Fault Tolerant Linear Algebra: goals and methods.

Julien Langou, University of Colorado Denver

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

5- ABFT-BLAS EXPERIMENTS (ERRASURE)

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

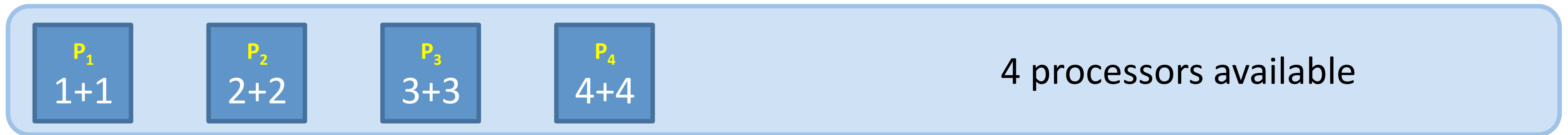
5- ABFT-BLAS EXPERIMENTS (ERRASURE)

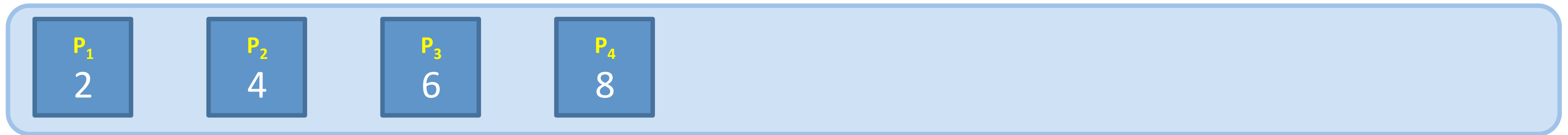
Goals

Perform reliable and efficent computation with unreliable units.

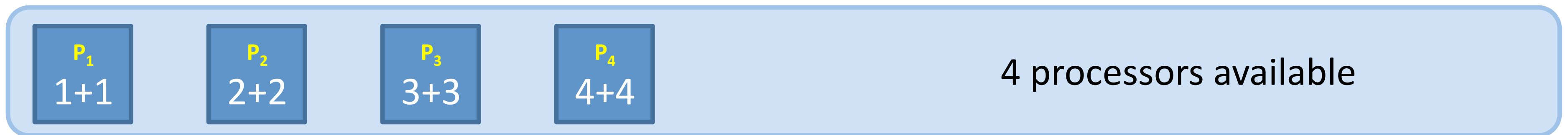
- Unreliable units: Process crash, hardware failure, erroneous communication, erroneous computation, …
- Our method: at the algorithm level.
- Motivation: cost effective, large unit count

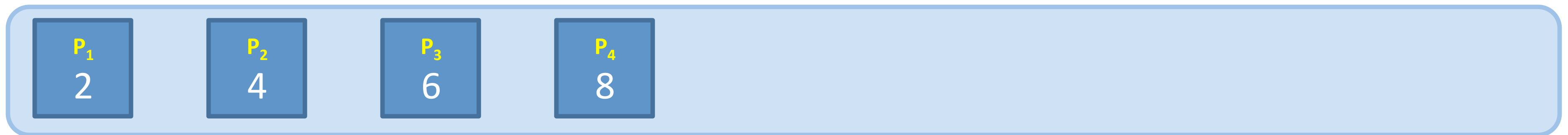
Errasure Problem





Error Problem





Errasure Problem

P_1		P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available

P_1		P_3	P_4	
2		6	8	Lost processor 2

Error Problem

P_1	P_2	P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available

P_1	P_2	P_3	P_4	
2	5	6	8	Processor 2 returns an incorrect result

Errasure Problem

P_1		P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available
2	4	6	8	Lost processor 2

- we know whether there is an errasure or not,

Error Problem

P_1	P_2	P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available
2	5	6	8	Processor 2 returns an incorrect result

- we do not know if there is an error,

Errasure Problem

P_1		P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available
2	4	6	8	Lost processor 2

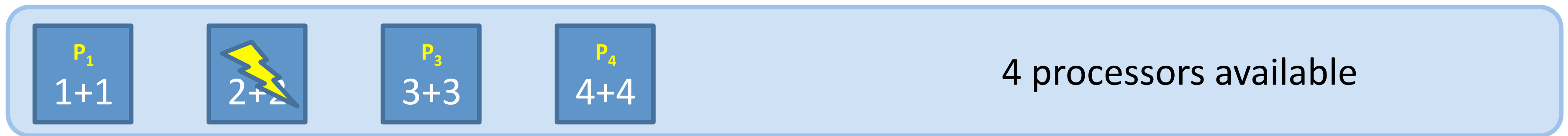
- we know whether there is an errasure or not,
- we know where the errasure is,

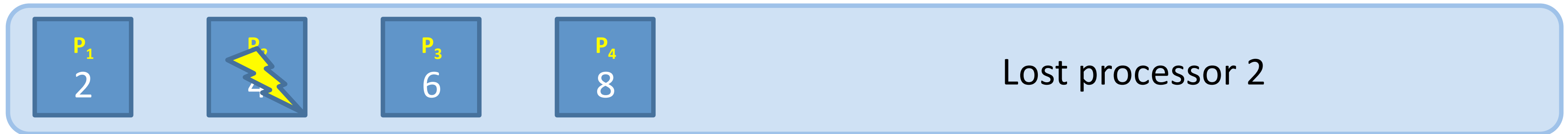
Error Problem

P_1	P_2	P_3	P_4	
1+1	2+2	3+3	4+4	4 processors available
2	5	6	8	Processor 2 returns an incorrect result

- we do not know if there is an error,
- assuming we know that an error occurs, we do not know where it is

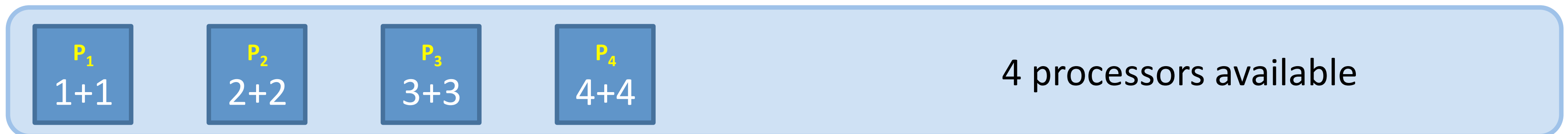
Errasure Problem

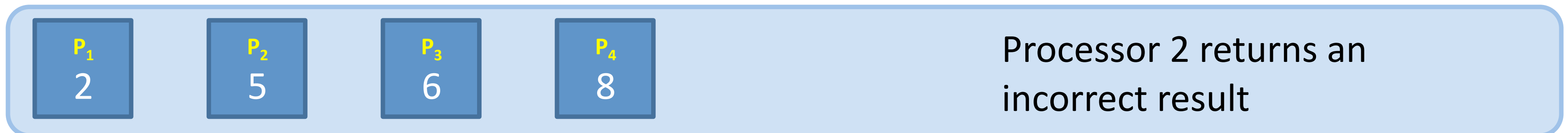




- we know whether there is an errasure or not,
- we know where the errasure is,
- so we only need to recover

Error Problem





- we do not know if there is an error,
- assuming we know that an error occurs, we do not know where it is
- we also need to recover

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

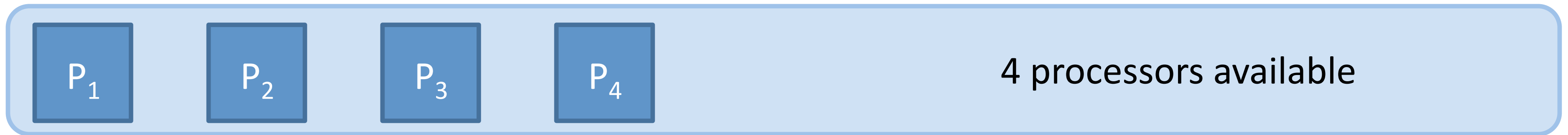
2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

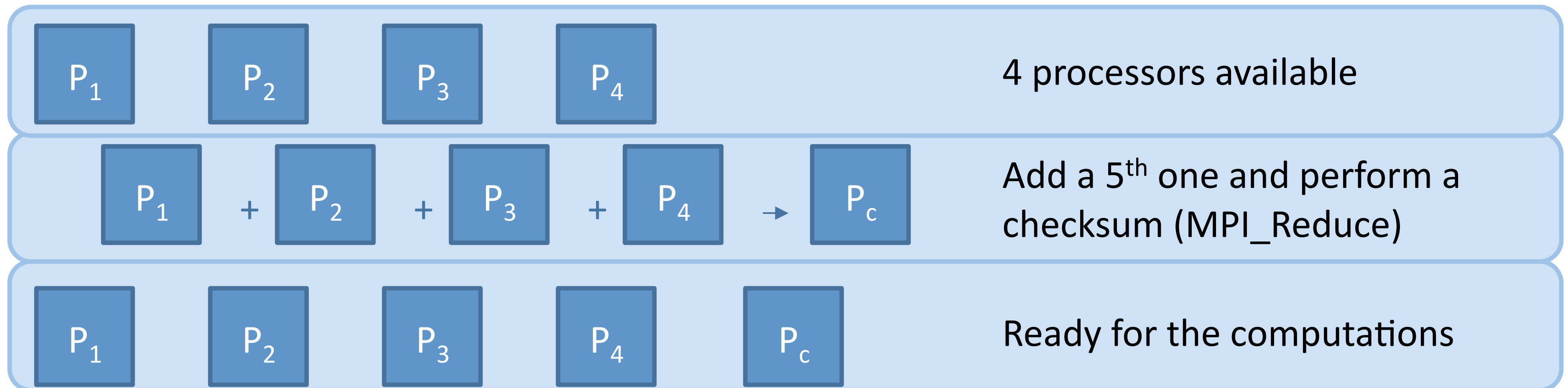
4- ABFT-BLAS LIBRARY

5- ABFT-BLAS EXPERIMENTS (ERRASURE)

Diskless checkpointing



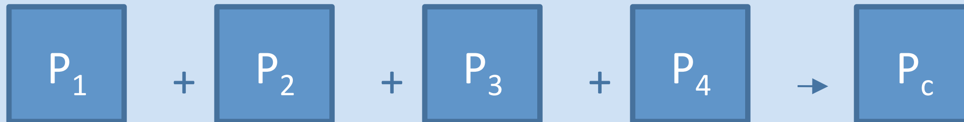
Diskless checkpointing



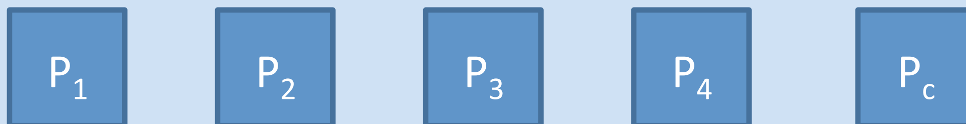
… … …

Diskless checkpointing

4 processors available

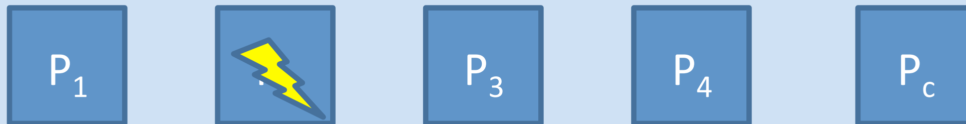


Add a 5th one and perform a checksum (MPI_Reduce)

