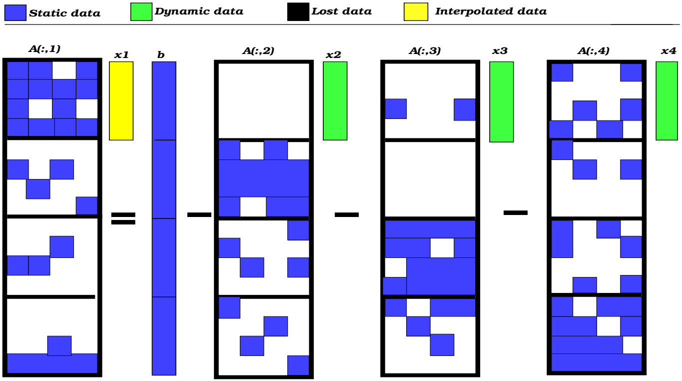
$$x_1 = \underset{x}{argmin} \|(b - A_{(:,2)}x_2 - A_{(:,3)}x_3 - A_{(:,4)}x_4) - A_{(:,1)}x\|_2.$$

Least squares interpolation (LSI)



$$x_1 = \underset{x}{argmin} \| (b - A_{(:,2)}x_2 - A_{(:,3)}x_3 - A_{(:,4)}x_4) - A_{(:,1)}x \|_2.$$

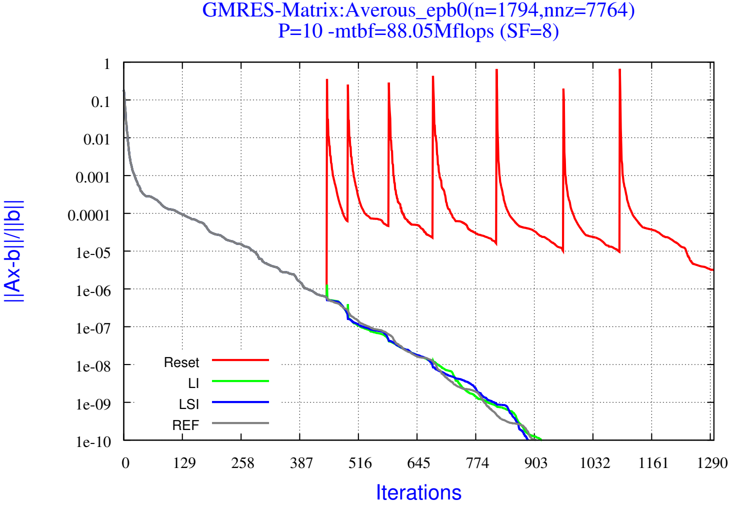
Least squares interpolation (LSI)



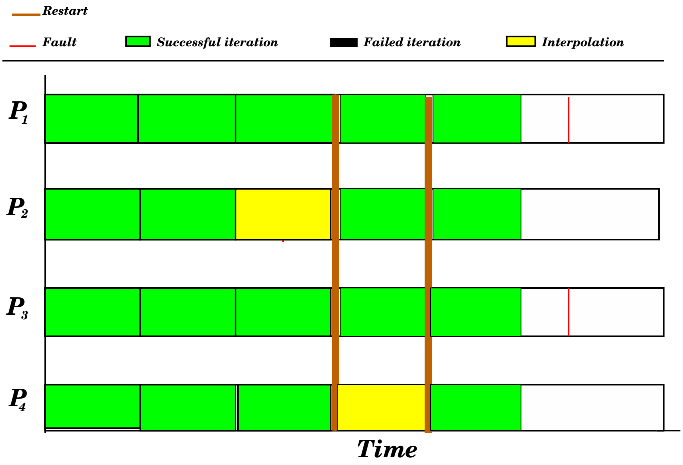
$$x_1 = \underset{x}{argmin} \left\| \left(b - A_{(:,2)} x_2 - A_{(:,3)} x_3 - A_{(:,4)} x_4 \right) - A_{(:,1)} x \right\|_2 .$$

LSI preserves the residual norm decrease monotony of GMRES.

