The Next MPI challenge(s)

George Bosilca

ICL UT INNOVATIVE COMPUTING LABORATORY

Has MPI really failed ?

- Difficult to define a success metric
  - Failure metric
    - How many MW were lost due to MPI?
  - Success metric
    - How many people get a job based on MPI skills?
- How much breakthrough science came to light due to MPI?

Thread based MPI

MPI Processes

- MPI is process based, threads are external entities outside of MPI knowledge
- Point-to-point communications between threads are possible by crafting special tags
- Collectives are process based, one process participate in the collective once
- Threads fight for messages instead of collaborating
- Different approach than TMPI and AMPI

MPI Threads

- What if:
  - MPI became threads based, i.e. each thread get a rank
  - Each thread is allowed to behave as a MPI process today
- We can use a thread based programming approach, mixed with message synchronization and collective communication
- Stay as close as possible to the current MPI standard

MPI Threads

- What if:
- MPI became threads based, i.e. each thread gets a rank
- Ea[...] pro[...]
- We[...] ap[...]zation an[...]
- Stay as close as possible to the current MPI standard

> - Stay as close as possible to the current MPI standard (Nx1 is a standard MPI application)
> - MPI_COMM_WORLD is still the same

MPI_Init_thread

- mpiexec -np NxM ...
  - will start N processes and notify them that each will have at most M threads
- Extend the standard with MPI_COMM_LOCAL including all M local threads
  - Each thread is required to call MPI_Init_thread to set its rank in the MPI_COMM_LOCAL

MPI ranks

- MPI_COMM_LOCAL is a fully featured intra-communicator
  - process based communicator vs. thread based communicator
- It can be used by any communicator creation function
- If any doubts about the rank of the thread in a communicator creation, the order will be based on the rank in the local communicator.

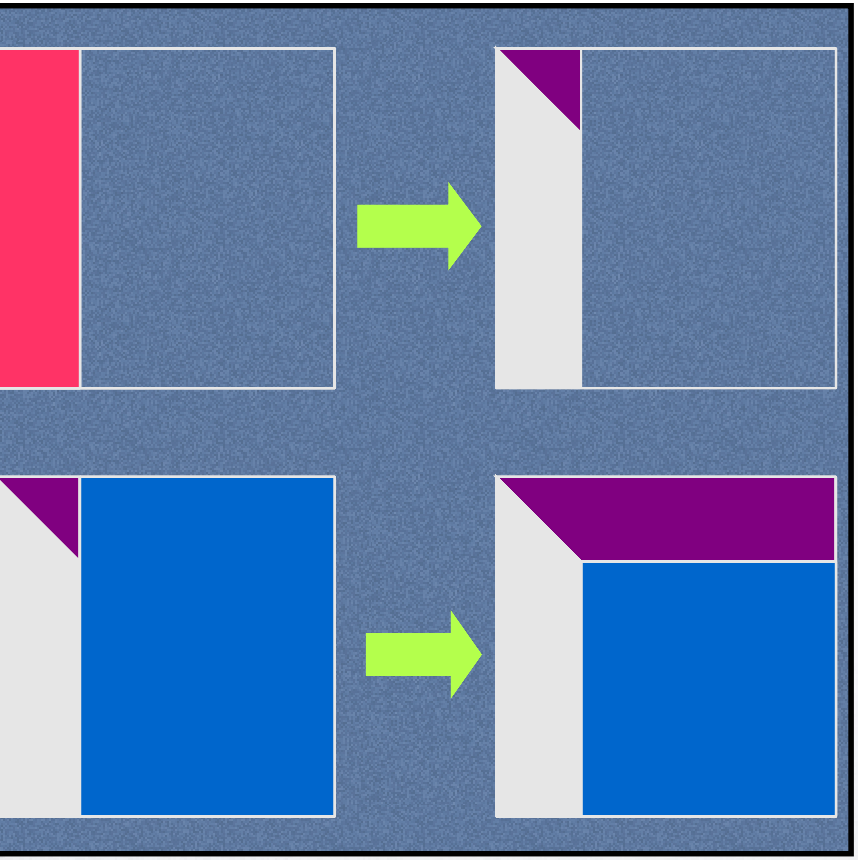
Receive Rules

- On the process based communicator su
as MPI_COMM_WORLD all threads ca
match a receive
- On all mixed communicators the receive
are named by rank (thread)
- Similar rules applies for collective
communications, i.e. a process can
participate multiple times in a collective.