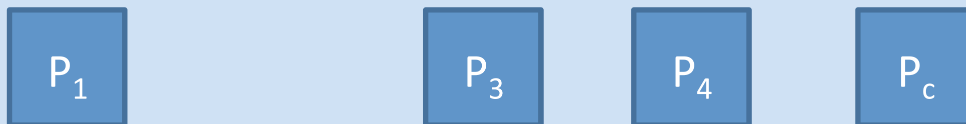


Ready for the computations

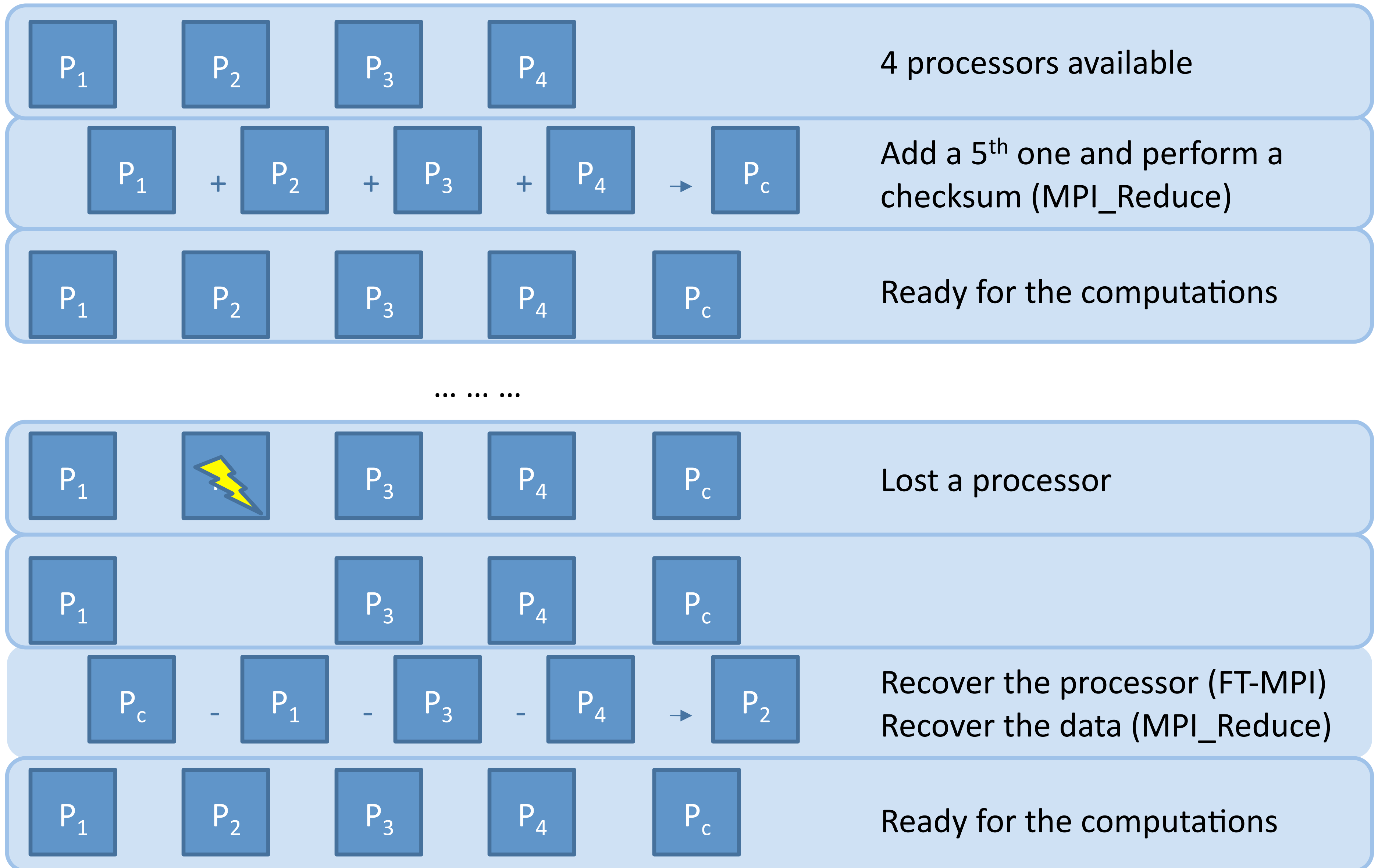
… … …



Lost a processor



Diskless checkpointing

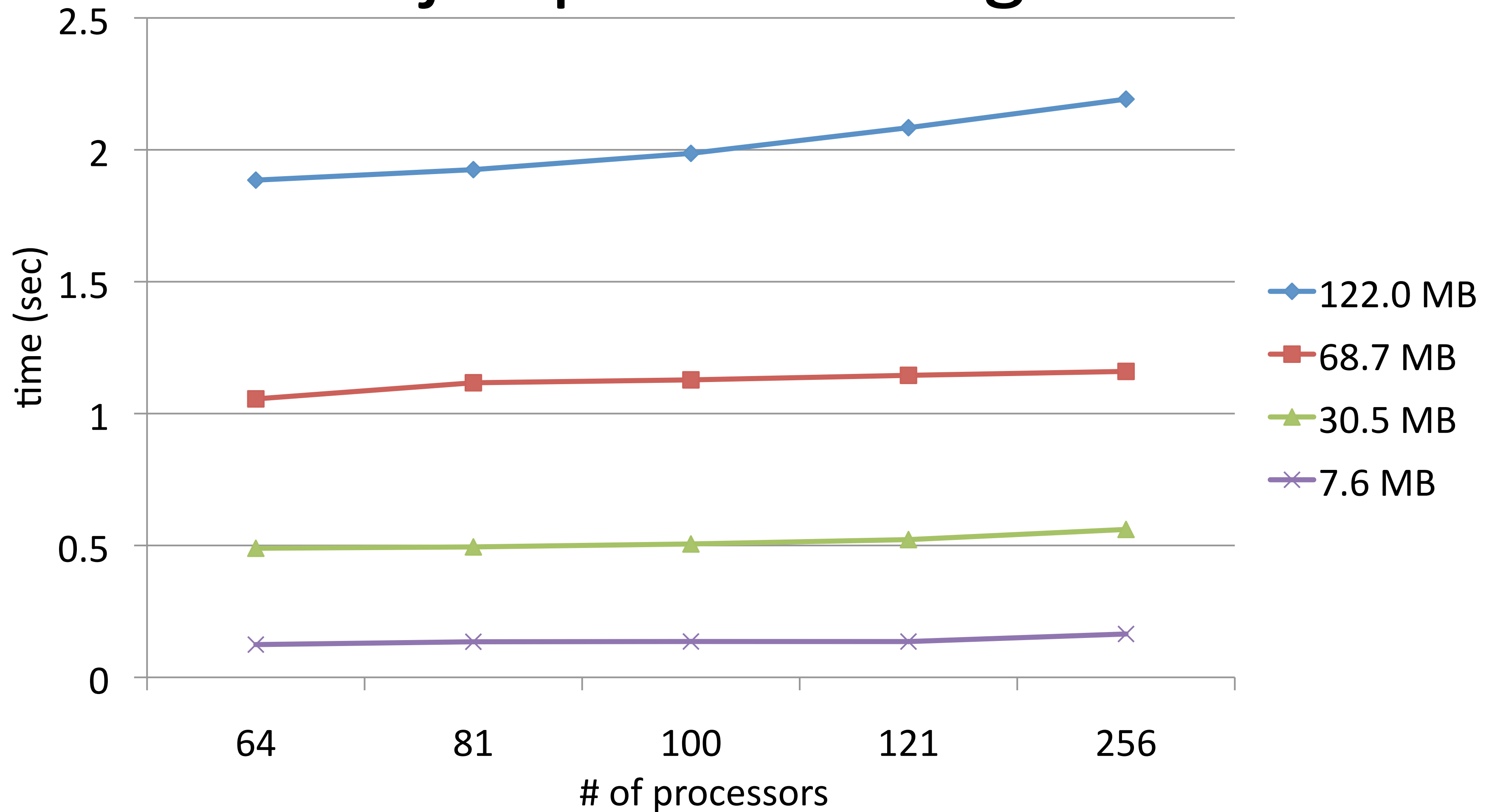


Diskless checkpointing (remarks)

- You can use either floating-point arithmetic or binary arithmetic for the checksum

- Multiple failures/errors supported through Reed-Solomon algorithm, optimal algorithm in the sense that, to support p simultaneous failures/errors, only need to add p processes.

Time for a MPI_Reduce (using MVAPICH) on Infiniband on jacquard.nersc.gov



Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

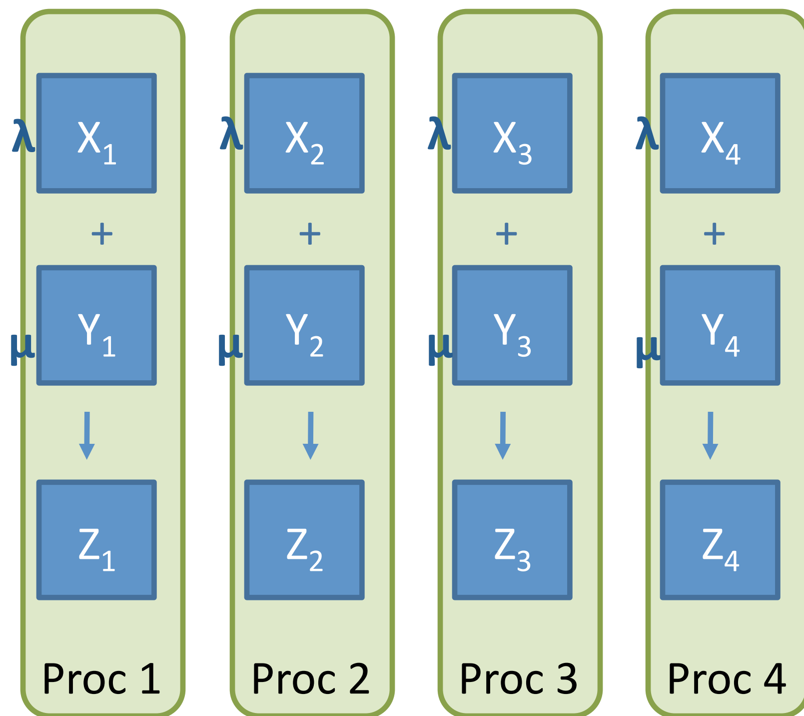
5- ABFT-BLAS EXPERIMENTS (ERRASURE)

ABFT = Algorithm Based Fault Tolerance.

- K. Huang, J. Abraham, "*Algorithm-Based Fault Tolerance for Matrix Operations*," IEEE Trans. on Comp. (Spec. Issue Reliable & Fault-Tolerant Comp.), C-33, 1984, pp. 518-528.
- If checkpoints are performed in floating-point arithmetic then we can exploit the linearity of the mathematical relations on the object to maintain the checksums

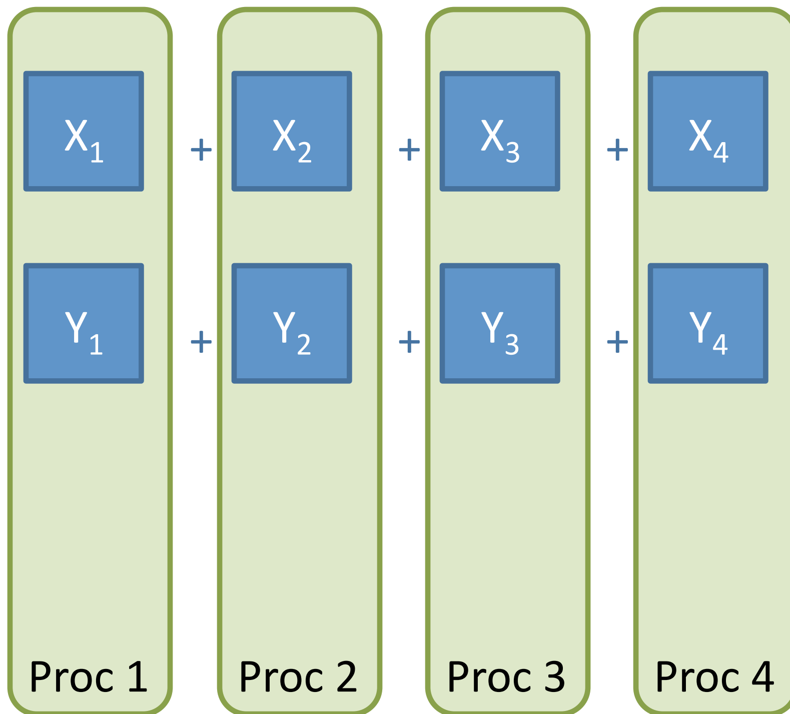
ABFT concept in an example

We want to perform z = λx+μy.



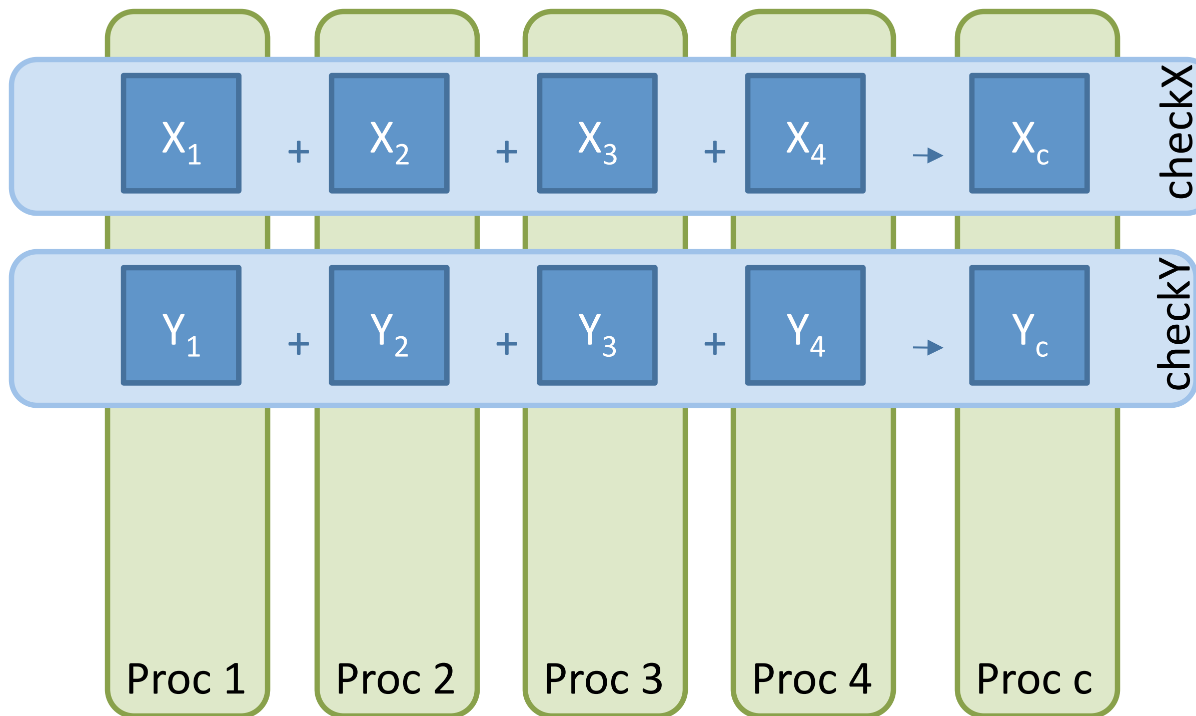
ABFT concept in an example

We want to perform $z = \lambda x+\mu y$.



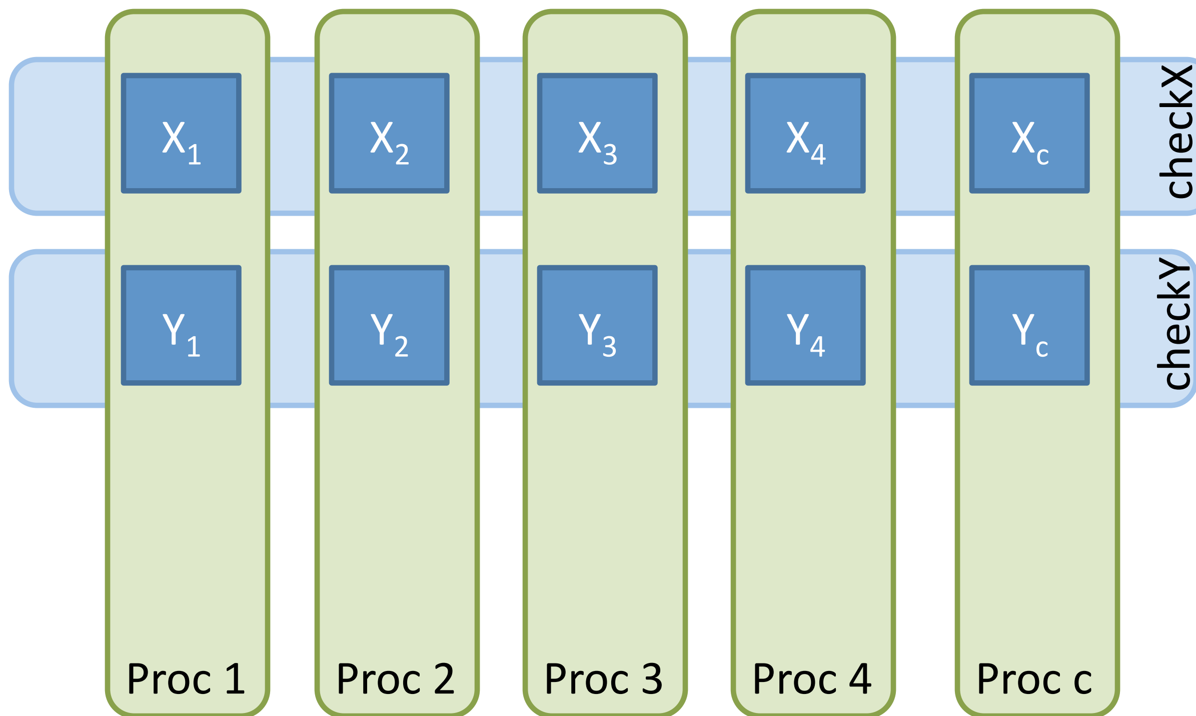
ABFT concept in an example

We want to perform z = λx+μy.