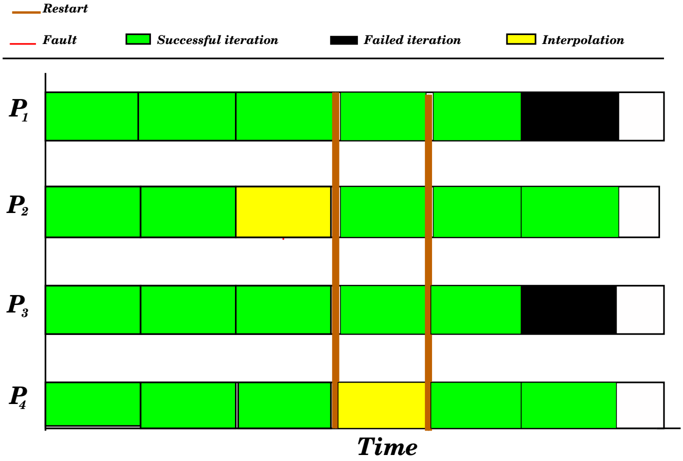
Least squares interpolation (LSI)



Multiple Faults

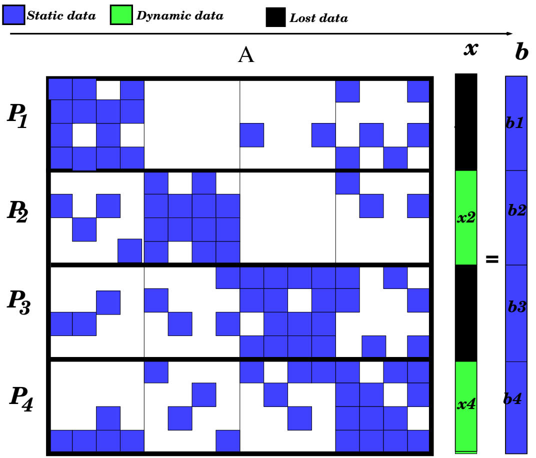


Multiple Faults



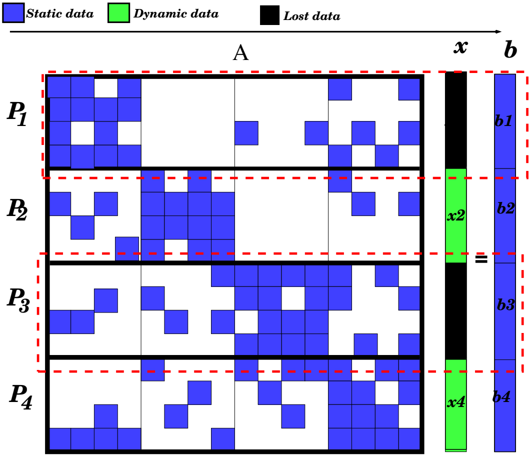
Multiple faults: more than one fault at the same iteration.

Multiple Faults

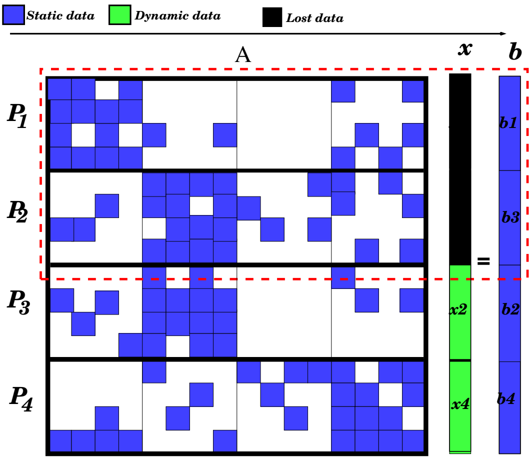


- x_3 is needed to interpolate x_1, vice-versa.
- How to deal with data dependency?