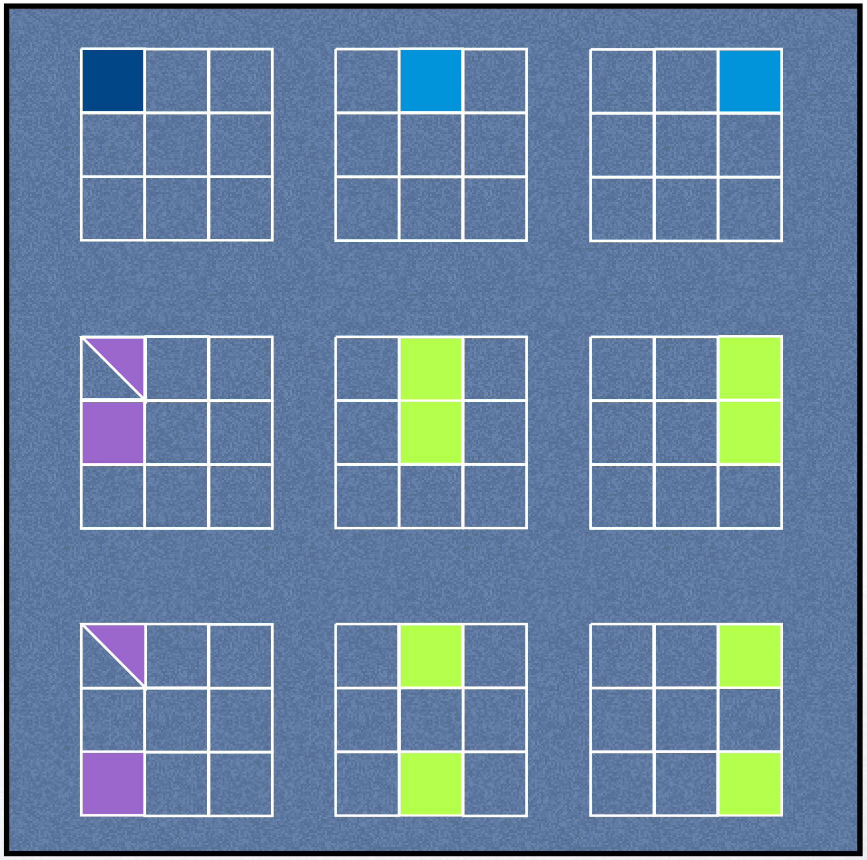
PLASMA

PLASMA: Tile Algorithms

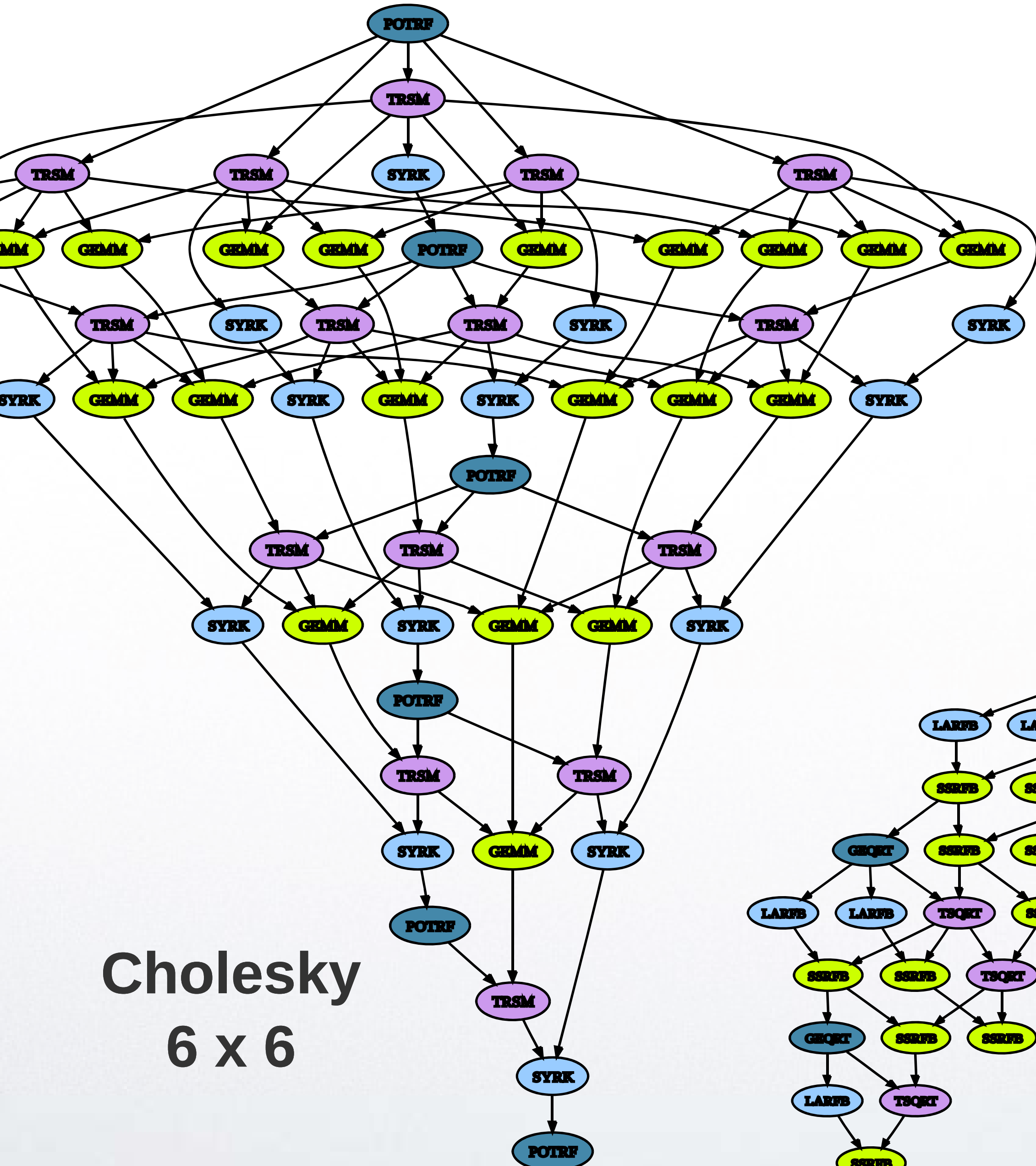
Block algorithms – LAPACK



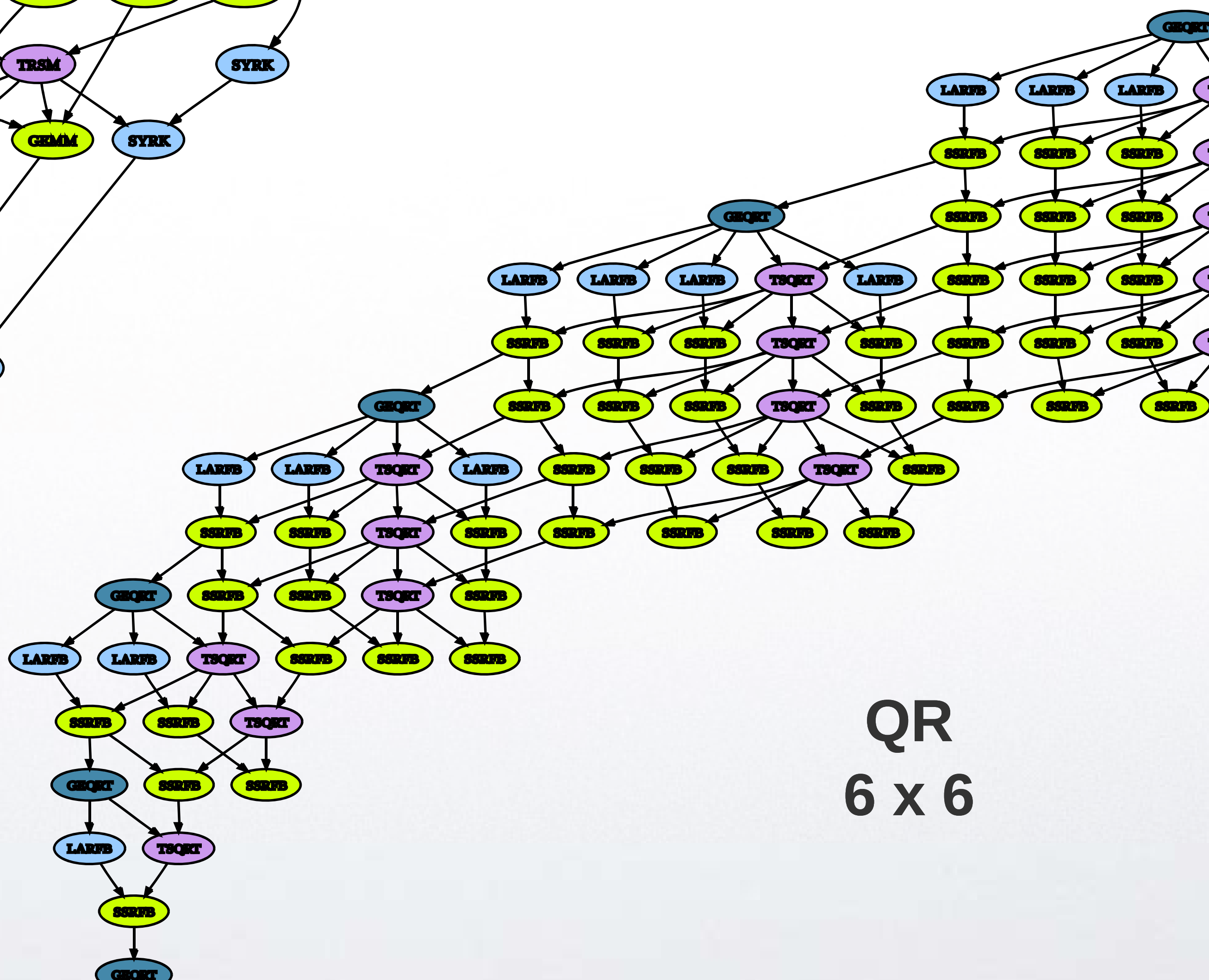
Tile algorithms – PLASMA



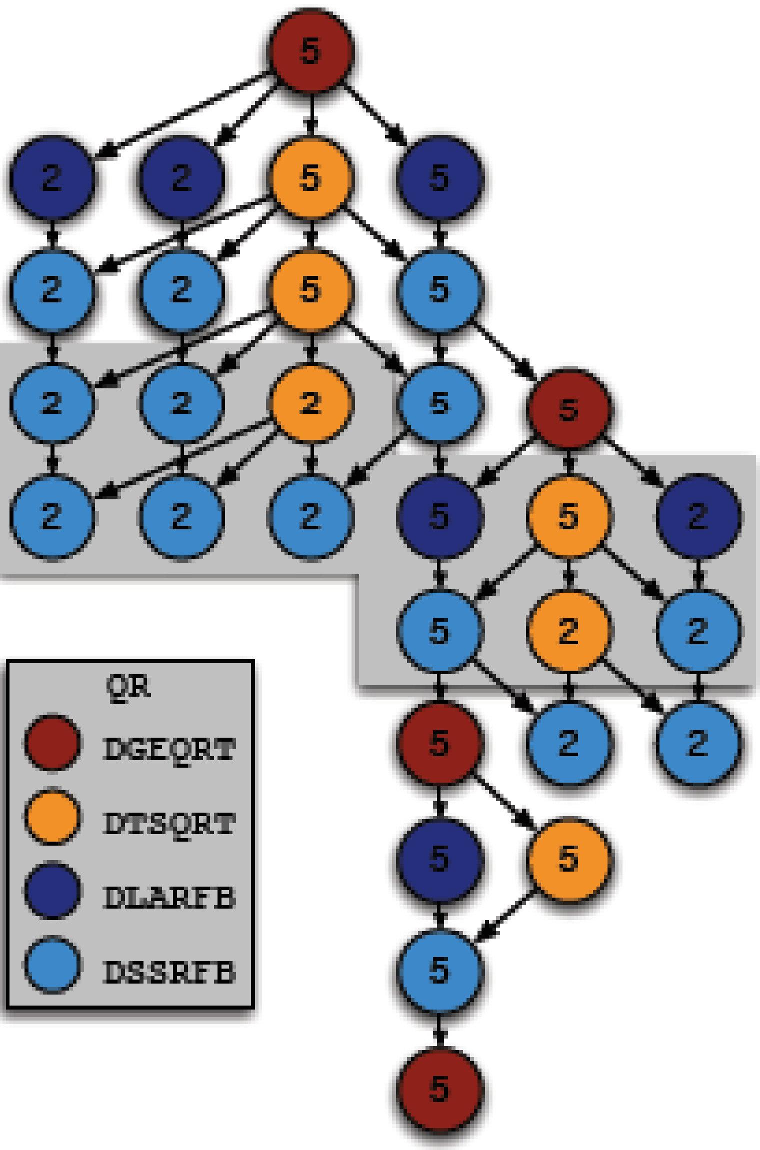
PLASMA: DAG Scheduling



Cholesky
6 x 6



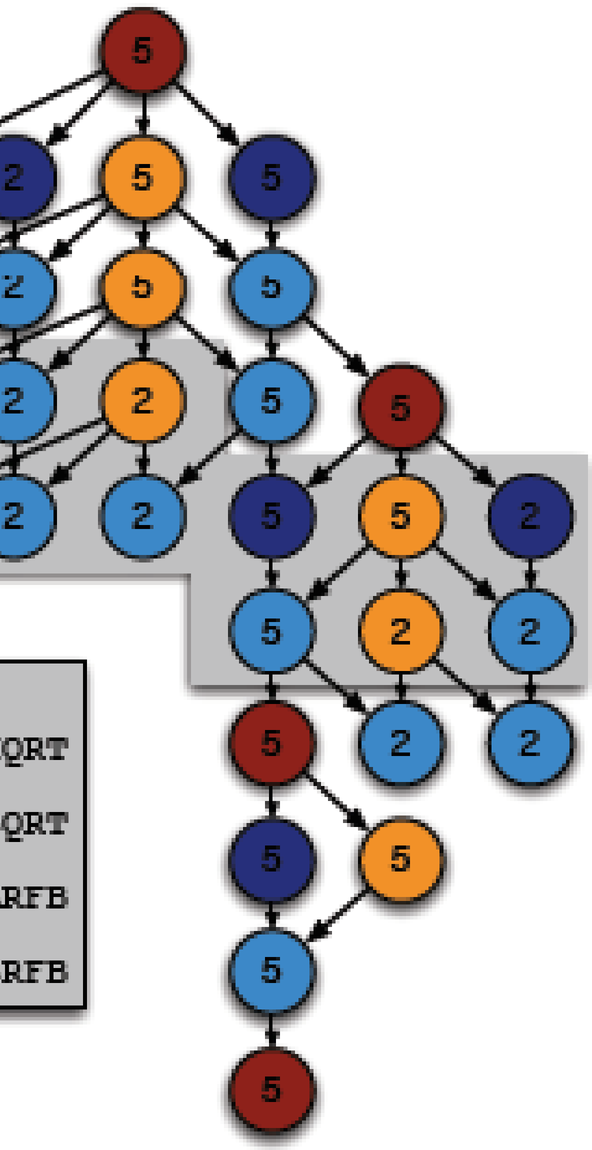
QR
6 x 6



Tiles for QR Factorization

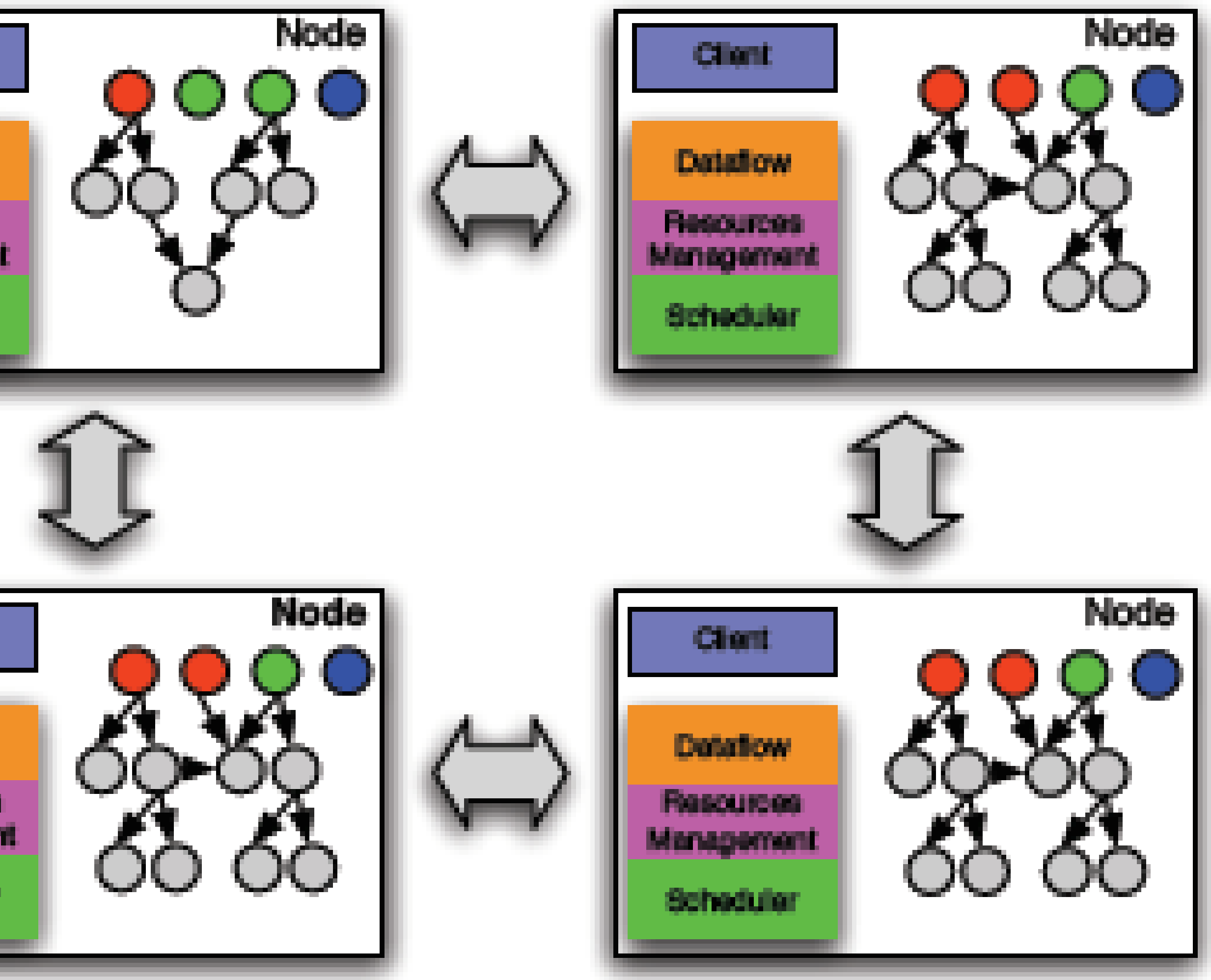
- acyclic representation of the algorithm as a directed graph with procedures attached to the nodes
- nodes are annotated with the list of input and output parameters
- special node for conditionals, loops and collective

Challenges



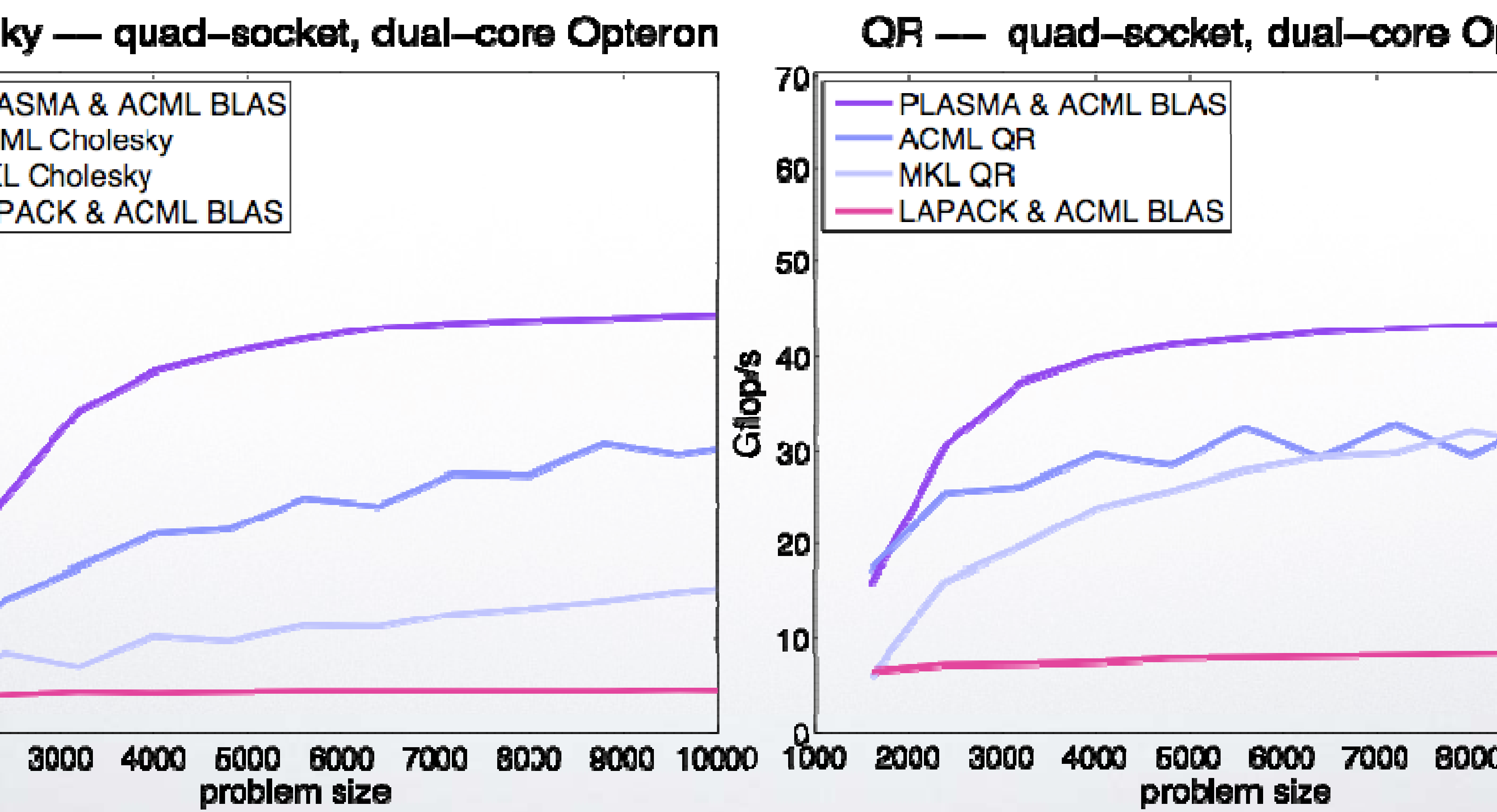
- DAG construction and exploration
    - initial approach: static partitioning and dynamic scheduling in each sub-domain
    - "sliding window" approach
- Dynamic scheduling: trade between data reuse and aggressive pursuit of the critica[l] path