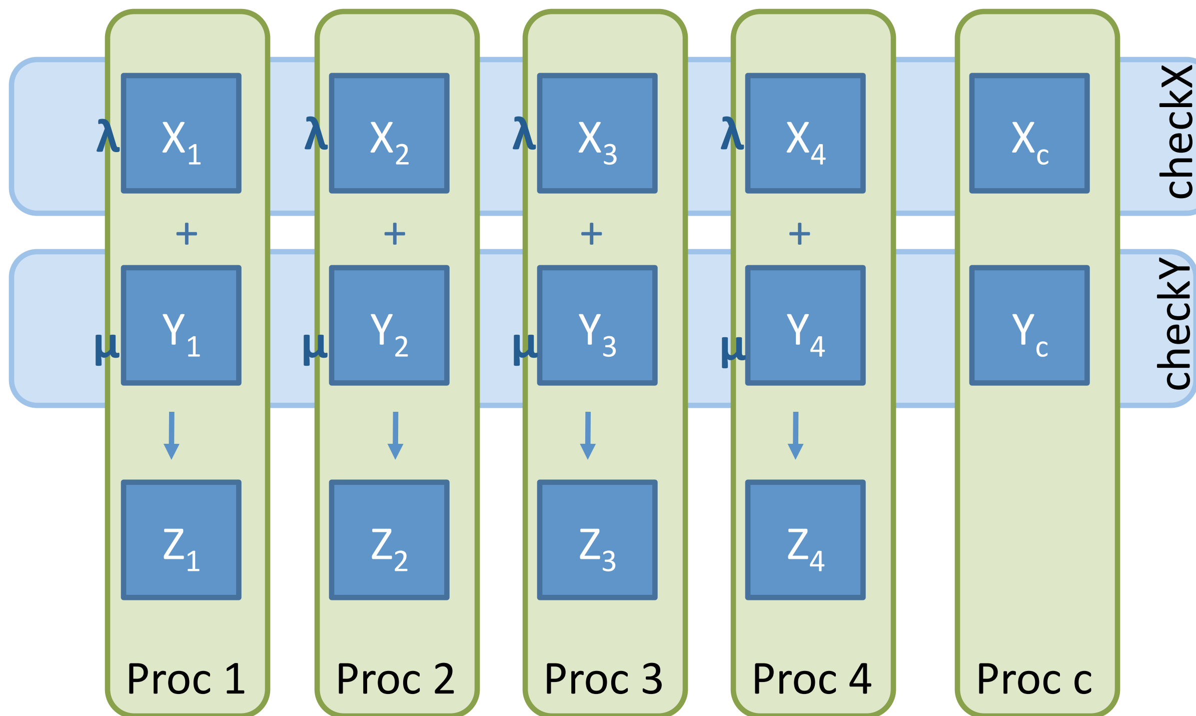
ABFT concept in an example

We want to perform $z = \lambda x + \mu y$.



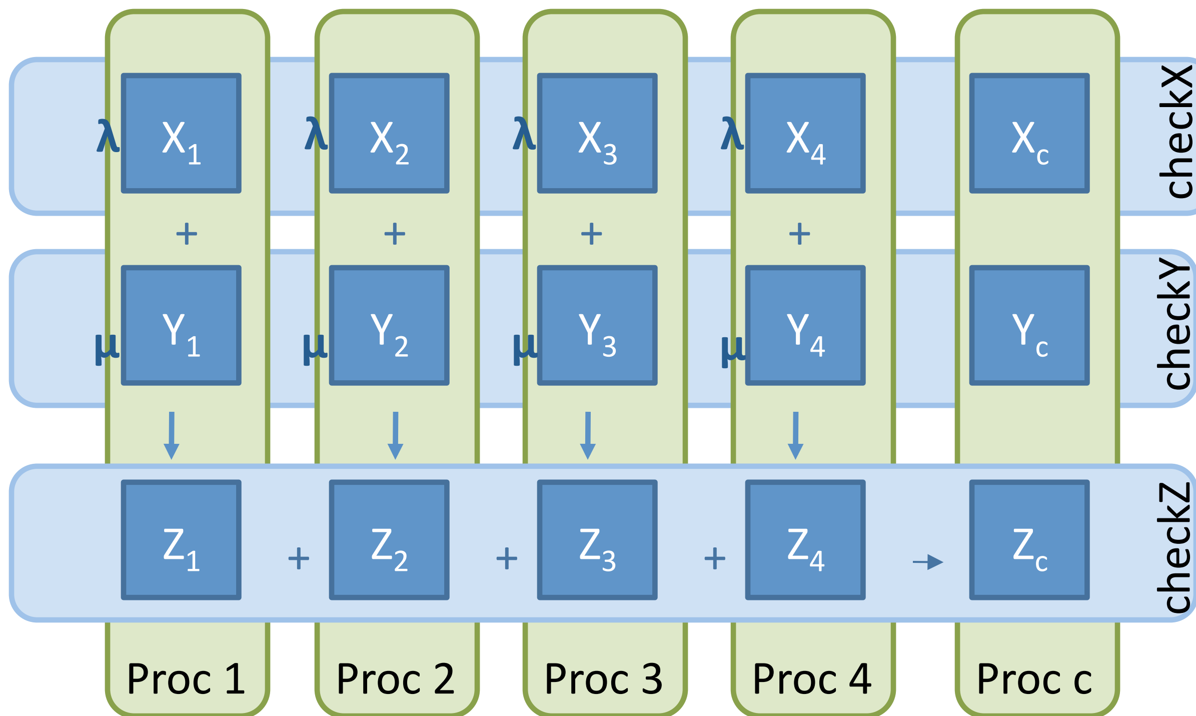
ABFT concept in an example

We want to perform z = λx+μy.



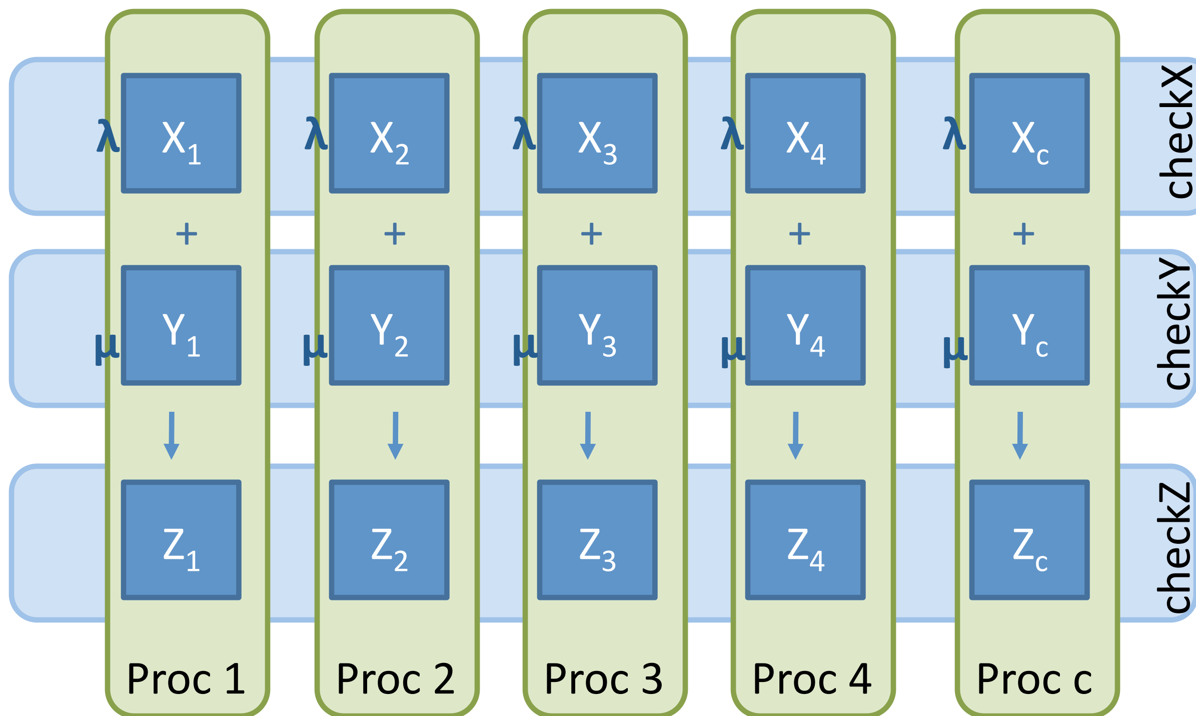
ABFT concept in an example

We want to perform $z = \lambda x + \mu y$.



ABFT concept in an example

We want to perform z = λx+μy.



No overhead to compute the checksum of Z.

Property used: $(\lambda x_1+\mu y_1)+(\lambda x_2+\mu y_2)=\lambda(x_1+x_2)+\mu(y_1+y_2)$, distributivity of external multiplication over intenal addition, associativity of internal addition.

ABFT summary.

- Relies on floating-point arithmetic checksums
- Exploit the checksum processors
- Algorithms exist for any linear operations:
  - AXPY, SCAL, (BLAS1)
  - GEMV (BLAS2)
  - GEMM (BLAS3)
  - LU, QR, Cholesky (LAPACK)
  - FFT

Our contribution (1)

- Lack of generalization in the previous approach which has restricted the number algorithms used: Cholesky, QR through Gram-Schmidt, LU without pivoting.
- With an ABFT BLAS and the LAPACK algorithm, we have developped:
  - QR with Houesholder reflection,
  - LU with pivoting
  - Hessenberg reduction

Our contribution (2)

- If there is no error then ABFT guarantees that the checksum of L and U are consisten at the end of the LU factorization.
- If there is an error, you can then detect it.
- However you can not correct it in the case where the error propagate.
- (Then why even using ABFT?)
- We have a light-weight mechanism ($O(n^2)$) to detect errors before they propagate

Our contribution (3)

- Error correcting codes are known to be unstable in floating-point arithmetic.
- We have developed a stable error correcting code (although naïve and not optimal, it works for us and is efficient enough).

Our contributions

1. Generalize ABFT to ``all'' LAPACK algorithms
2. Avoid error propagation
3. Stable error correcting code in floating-point arithmetic

=> Maybe ABFT might work after all!

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

5- ABFT-BLAS EXPERIMENTS (ERRASURE)

Error DETECTION: Residual checking

- To detect an error in

  $C \leftarrow \alpha A^* B + \beta C_{in}$ (1)

  1. Save C_{in}
  2. Perform $C \leftarrow \alpha A^* B + \beta C_{in}$
  3. Take random (vector) x, check that

     $|| Cx - \alpha(A^* (B x)) + (\beta C_{in} x) || < \varepsilon$
  4. If check is not good, start again from step 2.

- Works with almost anything (e.g. $A = VDV^T$)

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

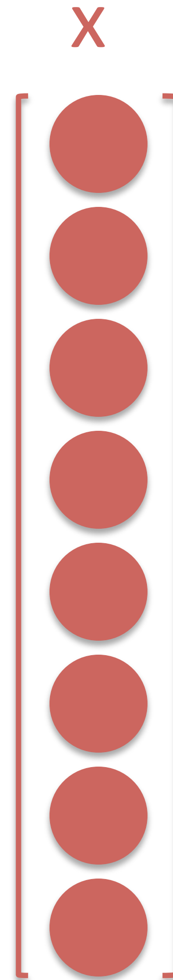
3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

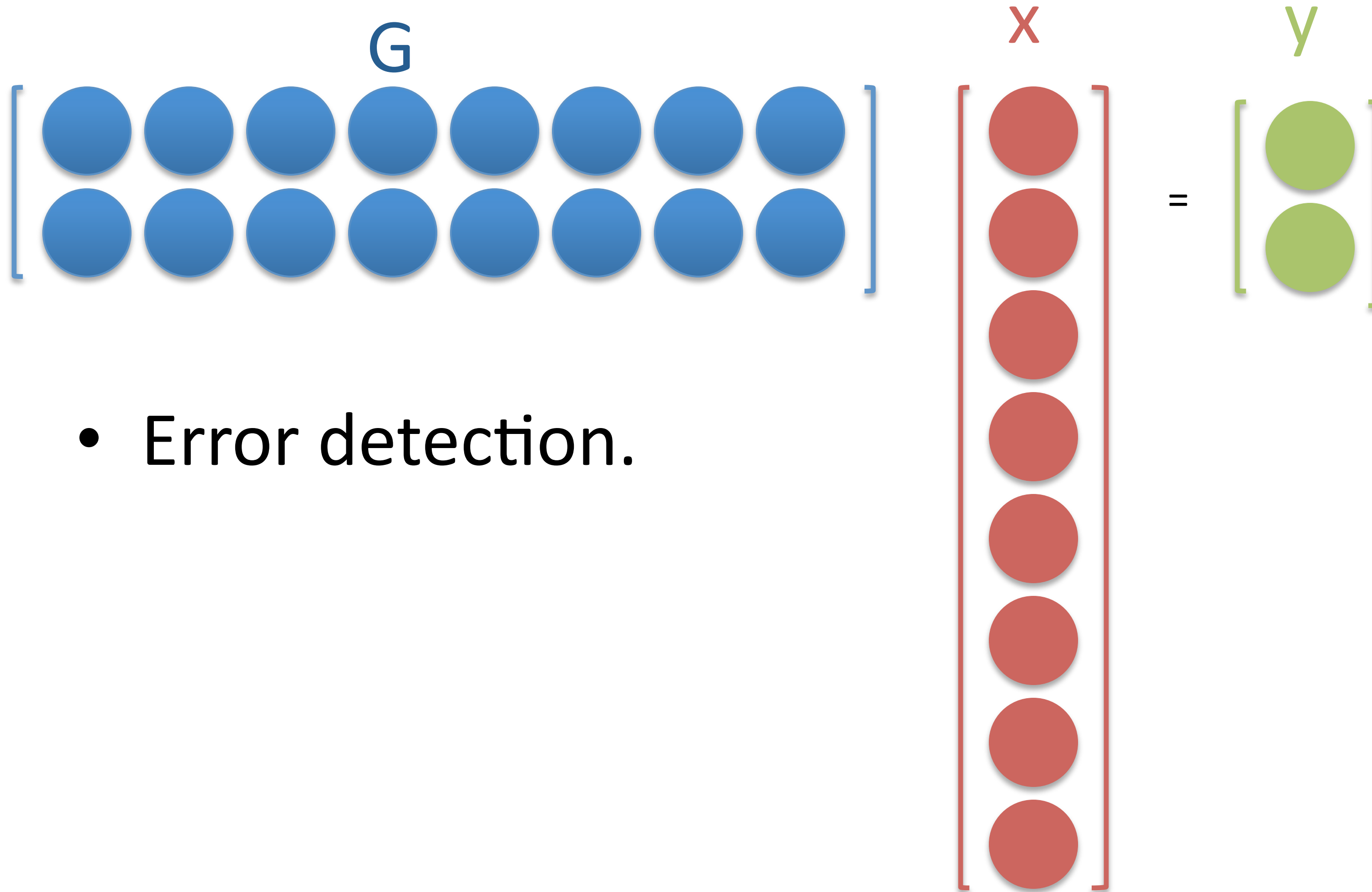
5- ABFT-BLAS EXPERIMENTS (ERRASURE)

Encoding

- Error detection.