Assembled recovery: LI-A/LSI-A

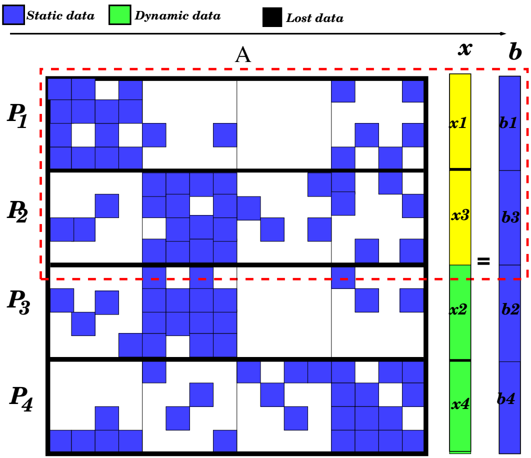


Assembled recovery: LI-A/LSI-A



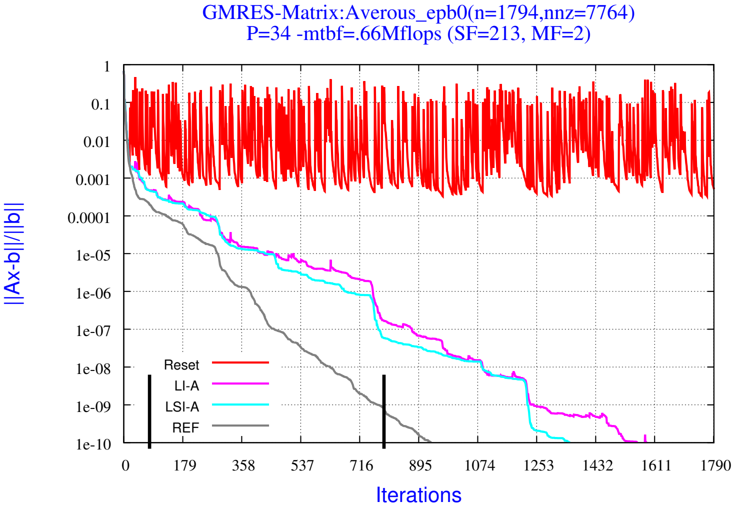
Failed blocks are assembled.

Assembled recovery: LI-A/LSI-A



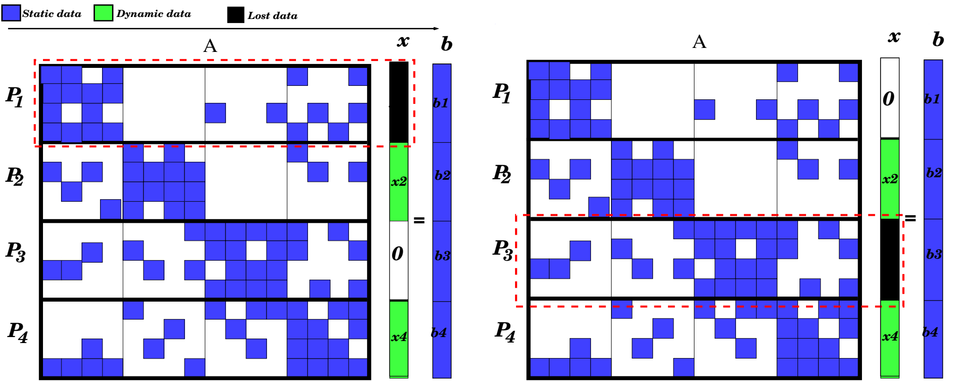
Failed blocks are assembled.

Assembled recovery: LI-A/LSI-A



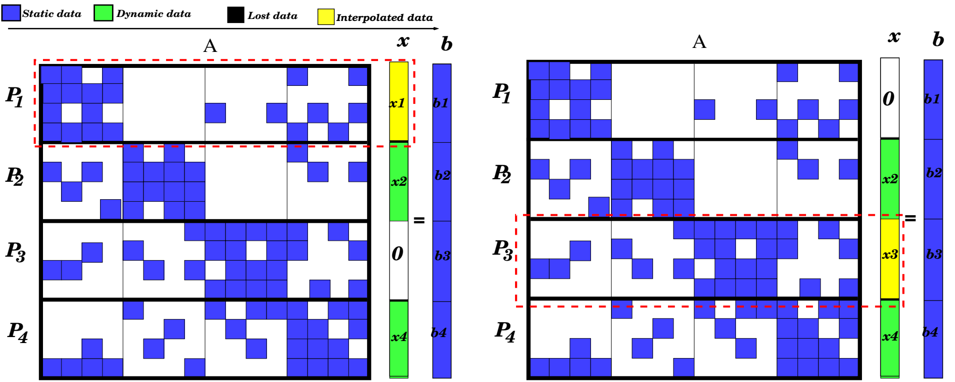
- 2 multiple faults.
- 56^{th} iteration and 784^{th} iteration.