SPMD/MPMD



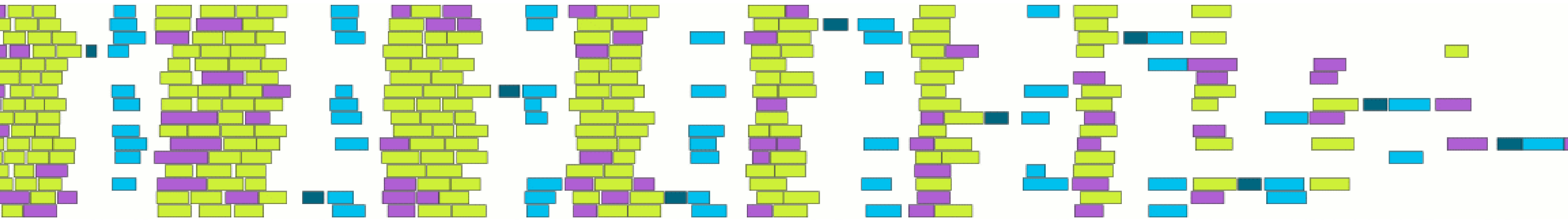
- Some dependenci
will point to local
variables, while ot
point to external d
- Communications a
implicit, and the
scheduler can extr
them from the DA
- Potential for
overlapping
communications a
computations

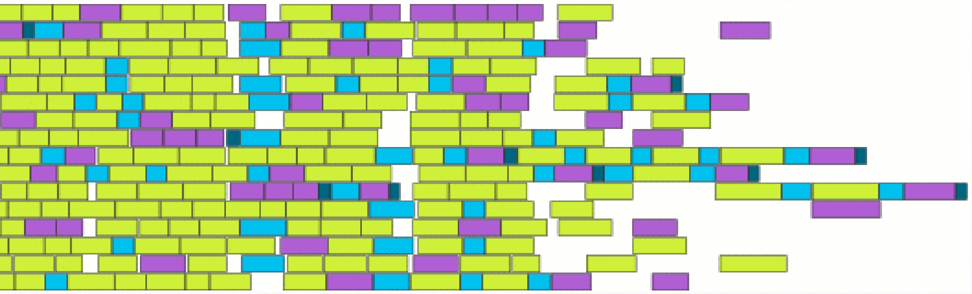
Early results



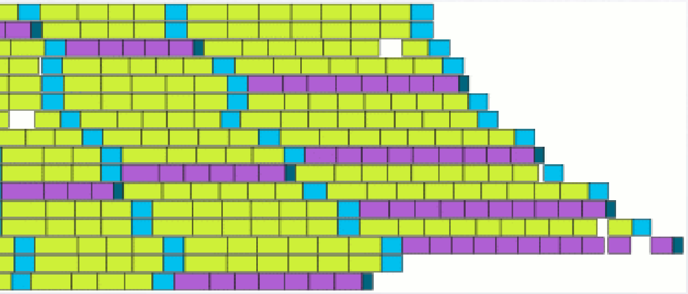
G Scheduling: Cholesky

TBB: nested parallelism





SMPSs:
arbitrary DAG,
dynamic scheduling,
data renaming



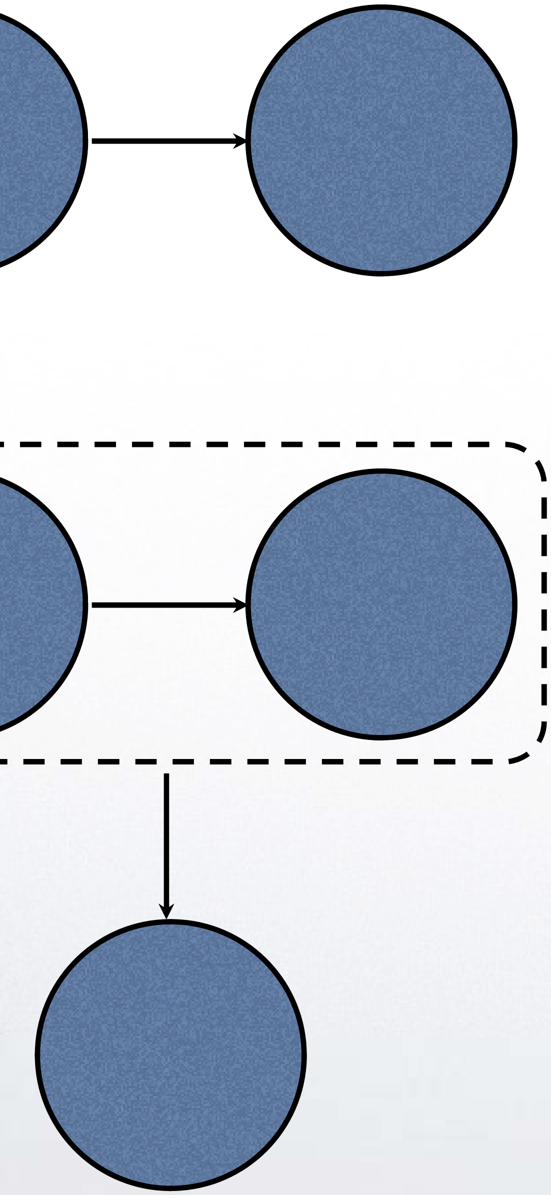
Current PLASMA scheduler

The runtime system

- Resource constraints
- Automatic Resource Management
- Asynchronous Task Executions
- Implicit communications
- Collective Communications
- Dynamic multi-level scheduling
- Fault Tolerance

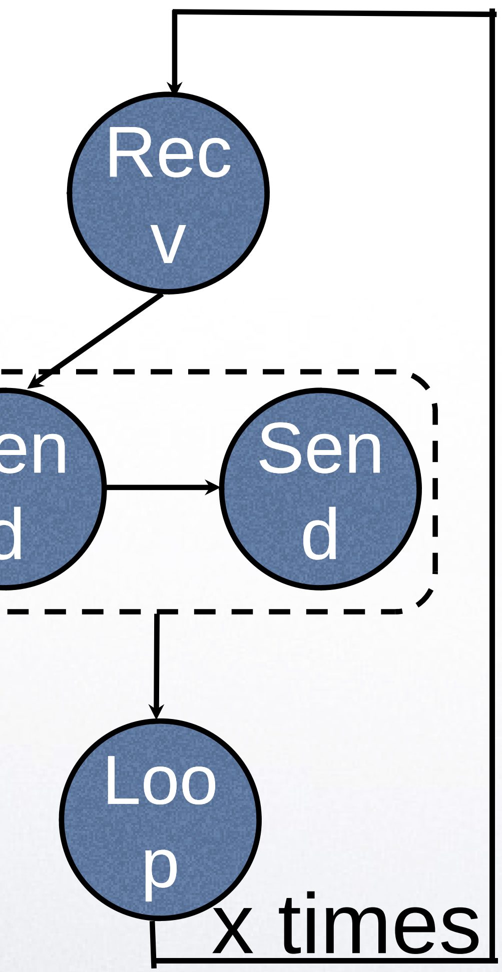
CCI

Low level DAG device



- Tasks: send, receive, op
- Horizontal arrow: concurrent execution
- Vertical arrow: sequential execution
- Dash line: multi dependencies

Pipelined Binary Reduce



- Created at the user level
- Executed by the lowest level
  - Small overhead
  - No interruptions
  - Asynchronous
  - Report on completion

FT-MPI

Why ?

- A lack of fault tolerant programming paradigms
- MPI is the de-facto programming model for parallel applications
- MPI Standard: "*Advice to implementors*: A good quality implementation will, to the greatest possible extent, circumvent the impact of an error, so that normal processing can continue after an error handler was invoked."

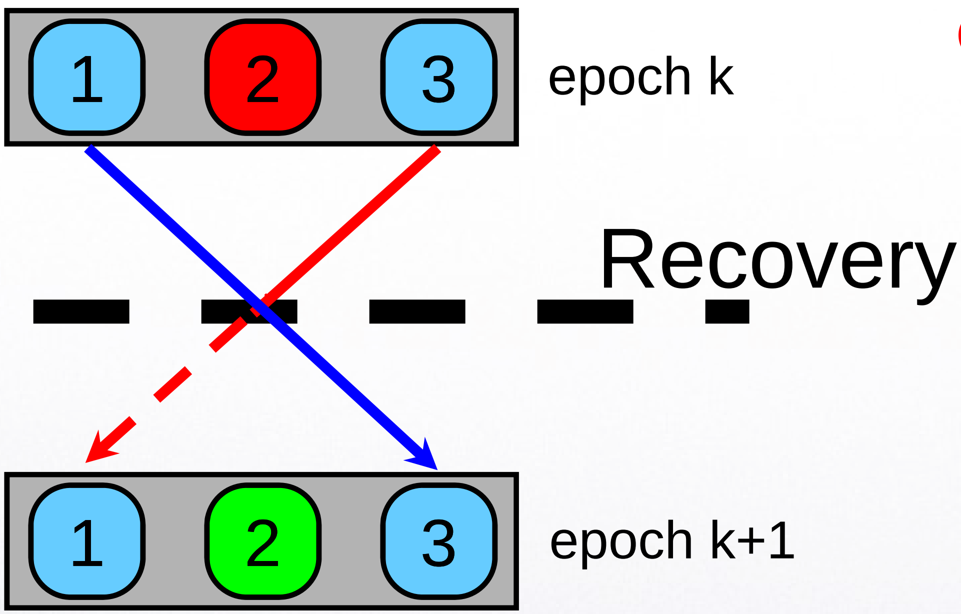
How ?

- Define the behavior of MPI [state] in case an error occurs
- Give the application the possibility to recover from a node-failure
- A regular, non fault-tolerant MPI program will run using FT-MPI
- Follows the MPI-1 and MPI-2 specification as closely as possible (e.g. no additional function calls)
- On error user program must do something (!)

Recovery modes

- ABORT, BLANK, SHRINK and REBUILD
- REBUILD: a new process is created, and it will return MPI_INIT_RESTARTED_PROC from MPI_Init
- BLANK: dead processes replaced by MPI_PROC_NULL, all communications with such a process succeed, they do not participate in the collectives
  - two sub-modes: local and global

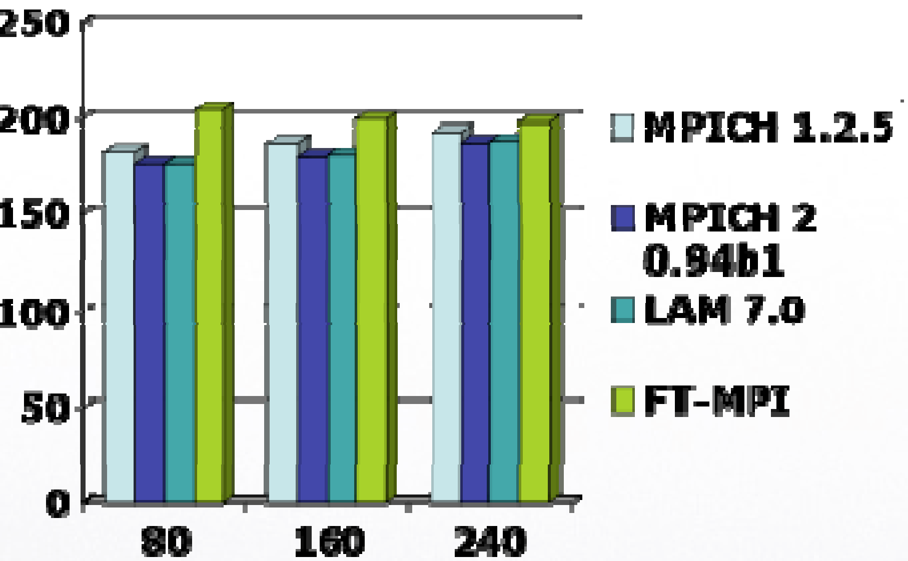
Communications modes



- RESET: the epoch should match in addi
to the MPI matching
requirements
- CONTINUE: only MP
matching