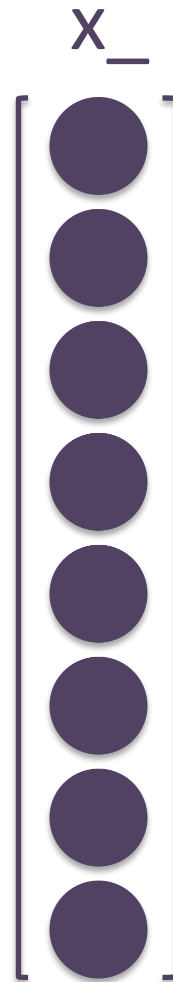


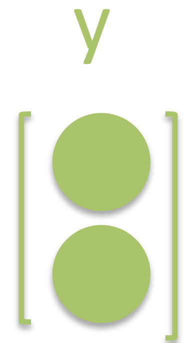
Encoding



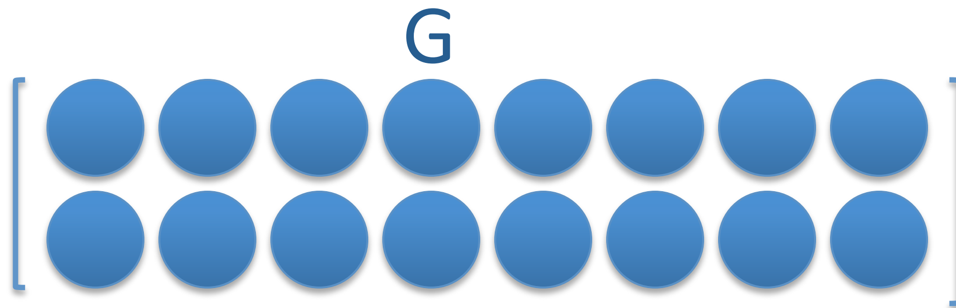
- Error detection.

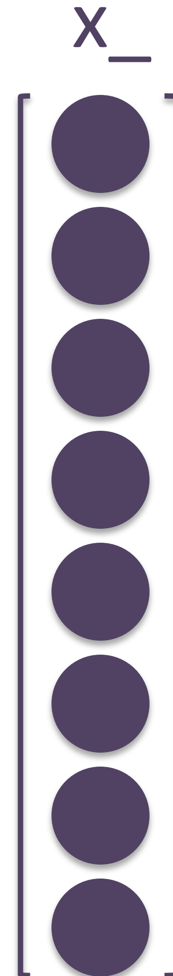
Encoding

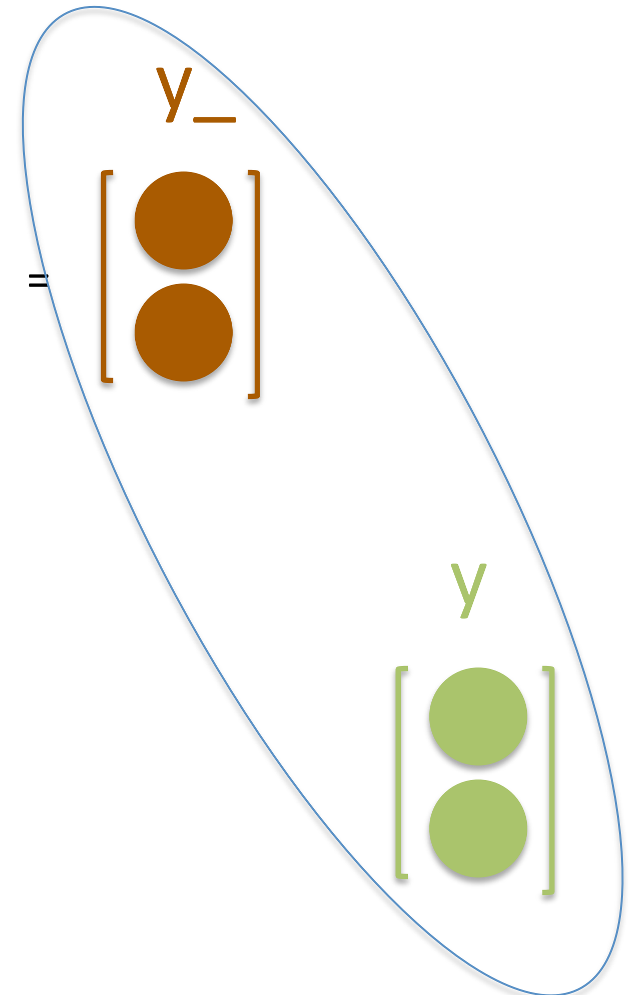




Encoding





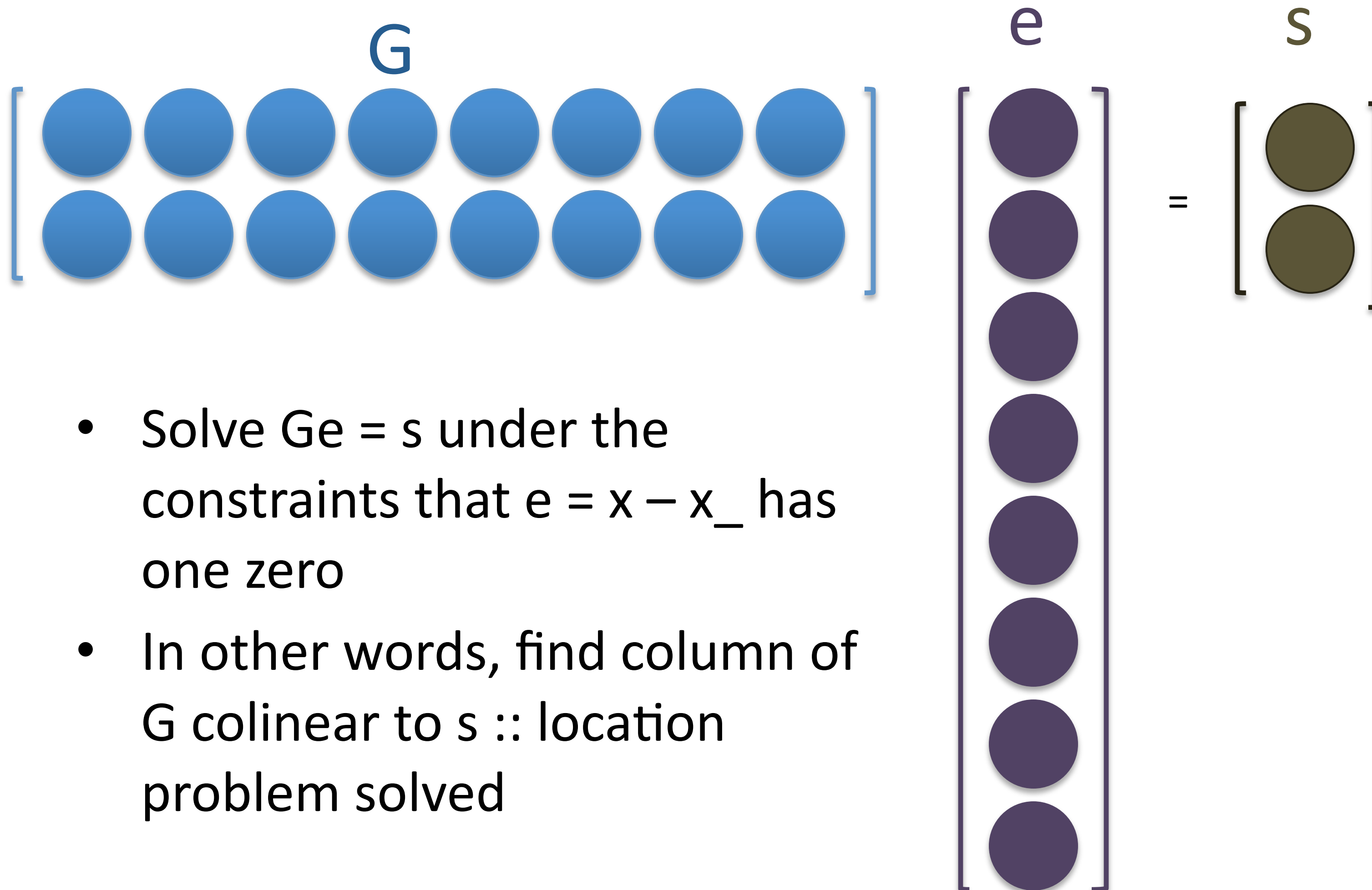


- Error **detection**.
- Note that G could have been 1-by-n (no need for 2-by-n)
- Note that **correction** is possible if **location** is known

$s = y - y_$

$G \, e = s$

- Solve Ge = s under the constraints that e has one zero
- In other words, find column of G colinear to s :: location problem solved


G
e
s
=
Solve Ge = s under the constraints that e = x – x_ has one zero
In other words, find column of G colinear to s :: location problem solved

$$G = \begin{bmatrix} 3 & 1 & 3 & 1 & 0 & 1 & 7 & 9 \\ 2 & 5 & 8 & 2 & 1 & 1 & 1 & 3 \end{bmatrix} \quad e = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = s = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

- Solve Ge = s under the constraints that e has one zero
- In other words, find column of G colinear to s :: location problem solved

$$G \, e = s$$

$$\begin{bmatrix} 3 & 1 & 3 & 1 & 0 & 1 & 7 & 9 \\ 2 & 5 & 8 & 2 & 1 & 1 & 1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

- Solve Ge = s under the constraints that e has one zero
- In other words, find column of G colinear to s :: location problem solved

Reed-Solomon

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \quad e = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} = s = \begin{bmatrix} 2 \\ 8 \end{bmatrix}$$

- Solve Ge = s under the constraints that e has one zero
- In other words, find column of G colinear to s :: location problem solved

Reed-Solomon

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{e} = \underbrace{\begin{bmatrix} 2 \\ 8 \end{bmatrix}}_{s}$$

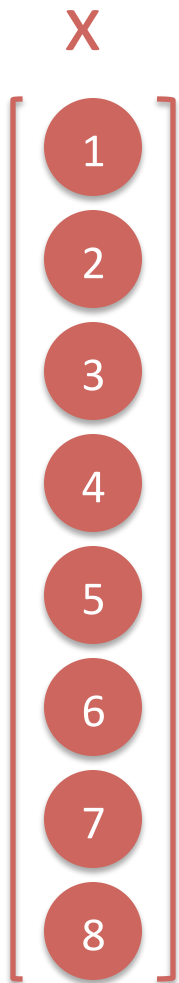
- Solve Ge = s under the constraints that e has one zero
- In other words, find column of G colinear to s :: location problem solved

$$G = \begin{bmatrix} 3 & 1 & 3 & 1 & 0 & 1 & 7 & 9 \\ 2 & 5 & 8 & 2 & 1 & 1 & 1 & 3 \\ 1 & 4 & 9 & 3 & 0 & 0 & 5 & 0 \end{bmatrix} \quad e = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = s = \begin{bmatrix} 4 \\ -7 \\ -4 \end{bmatrix}$$

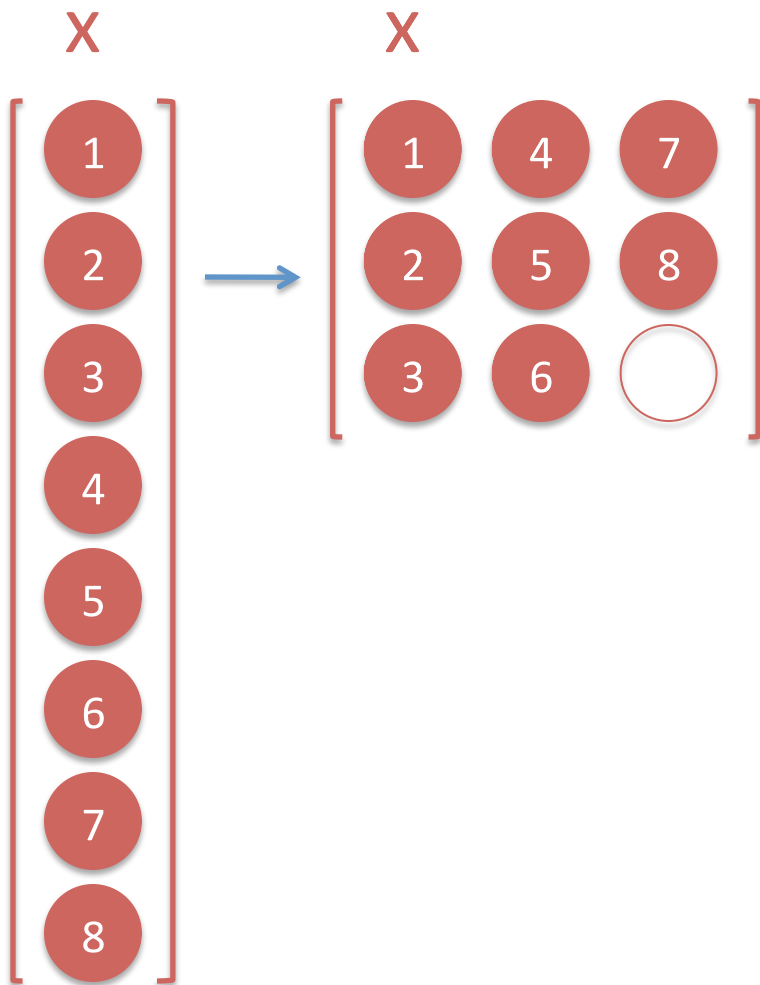
- Solve Ge = s under the constraints that e has two zero
- In other words, find the unique two columns that generates s. Complexity choose(n,2) = 28