Parallel recovery: LI-P/LSI-P



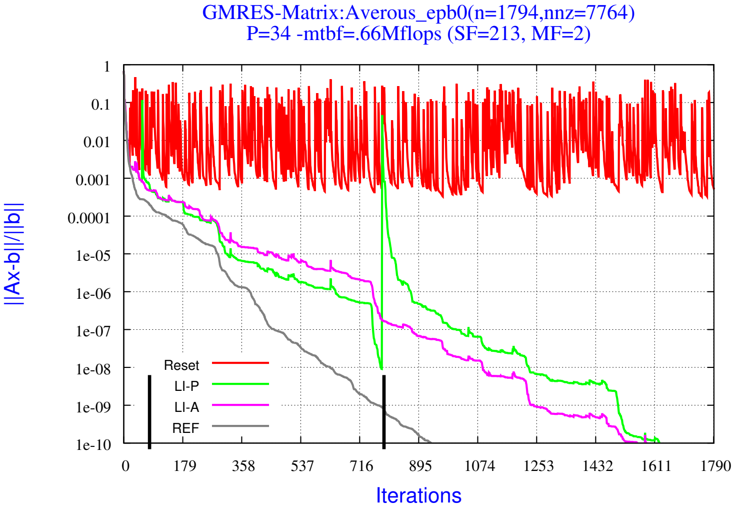
Interpolate x_3 assuming that x_1 is equal to zero subvector.
Interpolate x_1 assuming that x_3 is equal to zero subvector.

Parallel recovery: LI-P/LSI-P



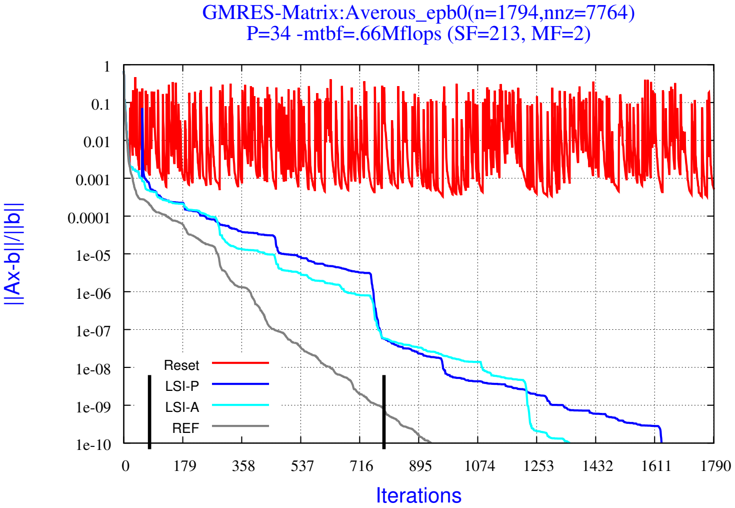
Interpolate x_3 assuming that x_1 is equal to zero subvector.
Interpolate x_1 assuming that x_3 is equal to zero subvector.

LI-P



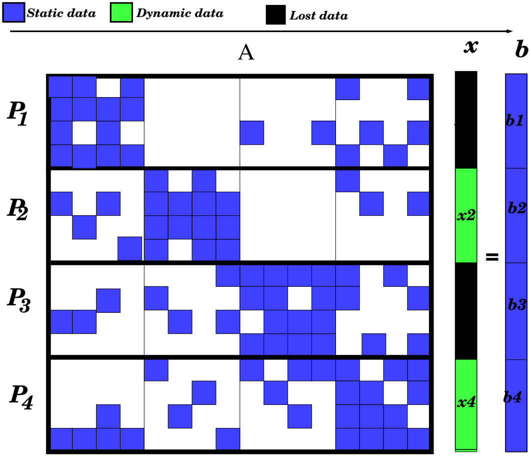
- 2 multiple faults.
- 56^{th} iteration and 784^{th} iteration.

LSI-P

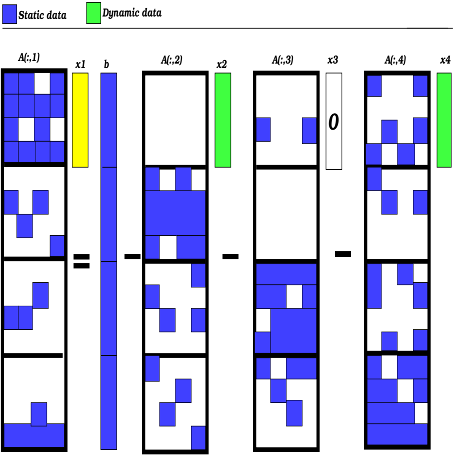


- 2 multiple faults.
- 56^{th} iteration and 784^{th} iteration.