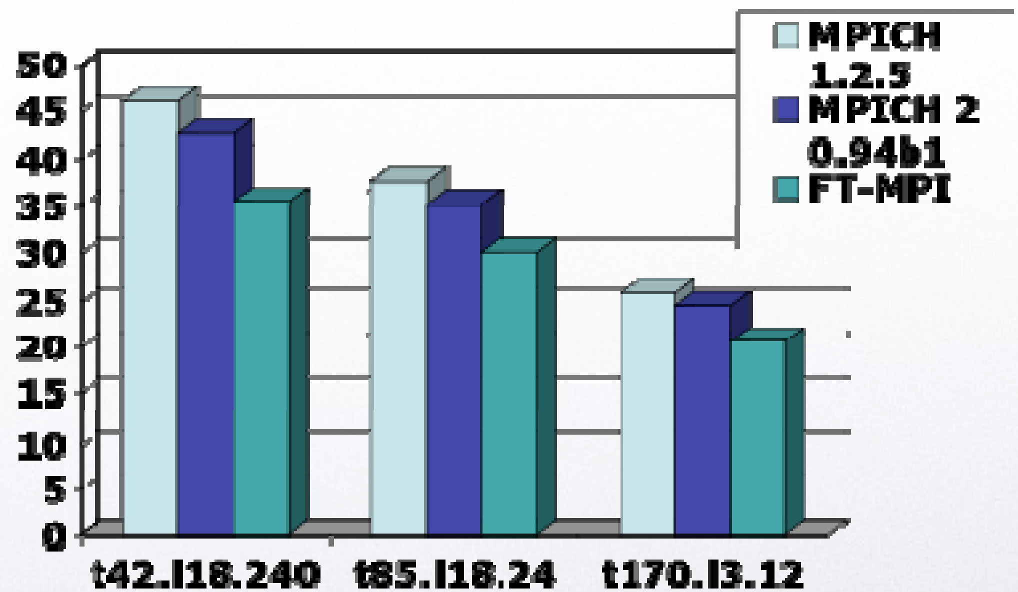
Shallow Water (PSTSWM) & HPL

32 nodes with Gigabit

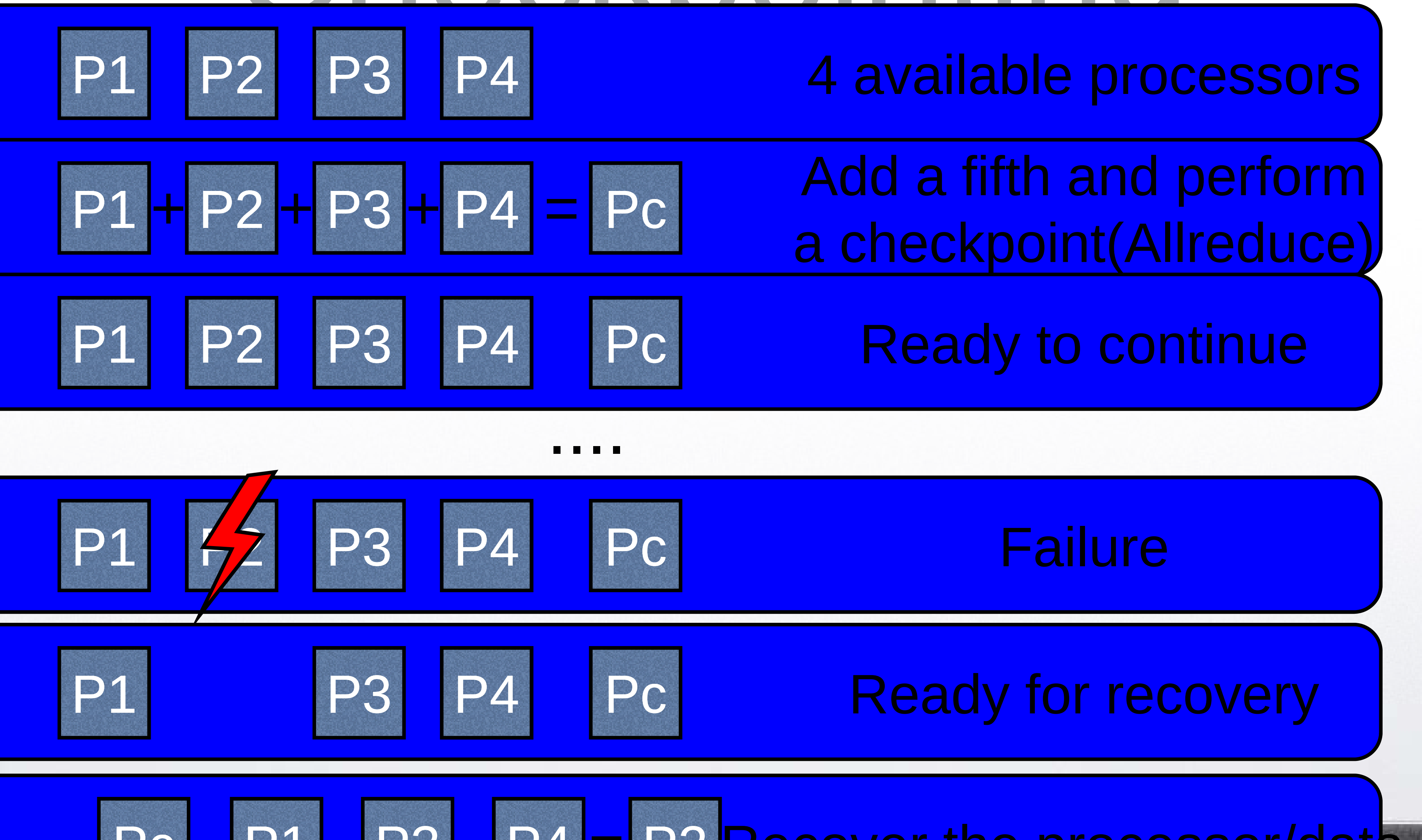


HPL

PSTSWM



Diskless Checkpointing



Diskless Checkpointing

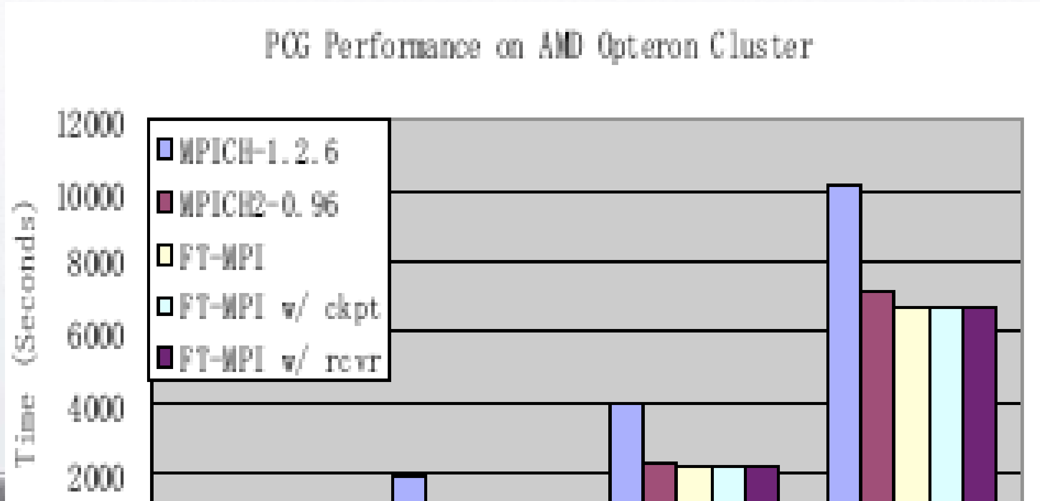
- How to checkpoint ?
  - either floating-point arithmetic or binary arithmetic will work
  - If checkpoints are performed in floating-point arithmetic then we can exploit the linearity of the mathematical relations on the object to maintain the checksums
- How to support multiple failures ?
  - Reed-Salomon algorithm
  - support p failures require p additional processors (resources)

PCG

- Fault Tolerant CG
- 64x2 AMD 64 connected using GigE

	Size of the Problem	Num. of Comp. Procs
Prob #1	164,610	15
Prob #2	329,220	30
Prob #3	658,440	60
Prob #4	1,316,880	120