	Stable	Recovery Cost	Extra Memory
Reed Solomon	✗	✓	nberr. ✓
Random	✓	✗	nberr. ✓

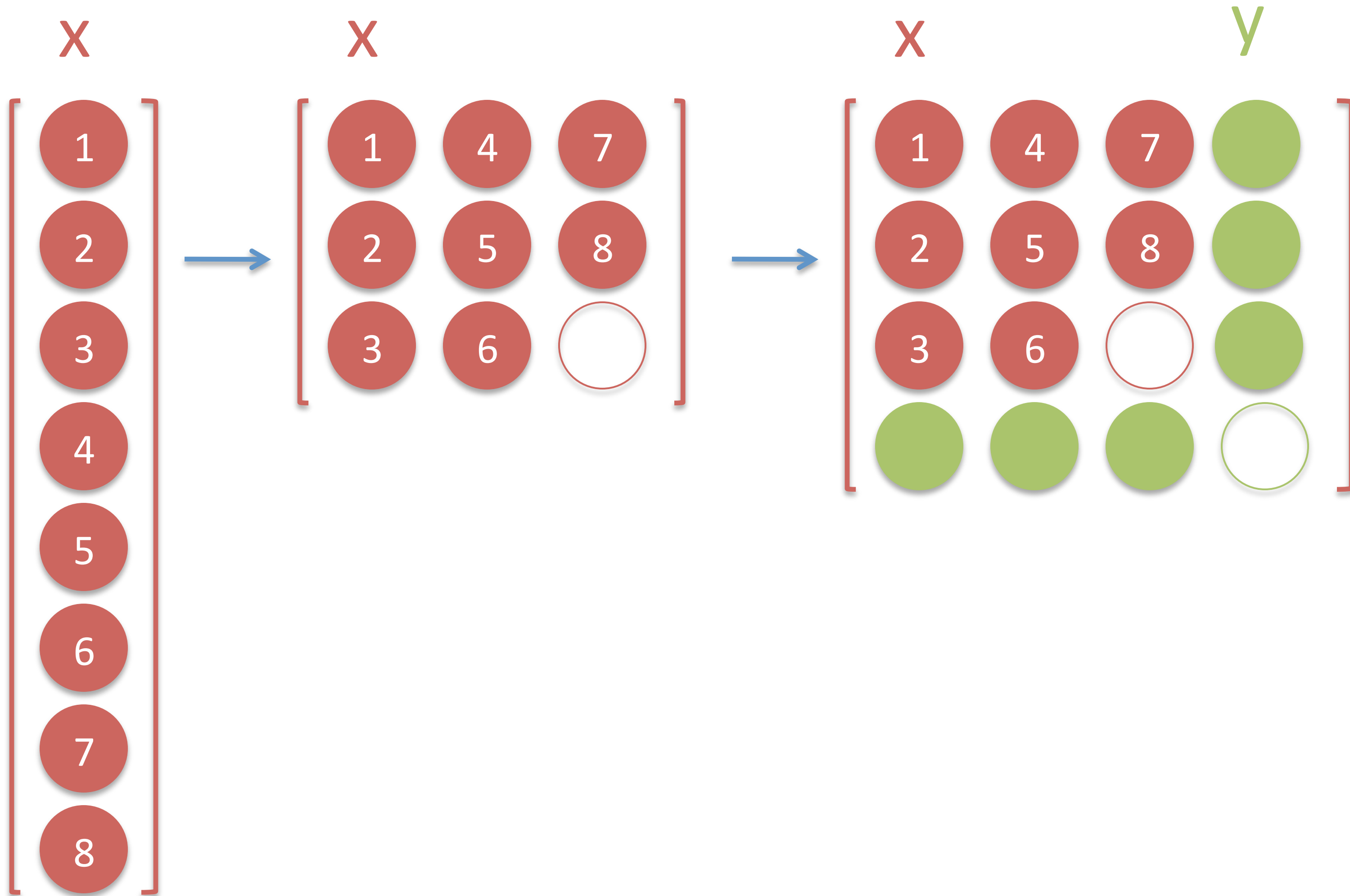
Encoding



Encoding

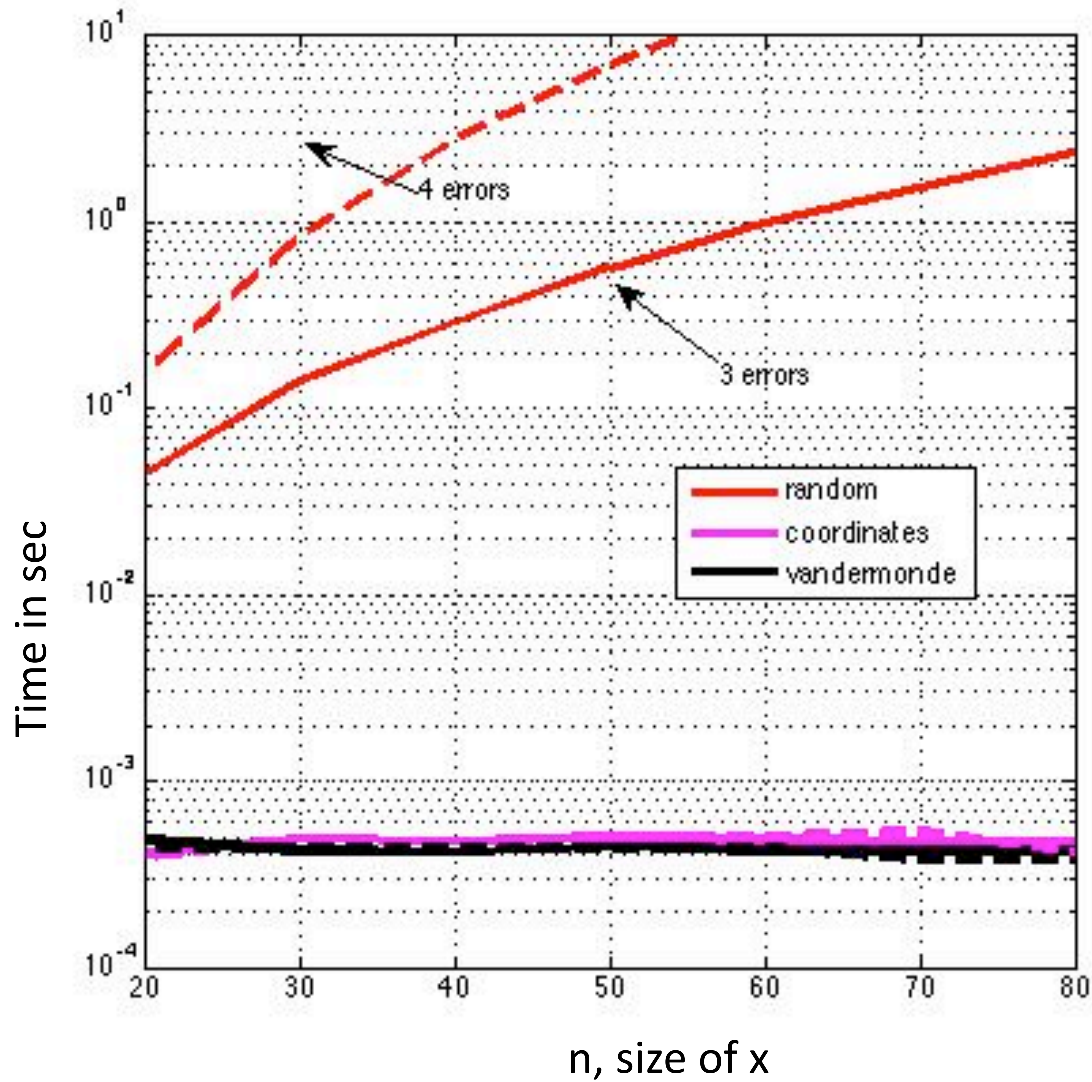


Encoding

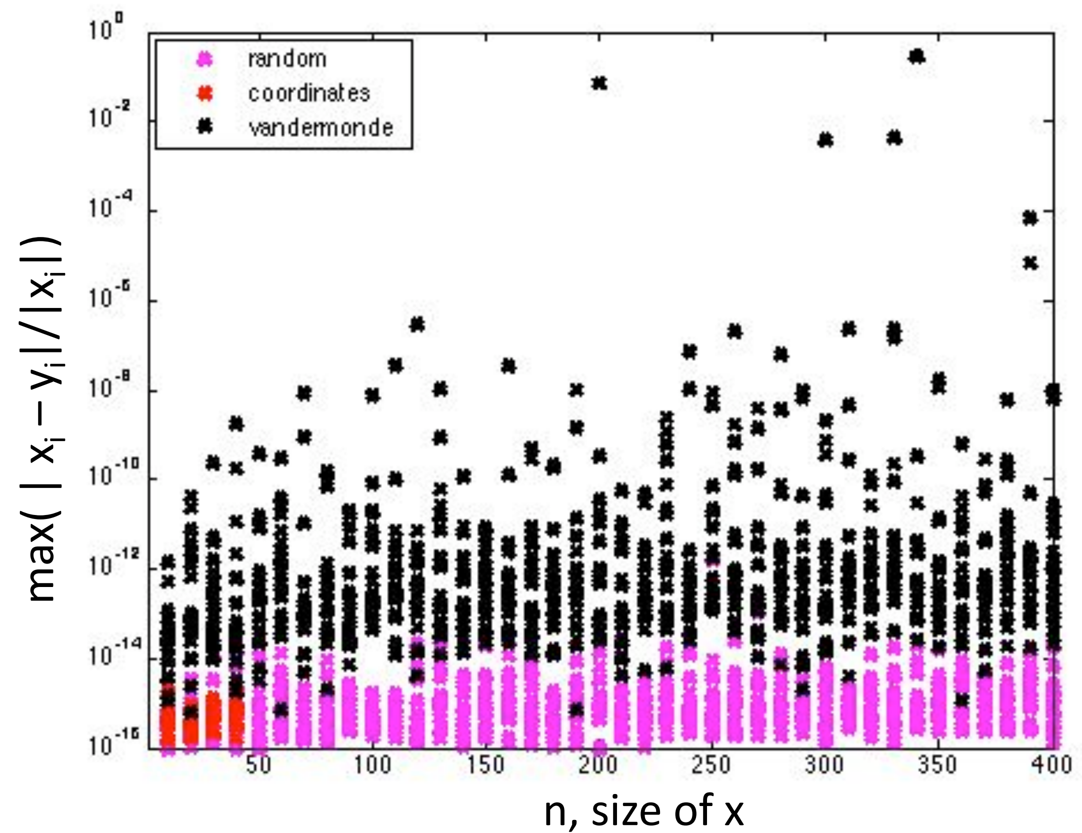


	Stable	Recovery Cost	Extra Memory
Reed Solomon	✗	✓	nberr. ✓
Random	✓	✗	nberr. ✓
Coordinate	✓	✓	Sqrt(n) ✗

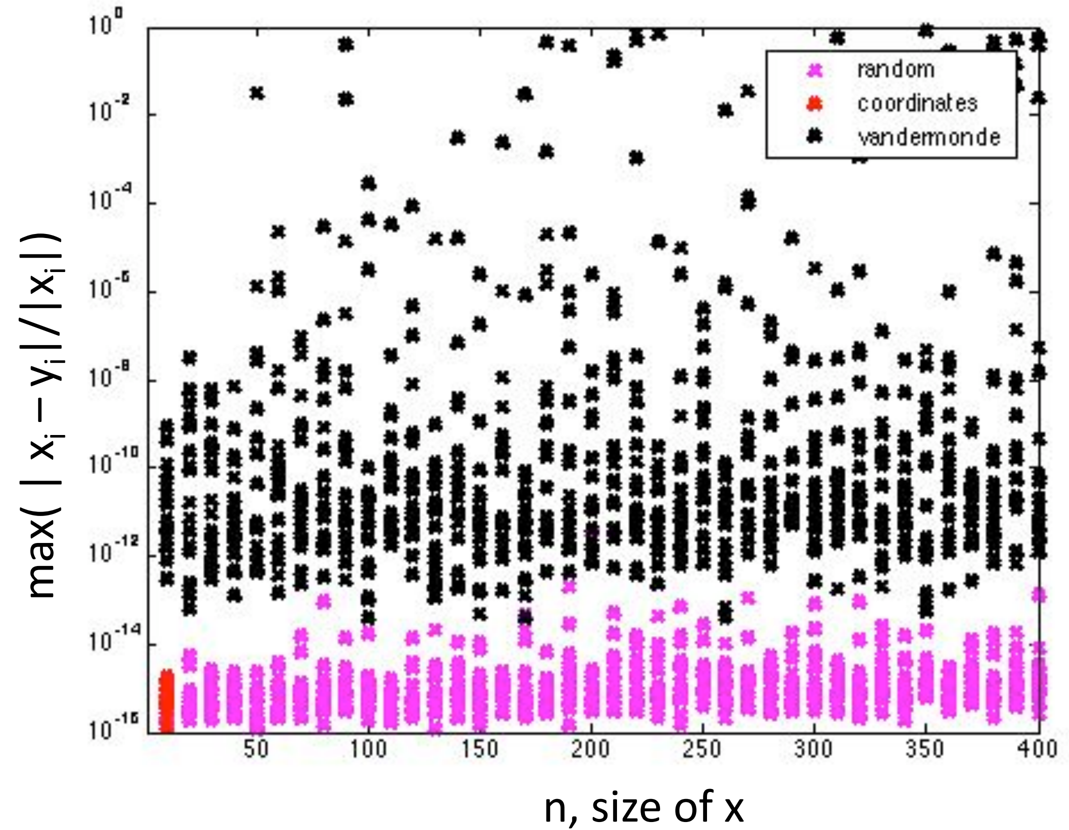
Timing for recovery

- 

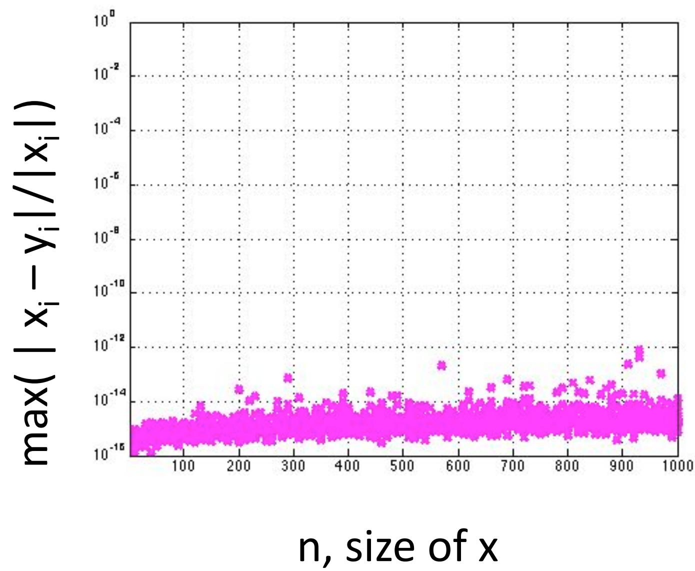
- Accuracy comparison after recovery.
- maxerr = 3
- nberr =3