Impact of data dependency in LSI-P

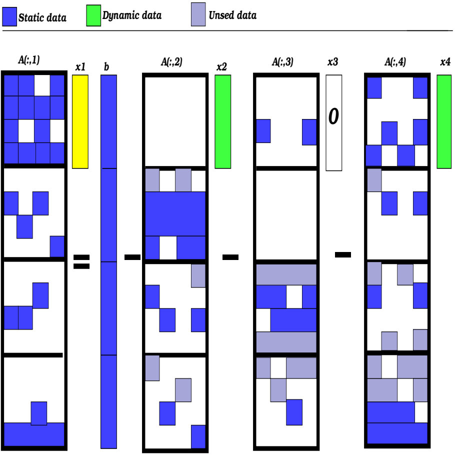


Impact of data dependency in LSI-P



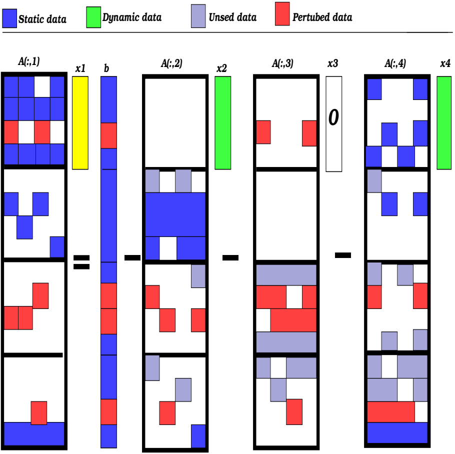
- ⋆ Taking $x_3 = 0$, induces perturbation.

Impact of data dependency in LSI-P



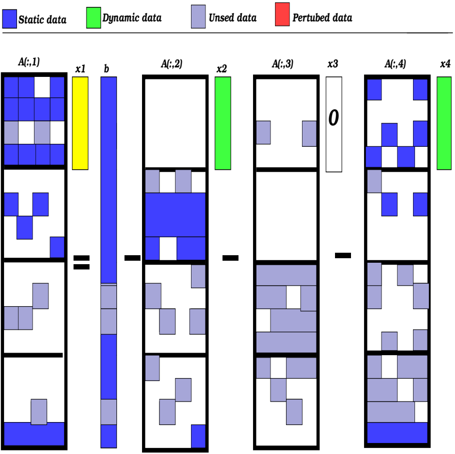
- ⋆ Taking $x_3 = 0$, induces perturbation.
- ⋆ Sparse overdetermined least squares.

Impact of data dependency in LSI-P



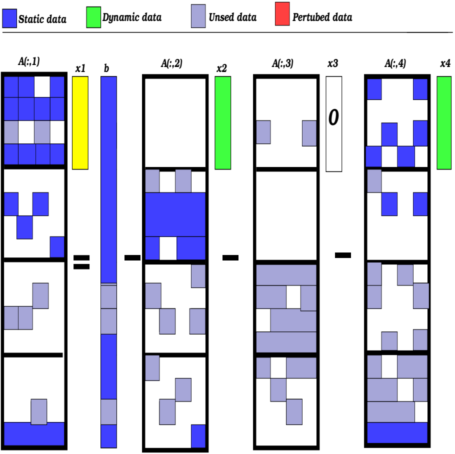
- ★ Taking $x_3 = 0$, induces perturbation.
- ★ Sparse overdetermined least squares.
- ★ Identification of perturbed rows.

Impact of data dependency in LSI-P



- ⋆ Taking $x_3 = 0$, induces perturbation.
- ⋆ Sparse overdetermined least squares.
- ⋆ Identification of perturbed rows.
- ⋆ Discard of perturbed rows.

Impact of data dependency in LSI-P



- ⋆ Taking $x_3 = 0$, induces perturbation.
- ⋆ Sparse overdetermined least squares.
- ⋆ Identification of perturbed rows.
- ⋆ Discard of perturbed rows.
- ⋆ Risk: rank deficiency.

Outline

Concluding remarks

Concluding remarks

- ⋆ Assembled approaches more robust than parallel approaches.
- ⋆ LSI-A preserve residual norm monotony for GMRES.
- ⋆ LSI-P more robust than LI-P.
- ⋆ Similar behavior for BICGSTAB and CG.
- ⋆ No fault, no overhead.

Perspectives

- ⋆ Consider the cost of interpolation.
- ⋆ Best combination of interpolation and selective checkpoint.
- ⋆ Real code.

Thank you for listening

http://hiepacs.bordeaux.inria.fr/

Concluding remarks

Concluding remarks

- ⋆ Assembled approaches more robust than parallel approaches.
- ⋆ LSI-A preserve residual norm monotony for GMRES.
- ⋆ LSI-P more robust than LI-P.
- ⋆ Similar behavior for BICGSTAB and CG.
- ⋆ No fault, no overhead.

Questions?

Perspectives

- ⋆ Consider the cost of interpolation.
- ⋆ Best combination of interpolation and selective checkpoint.
- ⋆ Real code.