Performance of PCG with different MPI librar

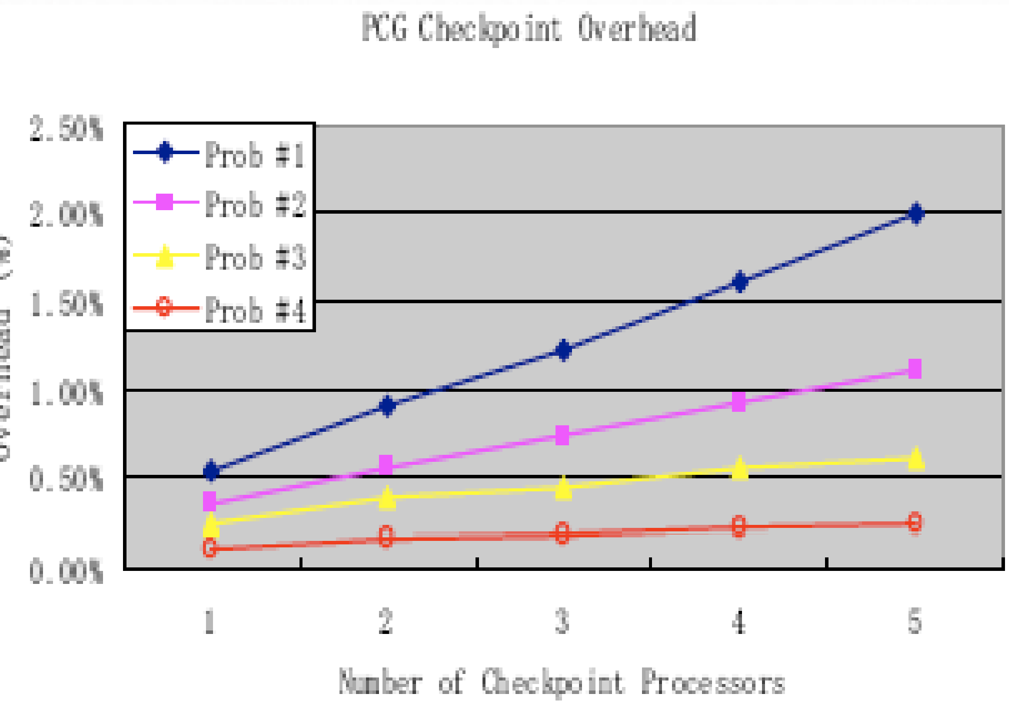


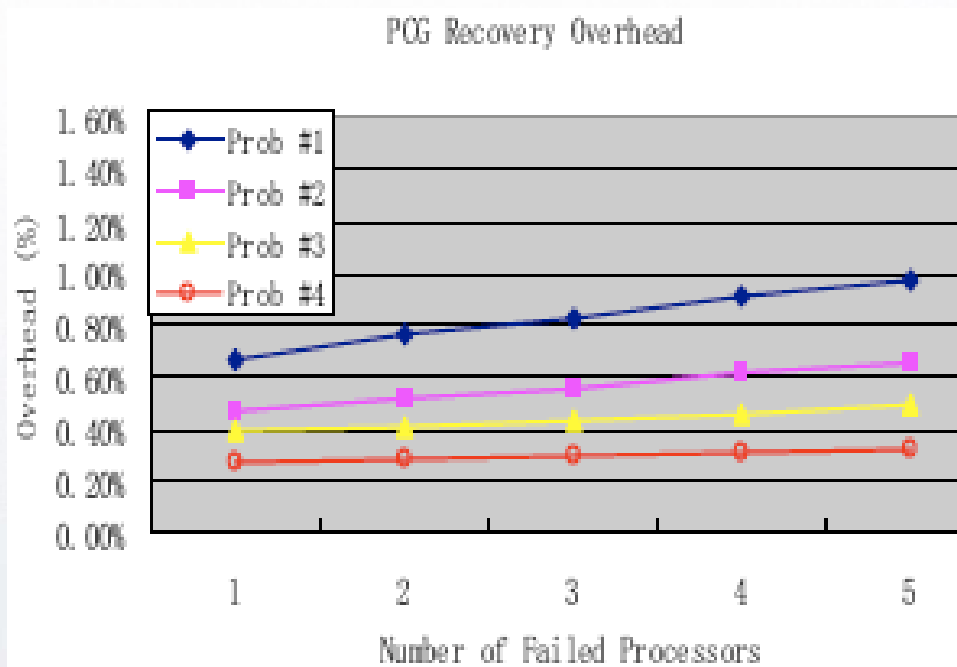
For cl
generate
every
itera

PCG

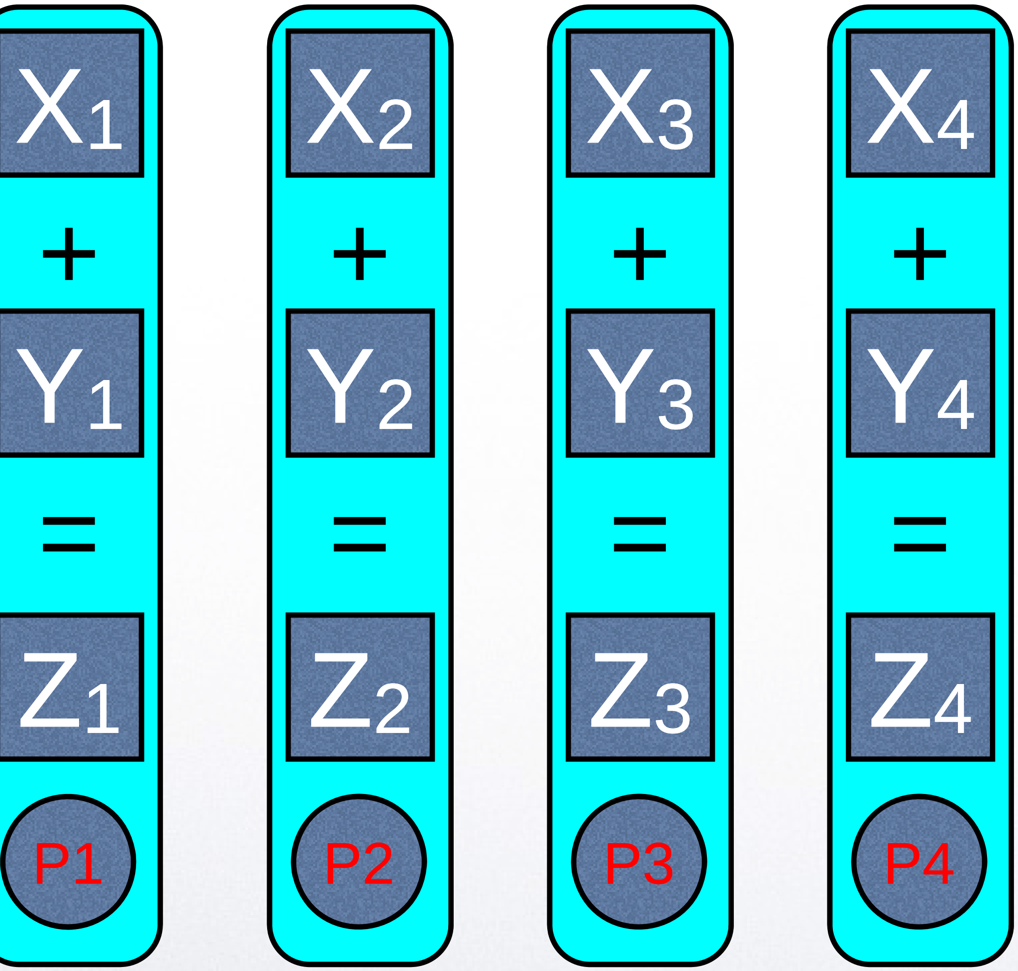
Time	Prob #1	Prob #2	Prob #3	Prob #4
1 ckpt	2.6	3.8	5.5	7.8
2 ckpt	4.4	5.8	8.5	10.6
3 ckpt	6.0	7.9	10.2	12.8
4 ckpt	7.9	9.9	12.6	15.0
5 ckpt	9.8	11.9	14.1	16.8

Checkpoint overhead in seconds





AFTB concept in example

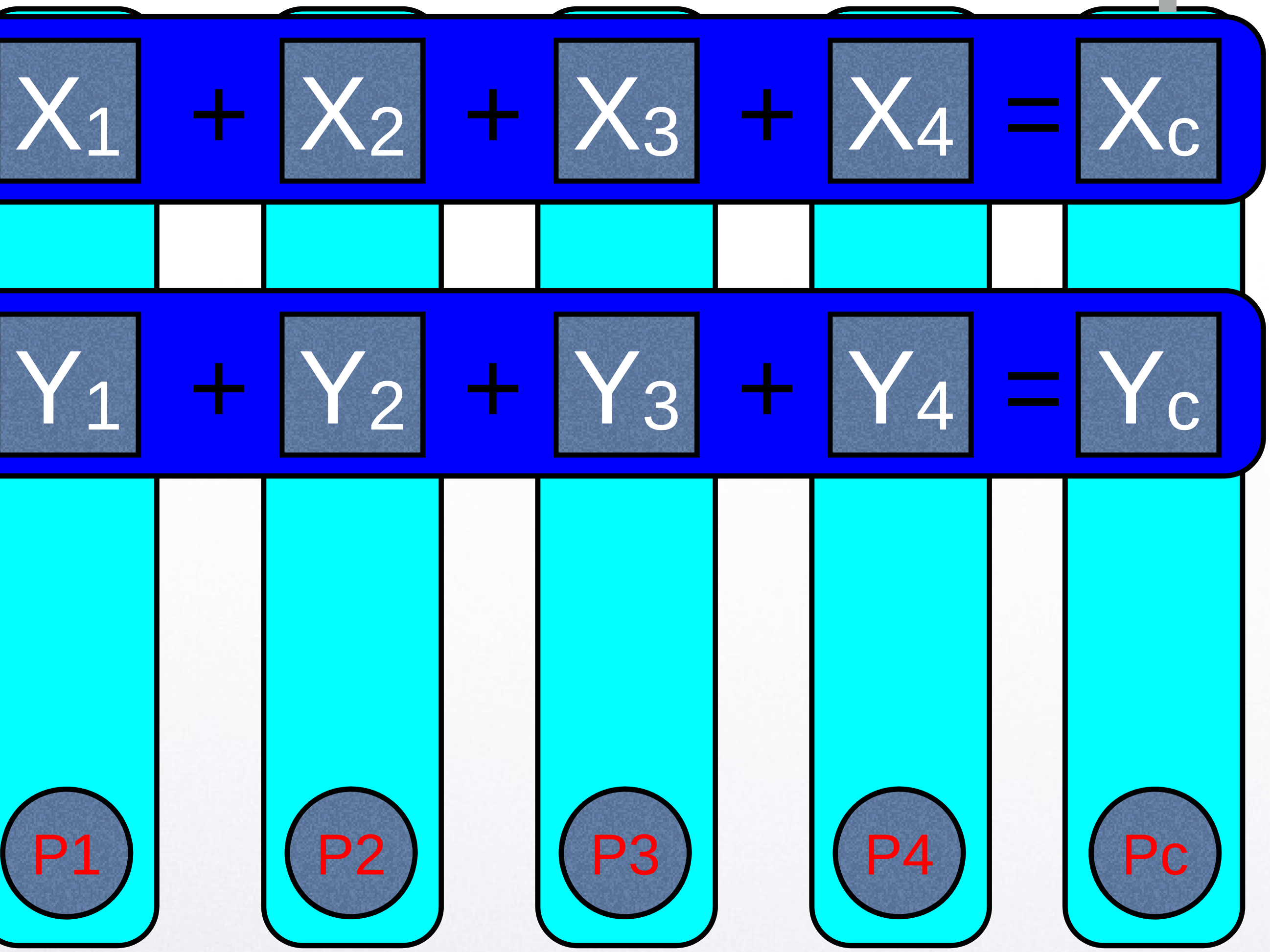


Perform in paral

$$z = x + y$$

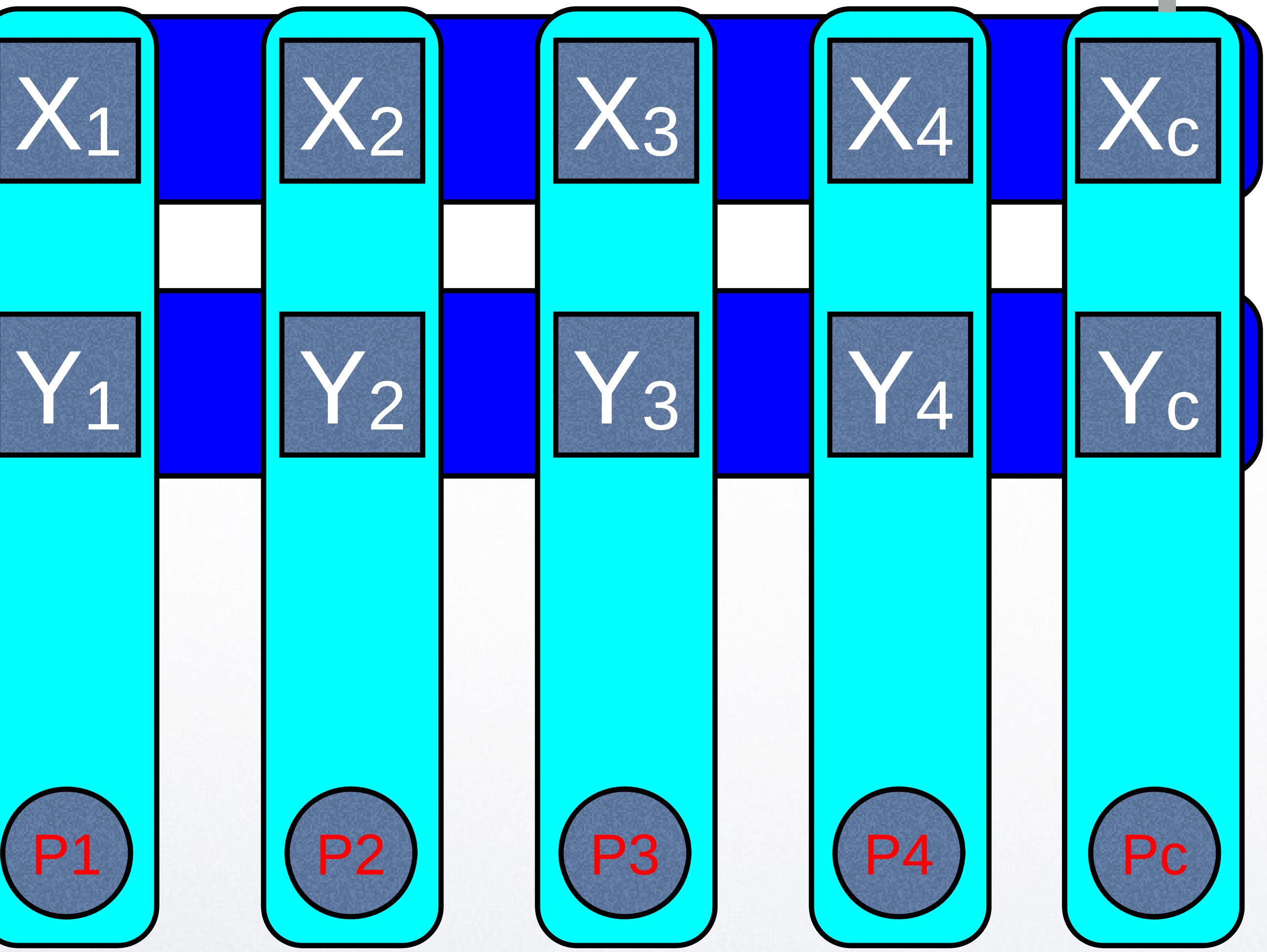
K. Huang, J. Abraham, "Algorithm-Based Fault Tole
Matrix Operations," IEEE Trans. on Comp. (Spec. I