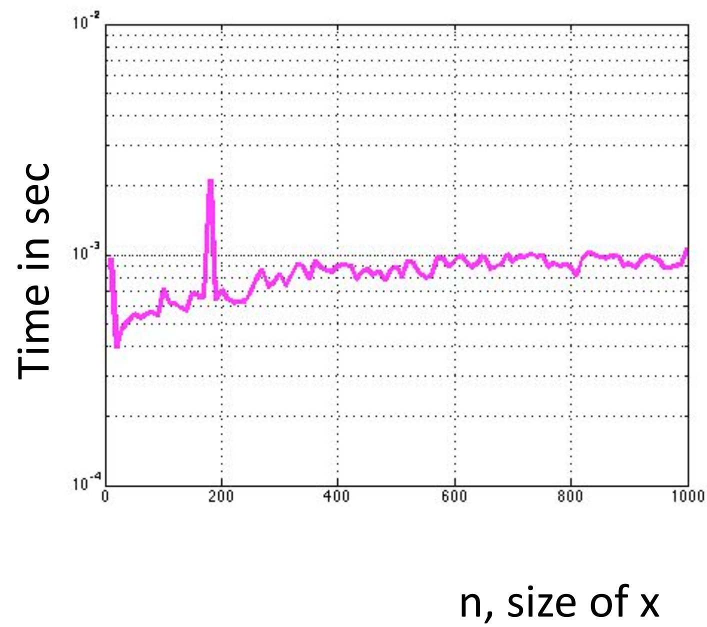


- Accuracy comparison after recovery.
- maxerr = 4
- nberr =4



- maxerr = 10
- nberr =10





Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

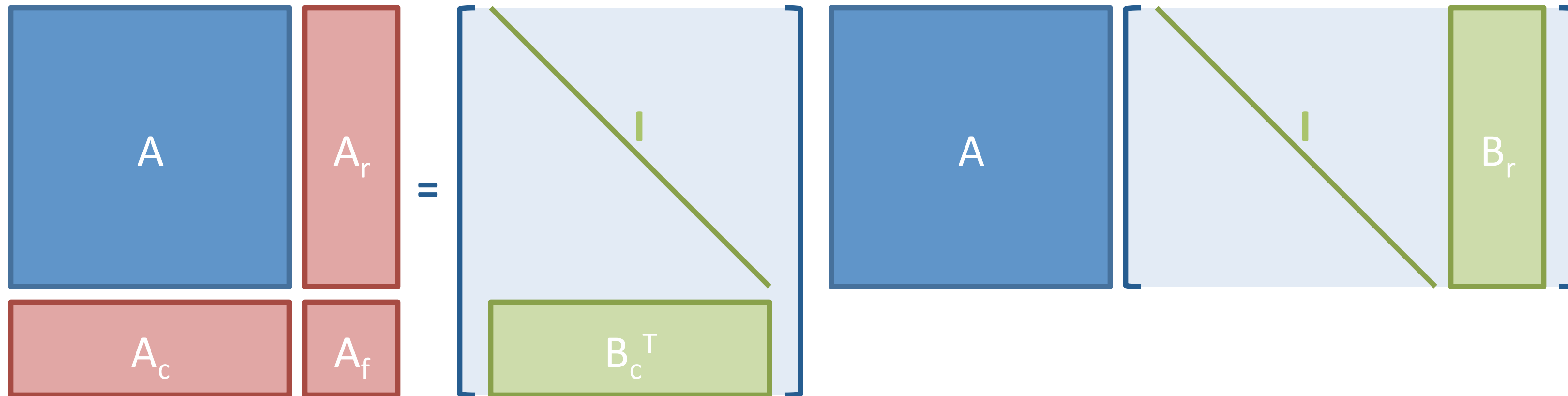
3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

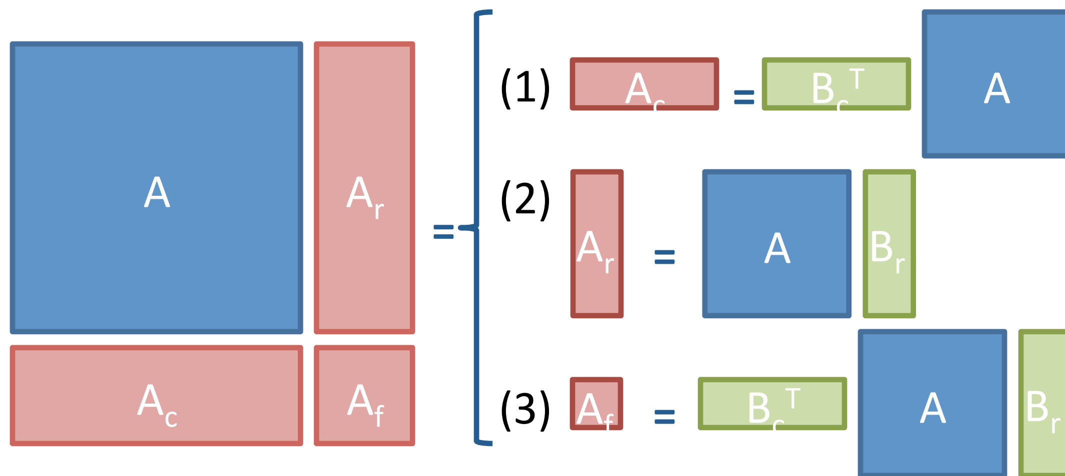
4- ABFT-BLAS LIBRARY

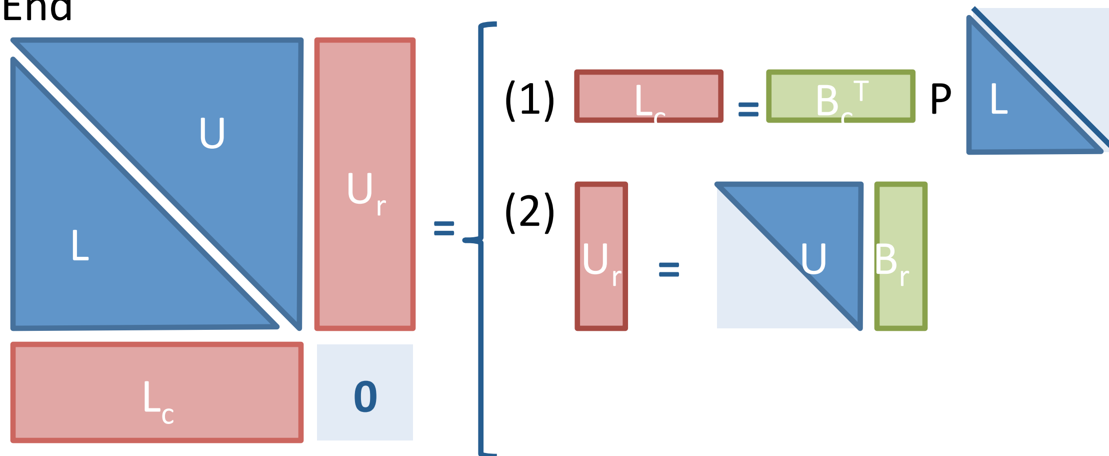
5- ABFT-BLAS EXPERIMENTS (ERRASURE)

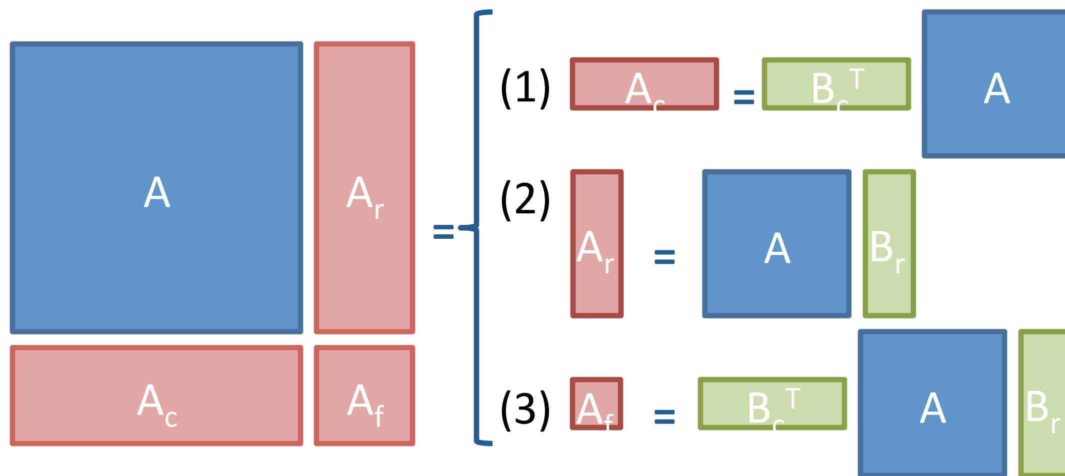
LU factorization:

Starting from the encoding:

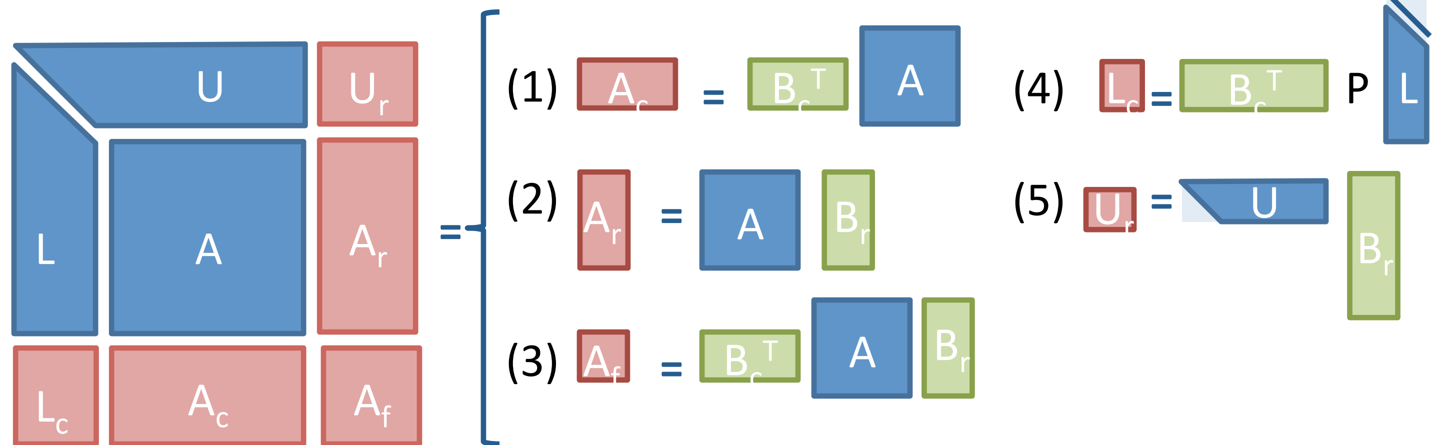



A
A_r
A_c
A_f
=
(1) A_c = B_c^T A
(2) A_r = A B_r
(3) A_f = B_c^T A B_r

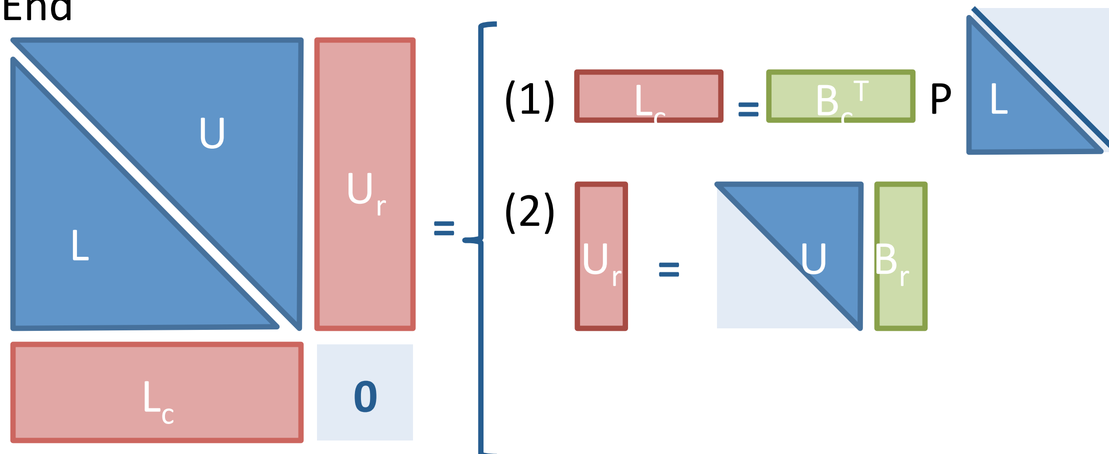

End
U
L
U_r
L_c
0
=
(1) L_c = B_c^T P L
(2) U_r = U B_r



For j=1:nb:n,



End



Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

5- ABFT-BLAS EXPERIMENTS (ERRASURE)

Experiments on MM

- **Goal:**
  - Write a FT-PDGEMM (Fault Tolerant matrix Matrix Multiply)

$$C \leftarrow \alpha\, A * B + \beta\, C$$

- **Testing:**
  - Perform FT-PDGEMM in a loop and check results with residual checking

$$C_{out}\, x - [\alpha\, (A\, (B\, x)) + \beta\, C_{in}\, x]$$

on top of this add on automatic process killer.

ABFT-BLAS: a Parallel Fault Tolerant library BLAS based on ABFT techniques

- Constructed on top of FT-MPI.
- Provides users a fault-tolerant environment:
  - Detect failures
  - Recover data automatically
  - Enables the user to stack computational routines the one on top of the others (two general problems)
  - Goal: research library for conducting experiments on fault tolerance
- Provides developpers with an automatic process killer

EXAMPLE CODE

```
int rc;
struct Vector v;
struct Matrix a;
struct Dataworld worldmpi;
struct Global_ddata normv;
struct Global_idata nbr_iter;
…
rc = MPI_Init(&argc, &argv);
rc = init_world(&worldmpi, p, q, rc);
rc = get_info_on_grid(&worldmpi, &me,
              &myrow, &mycol, &nprow, &npcol);
…
rc = allocate_vector(&v, POS_ROW, 0, nb_n, &worldmpi, "v");
rc = allocate_matrix(&a, m, n, nb_m, nb_n, &worldmpi, "a");
rc = allocate_dglobal(&normv, 1, &worldmpi);
rc = allocate_iglobal(&nbr_iter, 1, &worldmpi);
…
if (!worldmpi.recovering)
{
… here goes the user code to initialize objects …
rc = make_checksum_matrix(&a, &worldmpi);
rc = make_checksum_vector(&v, &worldmpi);
}
```

```
if (worldmpi.user_state == 0)
{
rc = ftdnrm2(&worldmpi, &v, normv.data);
worldmpi.user_state = 1;
}
if (worldmpi.user_state == 1)
{
… here goes any call to the ABFT-BLAS numerical routines …
worldmpi.user_state = 2;
}

free_vector(&v);
free_matrix(&a);
free_dglobal(&normv);
free_iglobal(&nbr_iter);
exit(0);
```

Fault-tolerant Linear Algebra: Goals and Methods.

0- GOALS

0.1- ERRASURE OR ERROR?

1- METHODS

1.1- ERRASURE: DISKLESS CHECKPOINTING AND ROLLBACK

1.2- ERRASURE & ERROR: ABFT: ALGORITHM BASED FAULT TOLERANCE

1.3- OTHERS

2- ERROR: DETECTING/LOCATING/CORRECTING IN FLOATING-POINT ARITHMETIC

3- NOVEL ABFT-ALGORITHM (GEMM, LU, QR, ETC.) (ERRASURE OR ERROR)

4- ABFT-BLAS LIBRARY

5- ABFT-BLAS EXPERIMENTS (ERRASURE)

PDGEMM.

PDGEMM :

For k = 1:nb:n,

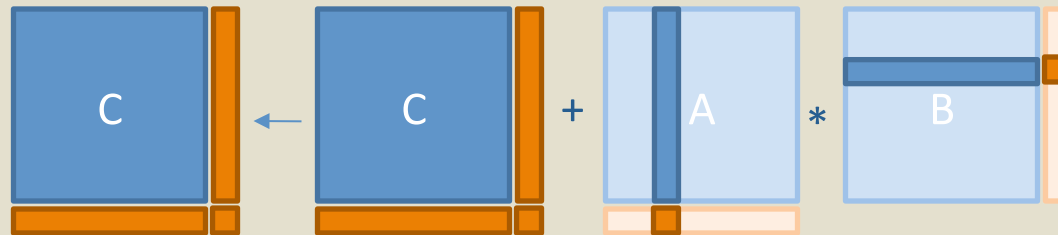
End For

PDGEMM-SUMMA
$\frac{2n^3}{p}\gamma + 2(n + 2\sqrt{p} - 3)(\frac{n}{\sqrt{p}}\beta)$

ABFT-PDGEMM.

ABFT-PDGEMM :

For k = 1:nb:n,



End For

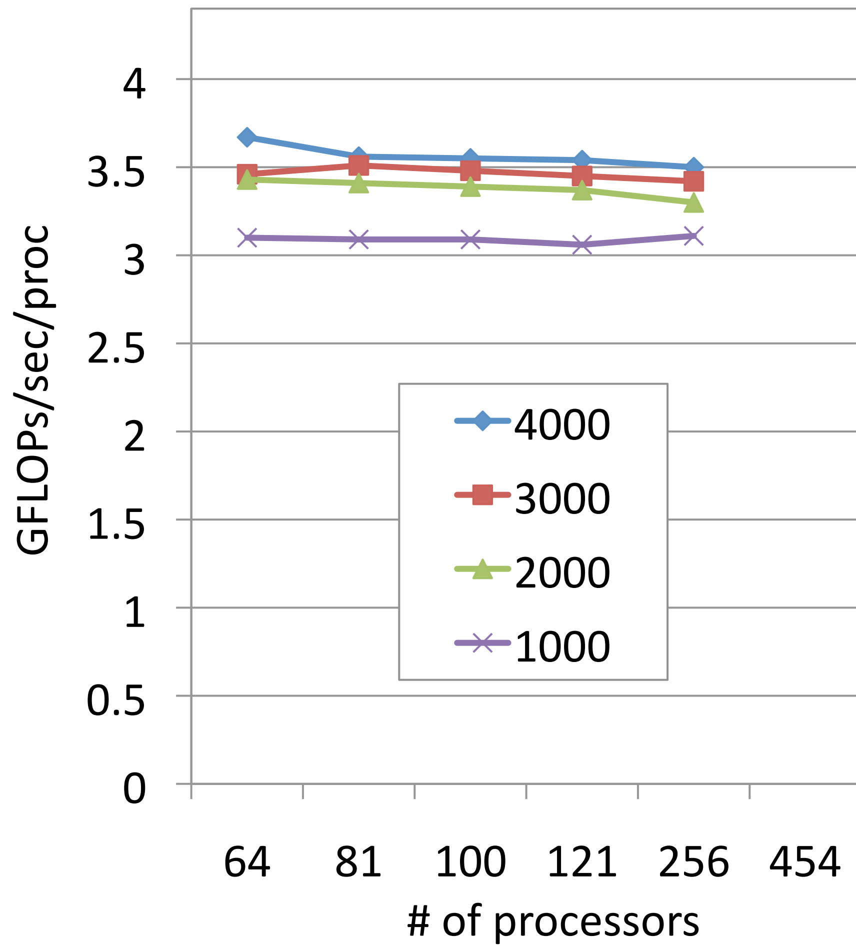
- The algorithm maintains the consistency of the checkpoints of the matrix C naturally.

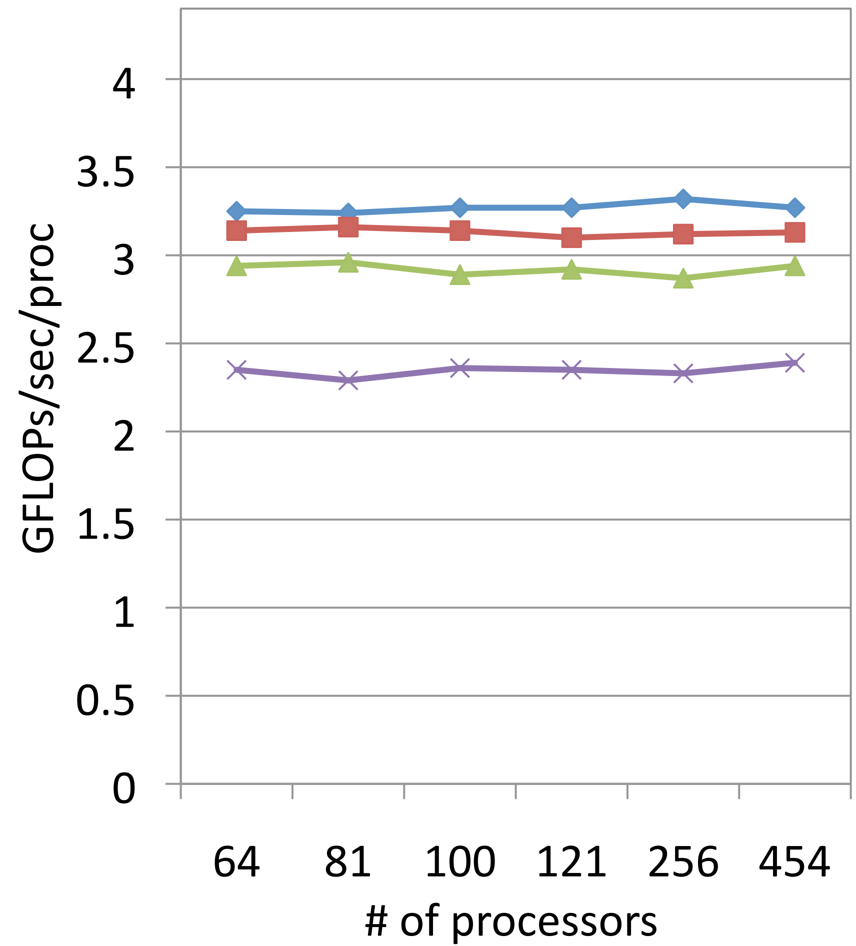
PDGEMM-SUMMA	ABFT-PDGEMM-SUMMA
$\frac{2n^3}{p}\gamma + 2(n + 2\sqrt{p} - 3)(\frac{n}{\sqrt{p}}\beta)$	$\frac{2n(n+nloc)^2}{p}\gamma + 2(n + 2\sqrt{p} - 3)(\frac{(n+nloc)}{\sqrt{p}}\beta)$

jacquard.nersc.gov

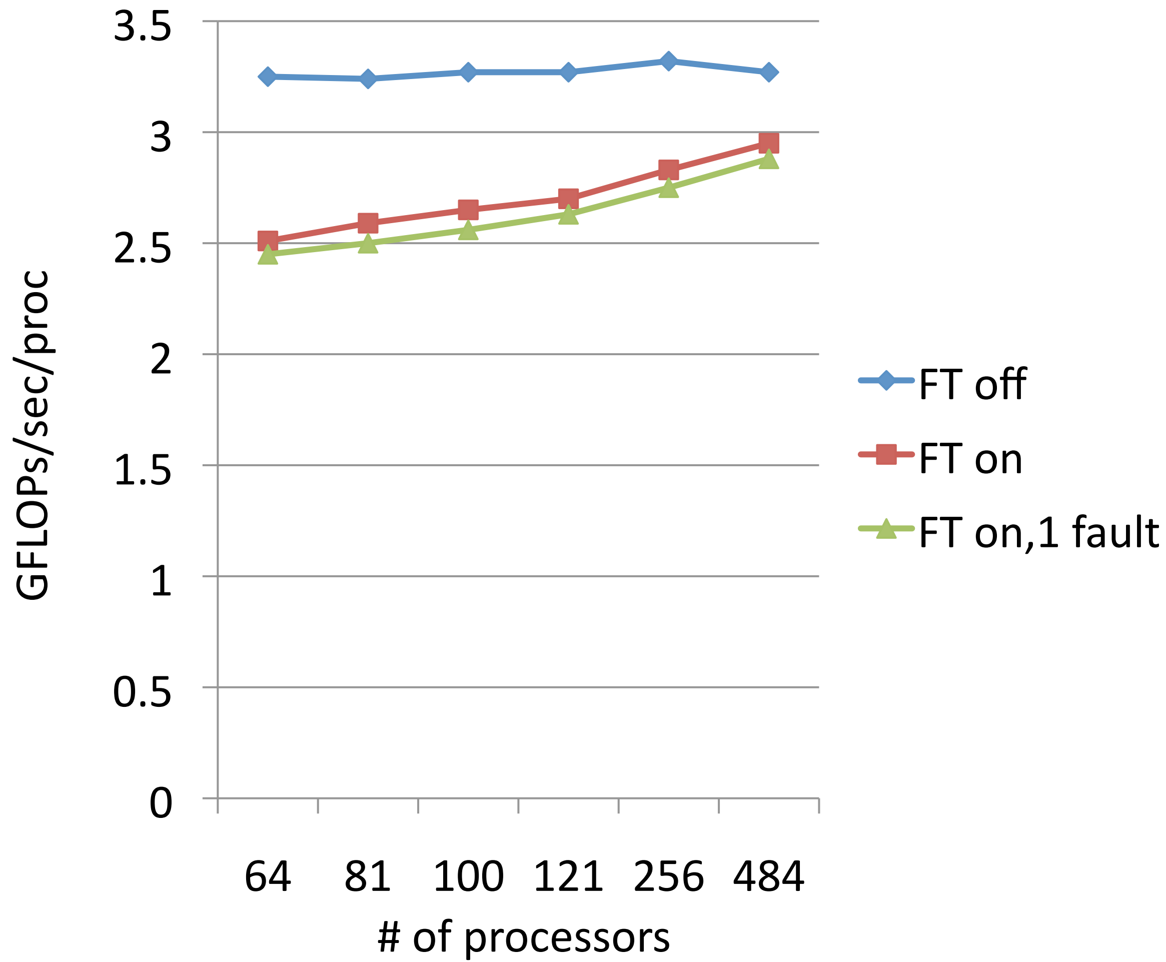
- **Processor type** Opteron 2.2 GHz
- **Processor theoretical peak** 4.4 GFlops/sec
- **Number of application processors** 712
- **System theoretical peak (computational nodes)** 3.13 TFlops/sec
- **Number of shared-memory application nodes** 356
- **Processors per node** 2
- **Physical memory per node** 6 GBytes
- **Usable memory per node** 3-5 GBytes
- **Switch Interconnect** InfiniBand
- **Switch MPI Unidrectional Latency** 4.5 µsec
- **Switch MPI Unidirectional Bandwidth (peak)** 620 MB/s
- **Global shared disk GPFS Usable disk space** 30 TBytes
- **Batch system** PBS Pro