AFTB concept in example



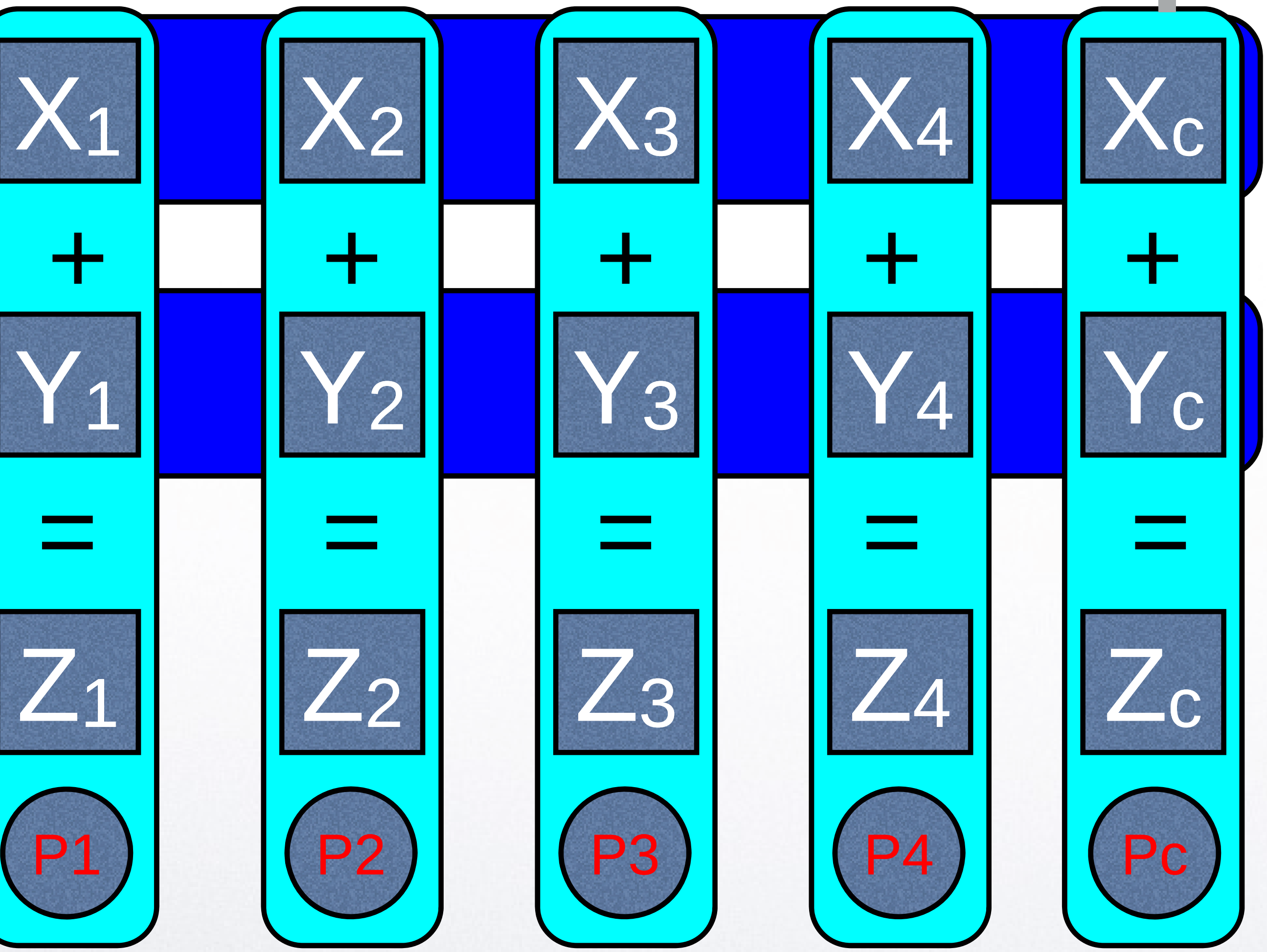
Compute in paral
the checksum o
x and y

AFTB concept in example



We're ready to proceed with the sum

AFTB concept in example



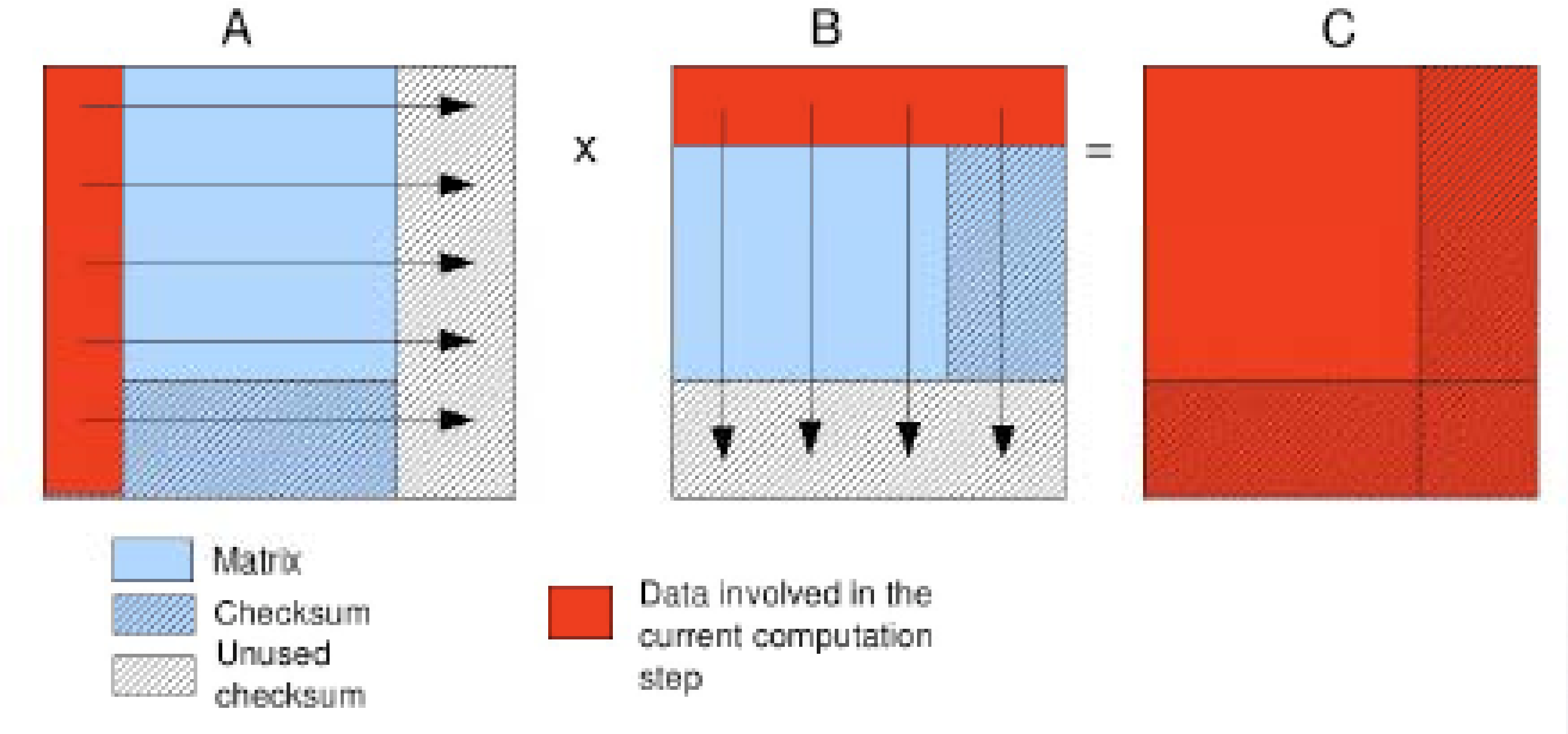
Compute in paral
the sum of
x and y

Simultaneously
can compute the
of the checkpoi

ABFT summary

- Relies on floating-point arithmetic
- Exploit the checksum processor
- Stable algorithms exist for any linear operation:
  - AXPY, SCAL (BLAS1)
  - GEMV (BLAS2)
  - GEMM (BLAS3)
  - LU, QR, Cholesky (LAPACK)
  - FFT

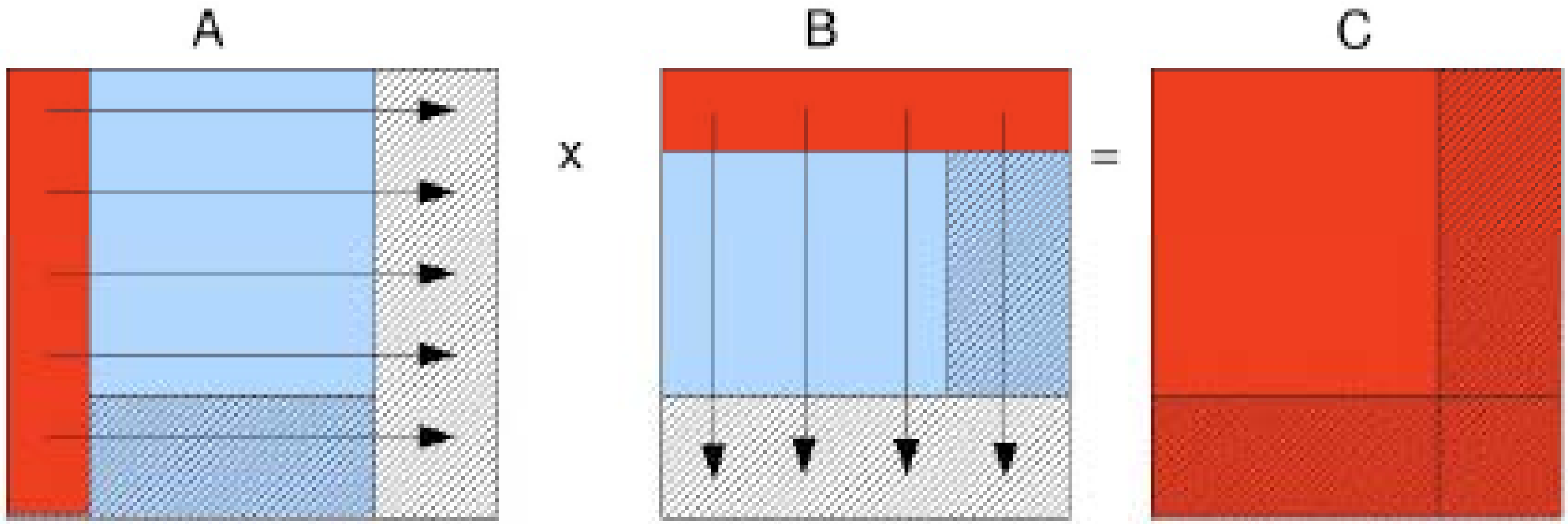
ABFT-PDGEMM



$$A_F = \begin{pmatrix} A & AC_R \\ C_C^T A & C_C^T A C_R \end{pmatrix} \quad \text{and} \quad B_F = \begin{pmatrix} B & BC_R \\ C_C^T B & C_C^T B C_R \end{pmatrix}$$

$$\begin{pmatrix} A \\ C_C^T A \end{pmatrix} \begin{pmatrix} B & BC_R \end{pmatrix} = \begin{pmatrix} AB & ABC_R \\ C_C^T AB & C_C^T ABC_R \end{pmatrix} = (AB)_F$$

ABFT-PDGEMM

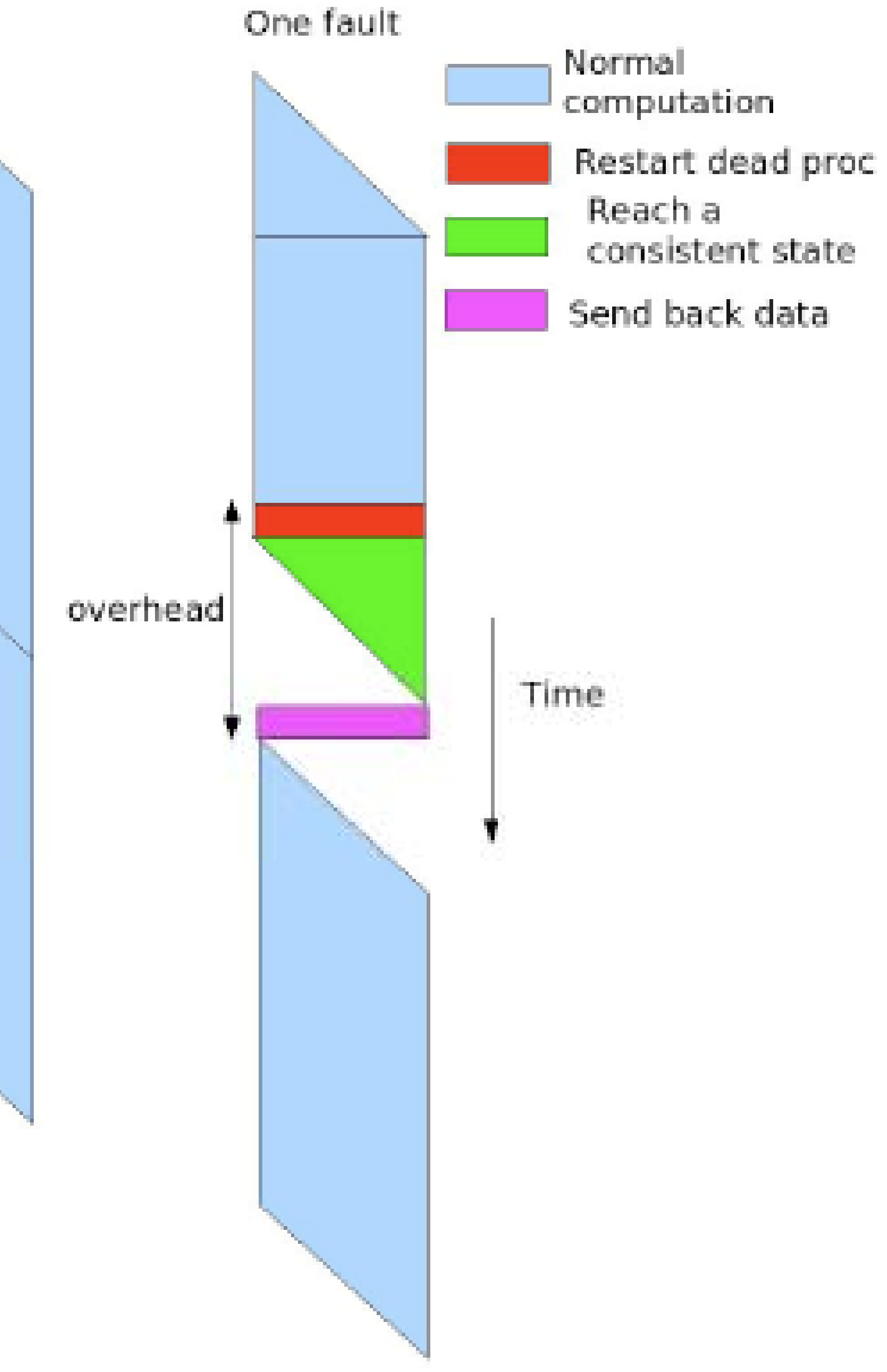


The overhead:

- 2p-1 extra processes for p2
- one extra process need to receive the data for the rows and columns