Mvapich vs FTMPI

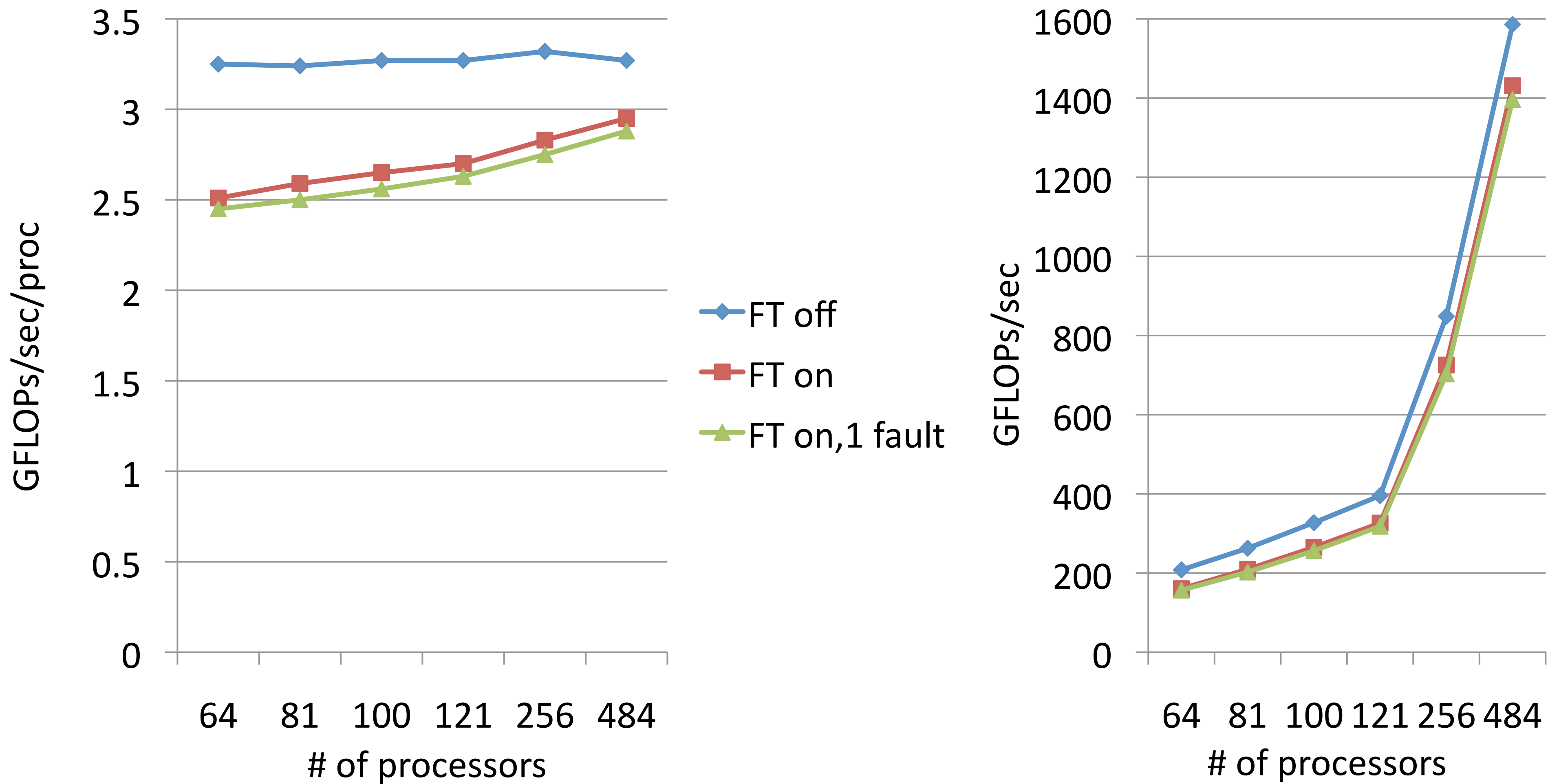




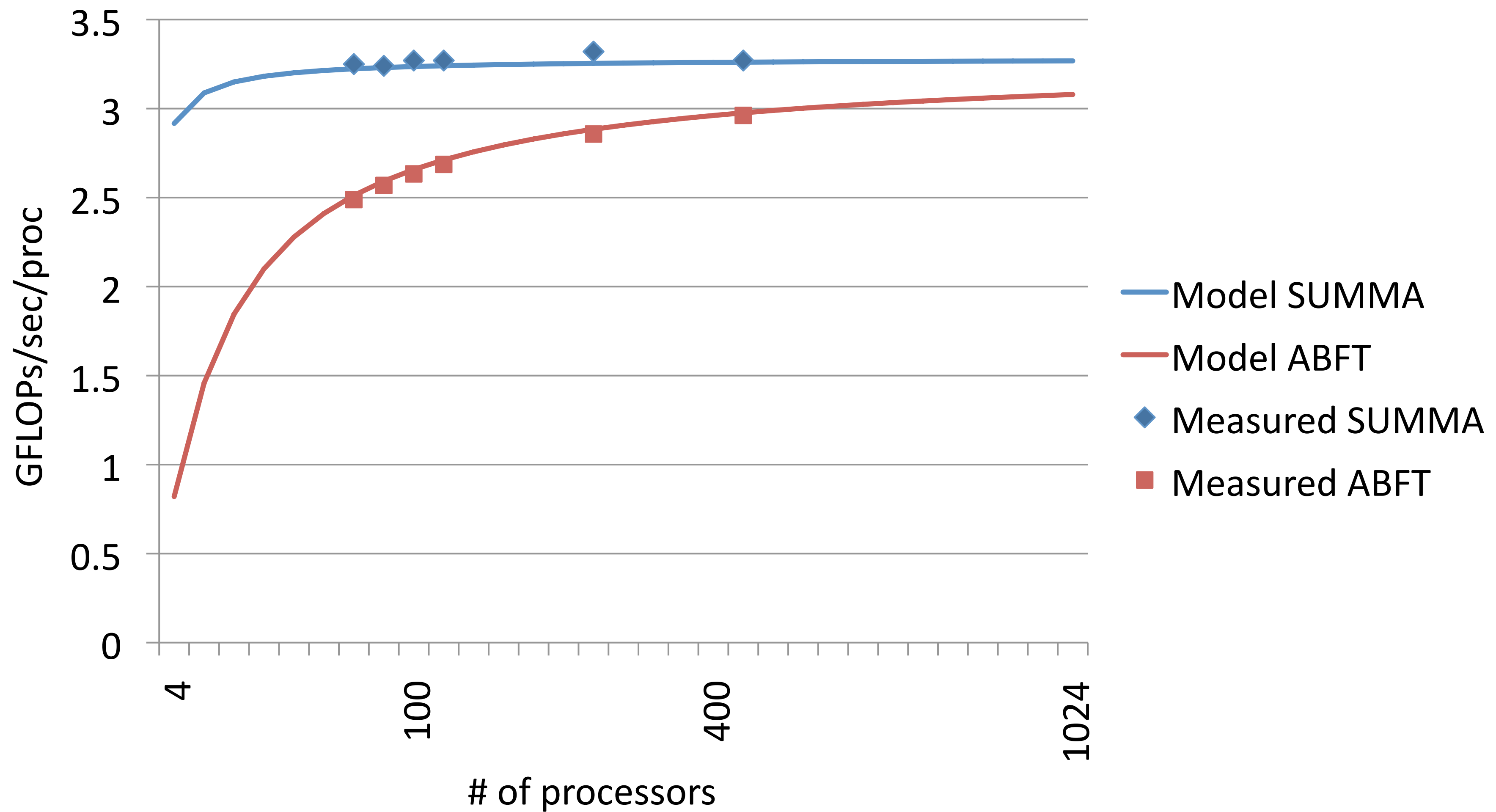
FT-PDGEMM -- nloc=4,000



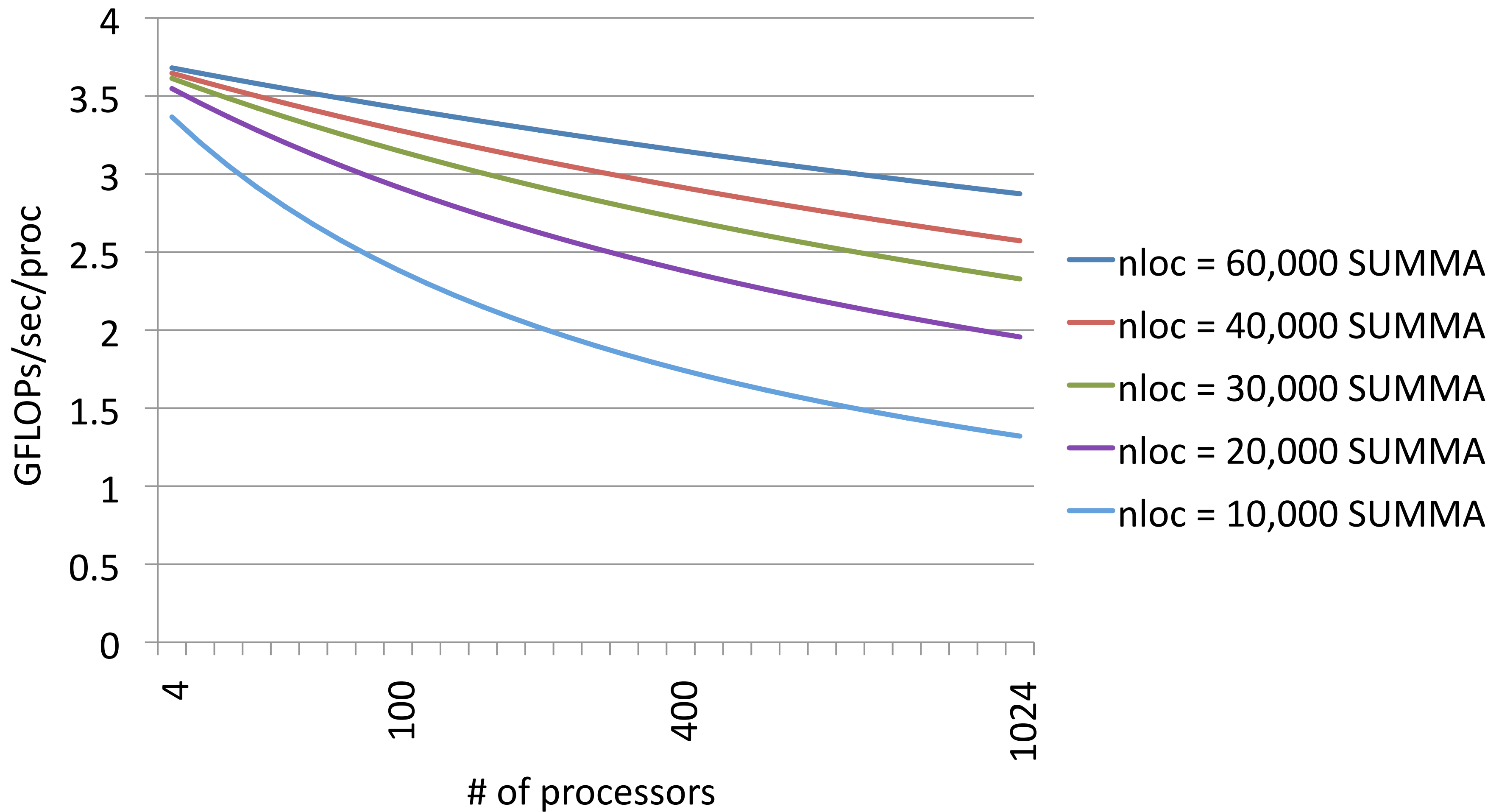
FT-PDGEMM -- nloc=4,000



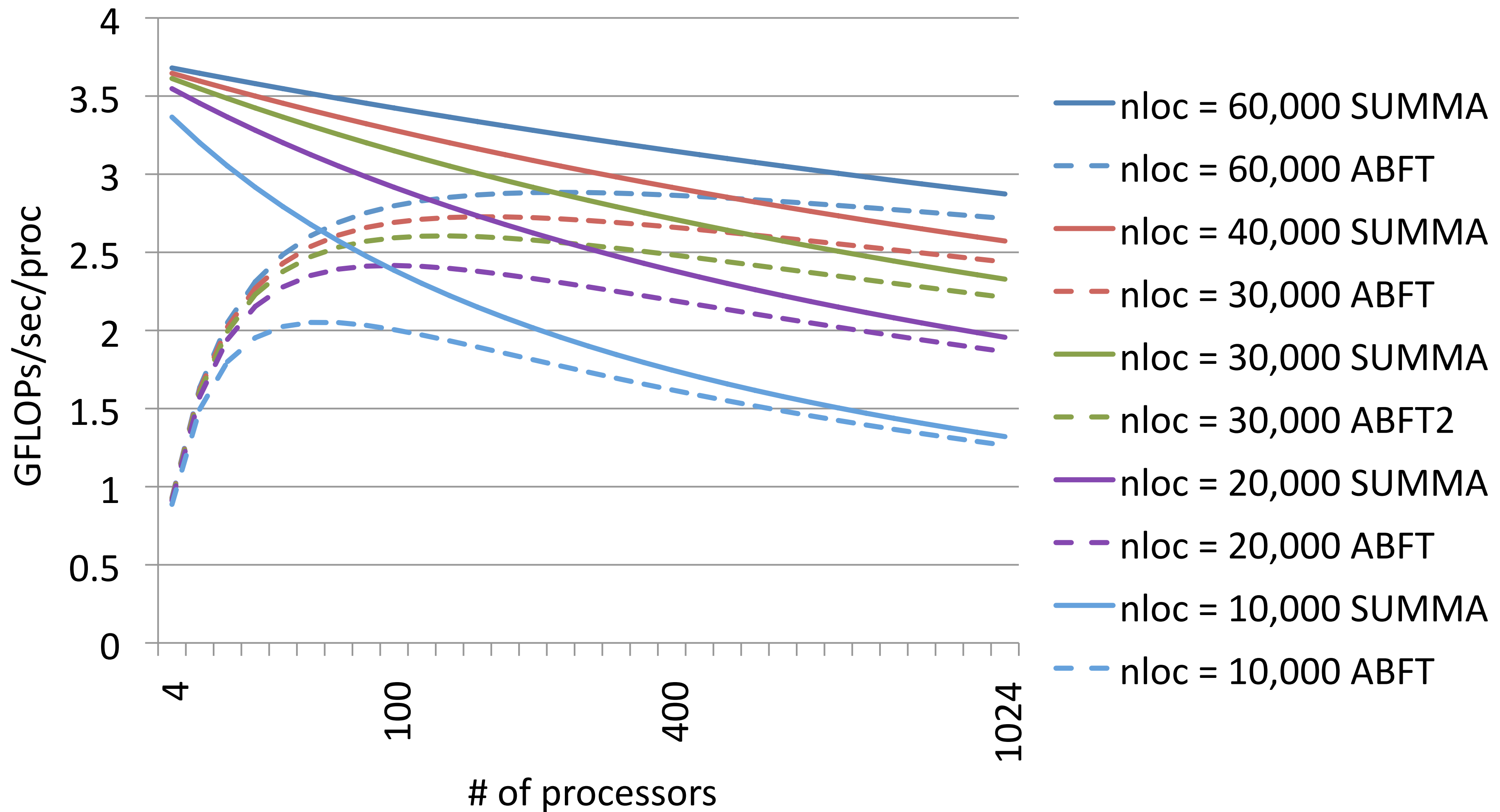
Performance modeling



Strong scalability

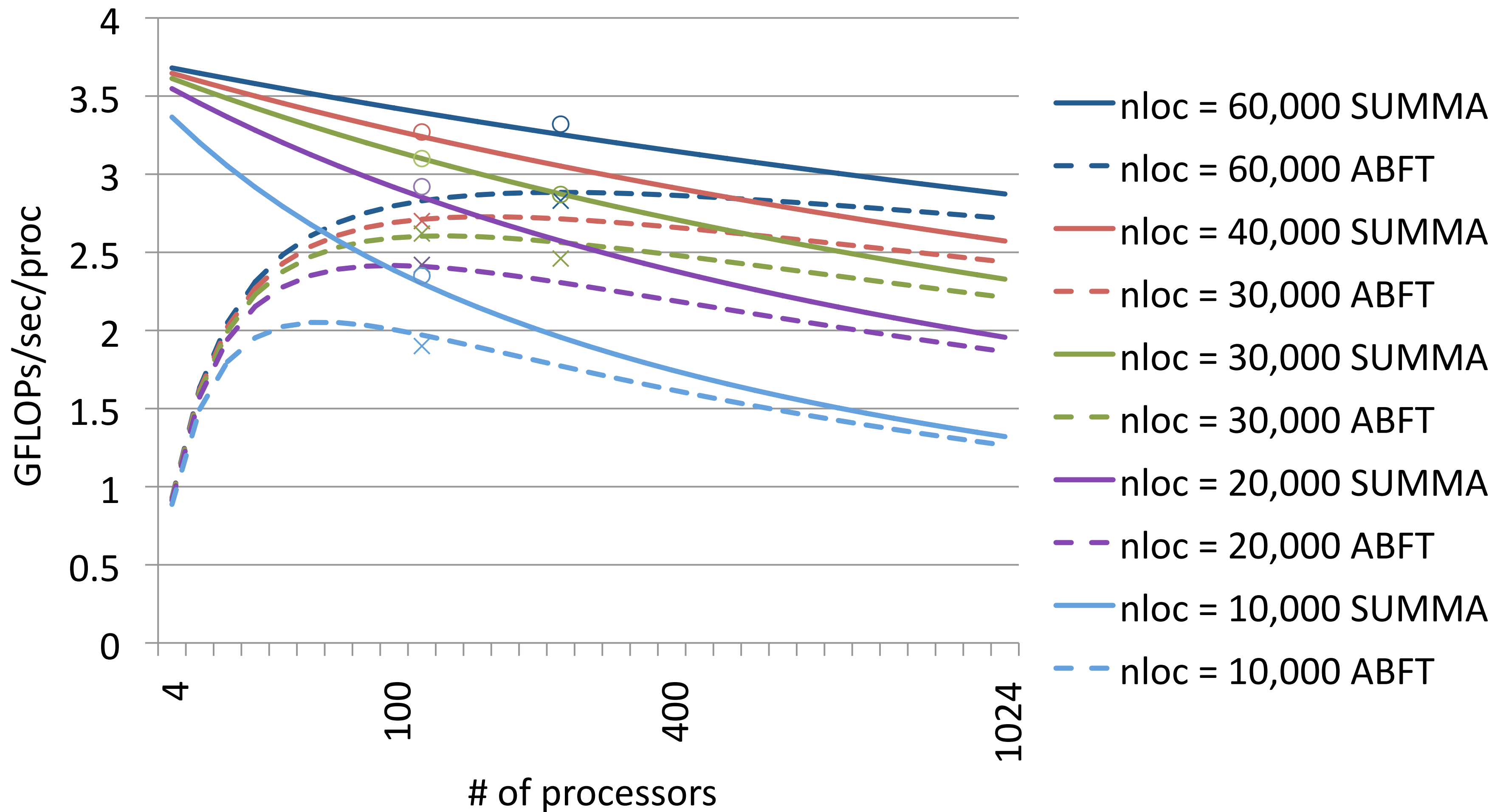


Strong scalability



ABFT represents the only known alternative to address fault tolerance in strong scalability

Strong scalability



ABFT represents the only known alternative to address fault tolerance in strong scalability

ABFT advantages over diskless checkpointing

- **Independent of the Surface (n^2) / volume (n^3).**
  - Important for n/n operations (e.g. FFT)
  - Important for MM with small n
- **Independent of failure rate**
  - No need to guess parameters
- **Fits nicely in the algorithm**
  - No need for explicit synchronization for example

ABFT disadvantage over diskless checkpointing

- **Rely on floating point arithmetic checksums**
  - Cancellation, ill-conditionned matrices, etc.