Conclusion: a very scalable approach, more processors means less overhead

Failure Overhead

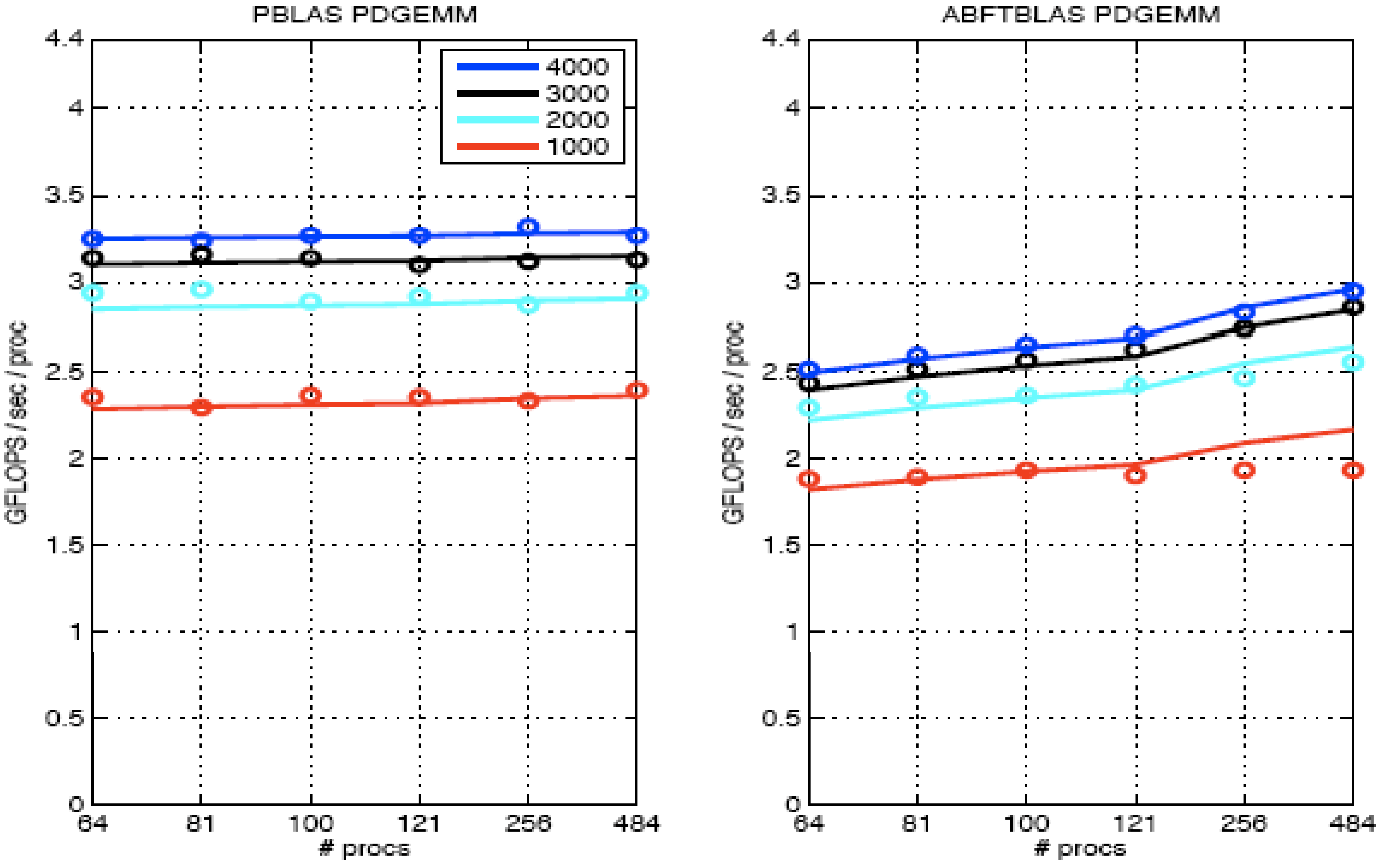


- FT-MPI will take care fault management
- Once the new process the MPI_COMM_WOR we have to rebuild the communicators
- Then we have to retrie the data from the checkpoint processor

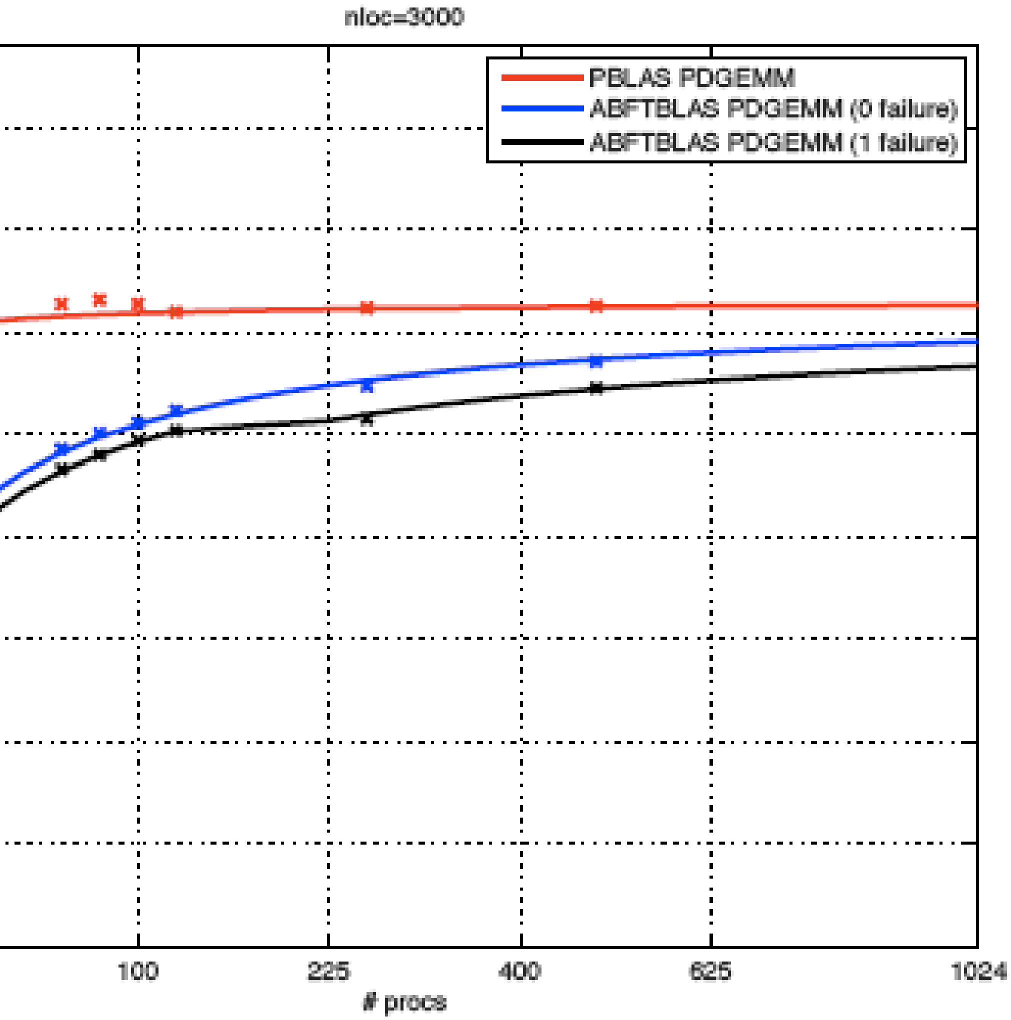
jacquard.nersc.gov

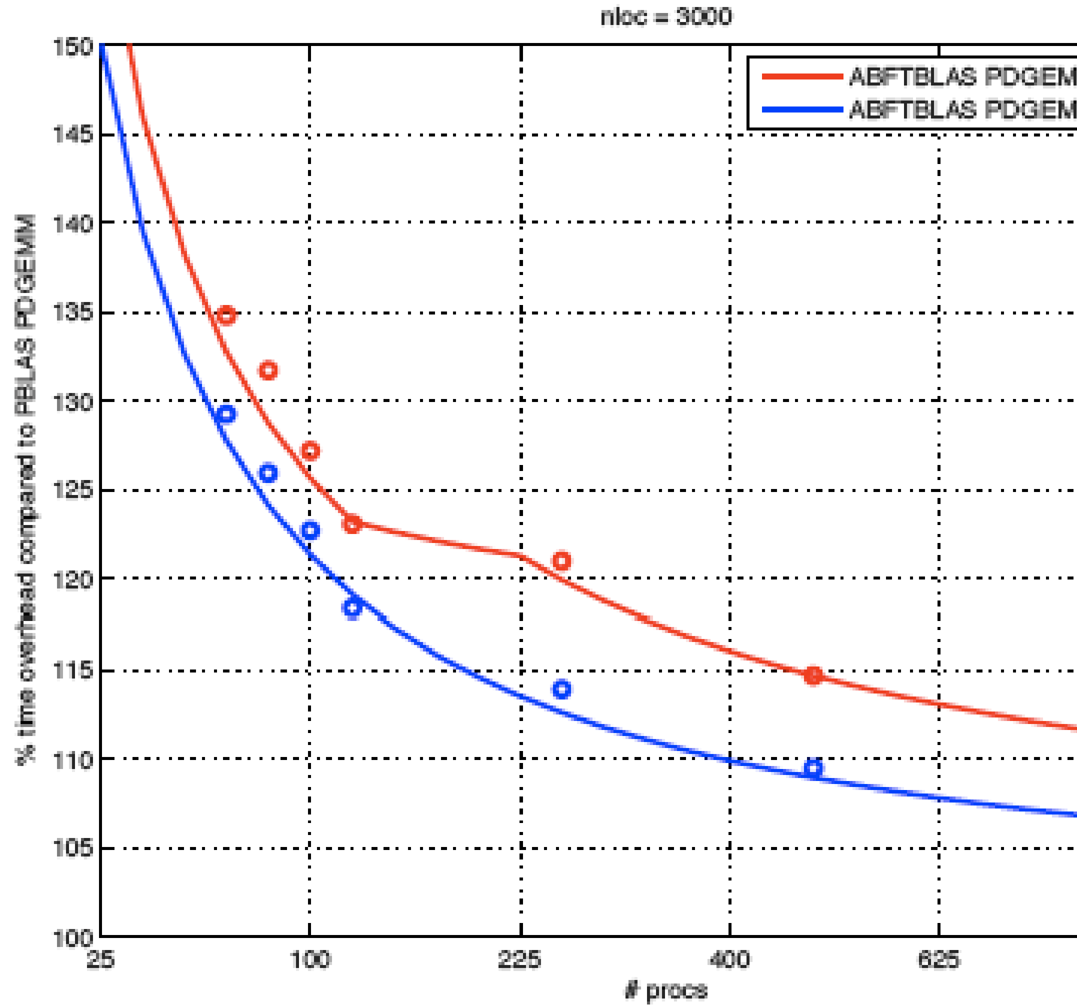
- **Processor type** Opteron 2.2 GHz
- **Processor theoretical peak** 4.4 GFlops/sec
- **Number of application processors** 712
- **System theoretical peak (computational nodes)** 3.13 TFlops/sec
- **Number of shared-memory application nodes** 356
- **Processors per node** 2
- **Physical memory per node** 6 GBytes
- **Usable memory per node** 3-5 GBytes
- **Switch Interconnect** InfiniBand
- **Switch MPI Unidirectional 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

PBLAS vs. ABFT BLAS (0 failure)

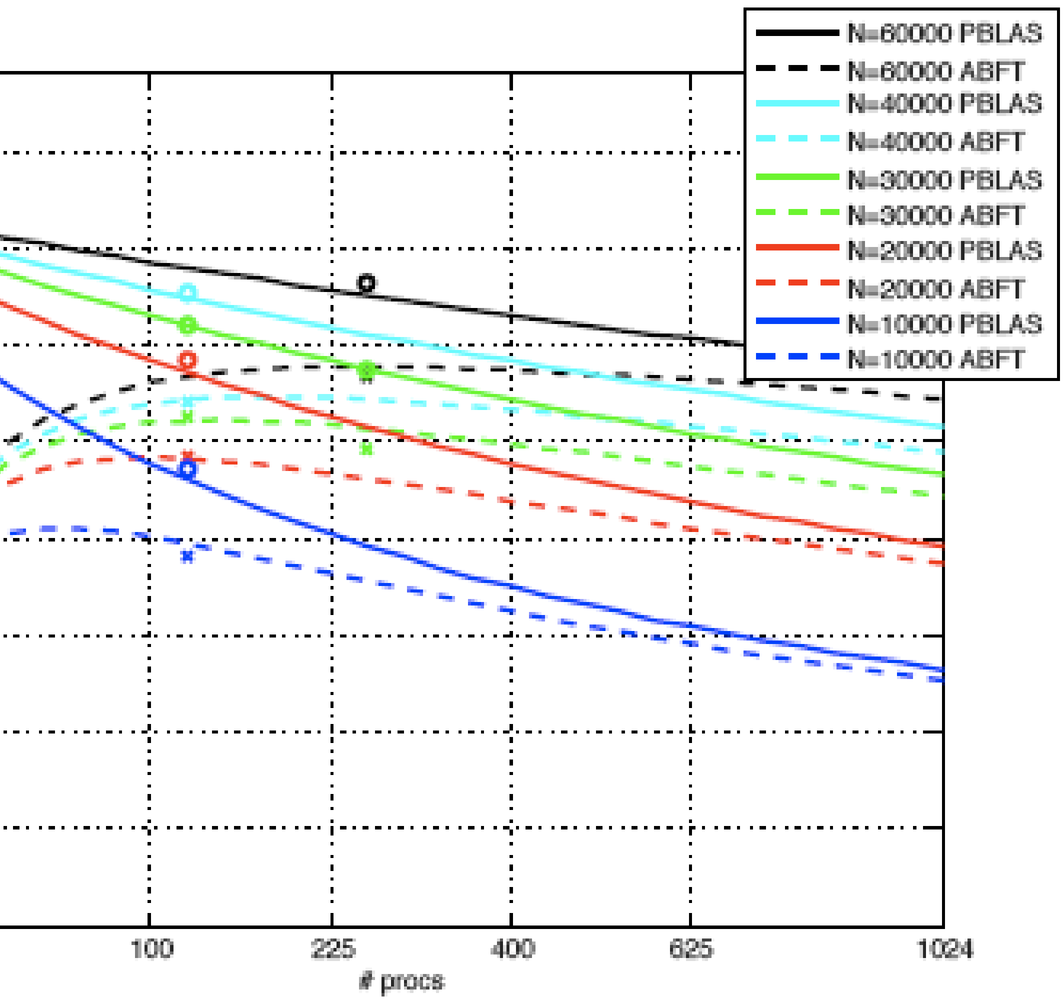


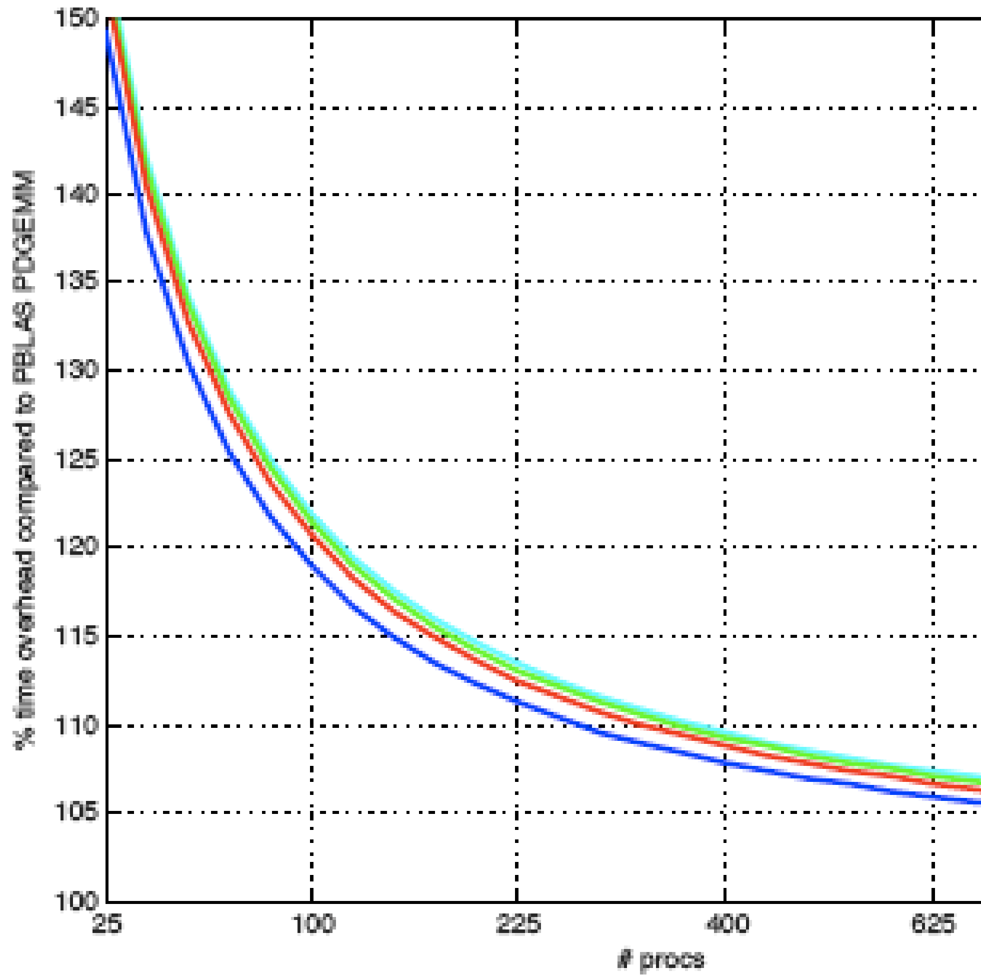
Weak scalability





Strong Scalability

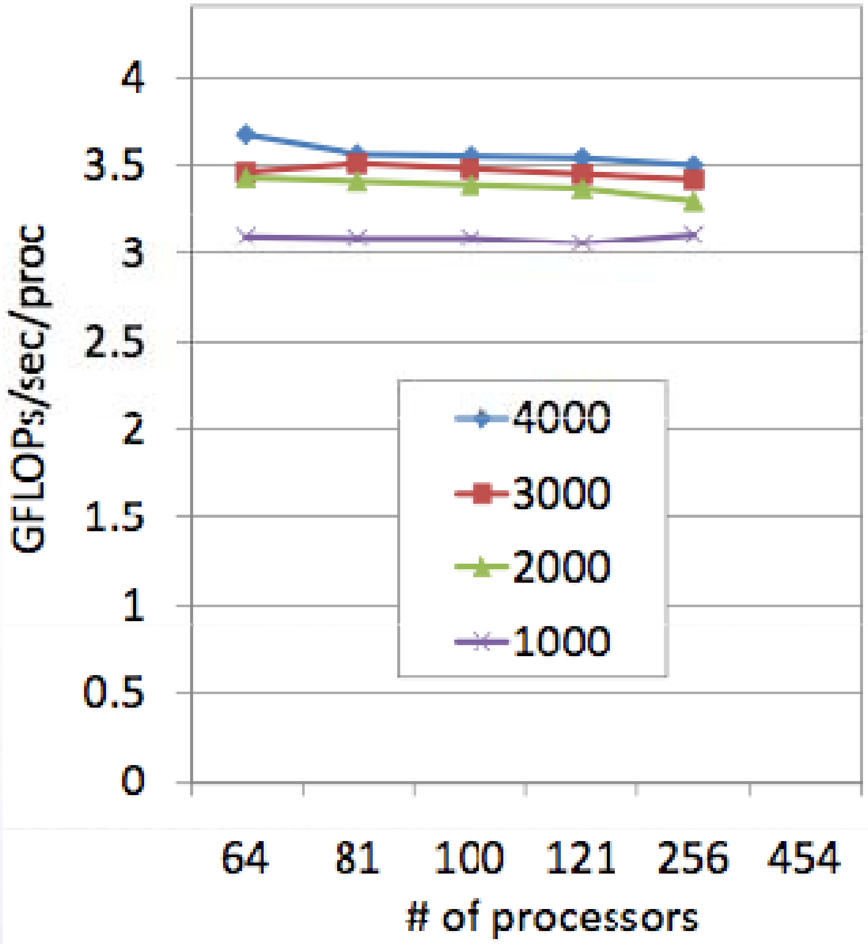


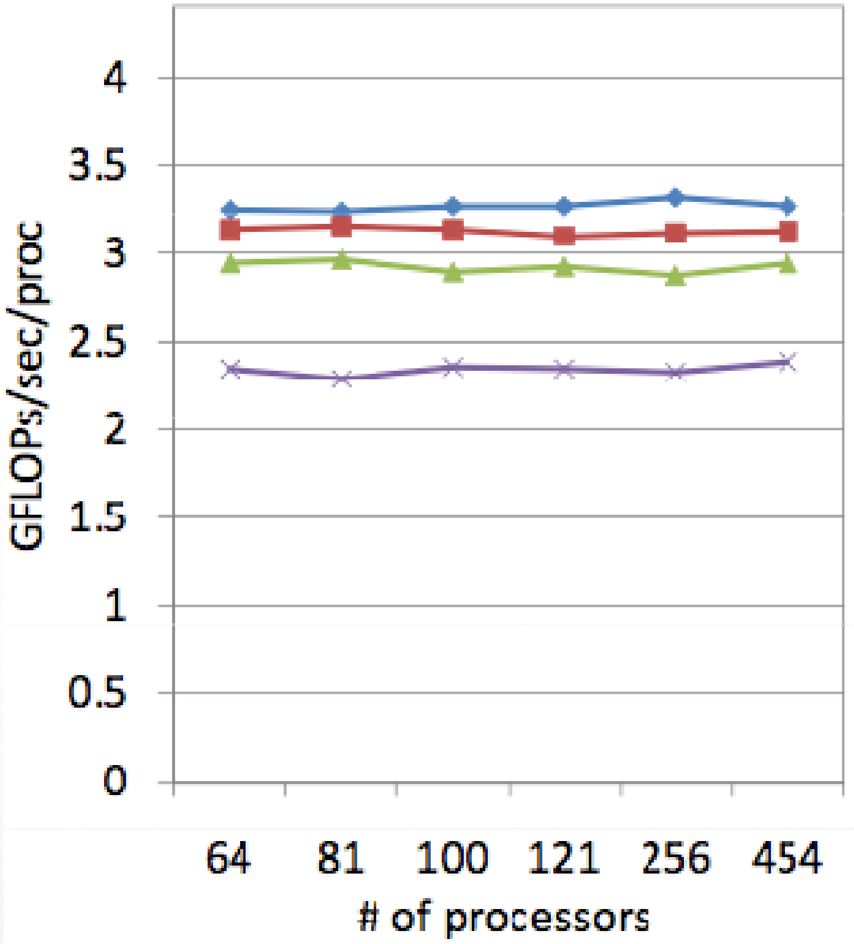


Conclusion

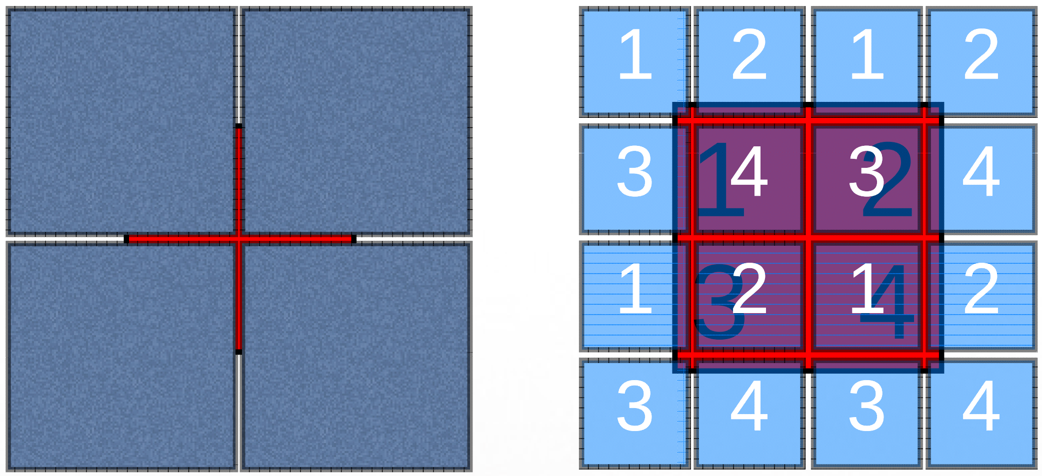
- Data-flow programming models an interesting alternative
- Fault tolerance is a requirement
  - FT-MPI approach a viable possibility with algorithms already available
- The future of MPI is decided now !

MVAPICH vs. FT-MPI

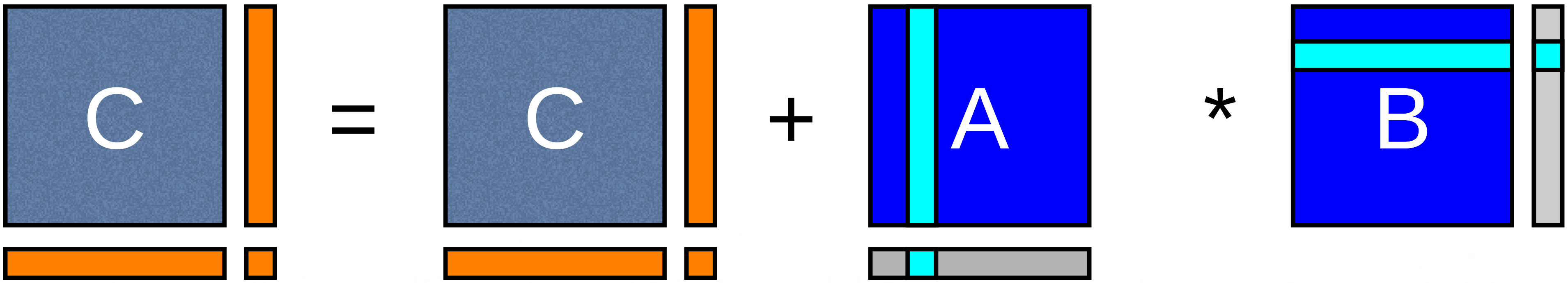




- MVAPICH over Infiniband
- FT-MPI over socket on Infiniband


1 2 1 2
3 4 3 4
1 2 1 2
3 4 3 4
1 2
3 4

ABFT-PDGEMM



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

- The algorithm maintain the consistency of the checkpoints of the matrix C naturally