www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Resilience for Task-Based Parallel Codes

Marc Casas Guix

Contributors

- Franck Capello
- Marc Casas
- Luc Jaulmes
- Jesus Labarta
- Tatiana Martsinkevich
- Omer Subasi
- Osman Unsal

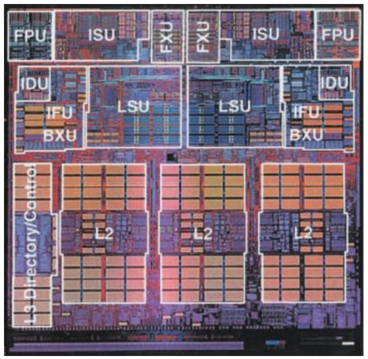
Outline

- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Conclusions

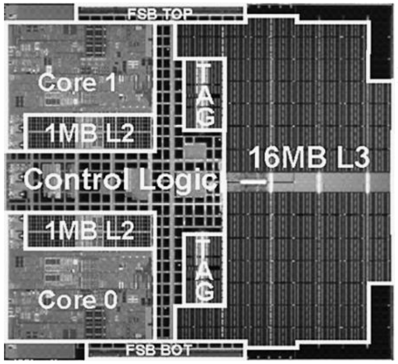
The MultiCore Era

- Moore’s Law + Memory Wall + Power Wall

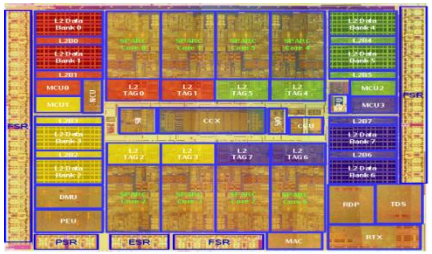
Chip MultiProcessors (CMPs)



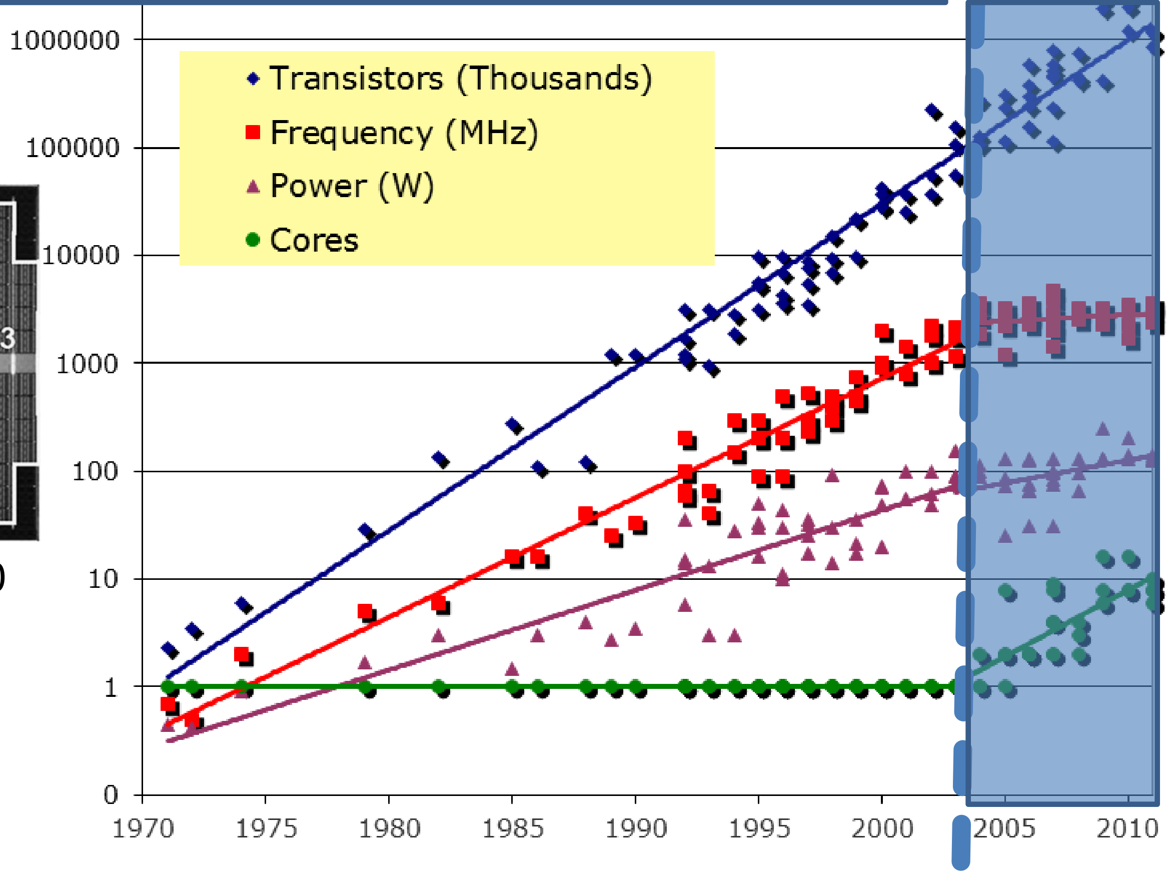
POWER4 (2001)



Intel Xeon 7100 (2006)



UltraSPARC T2 (2007)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

How are the Multicore architectures designed?

IBM Power4 (2001)

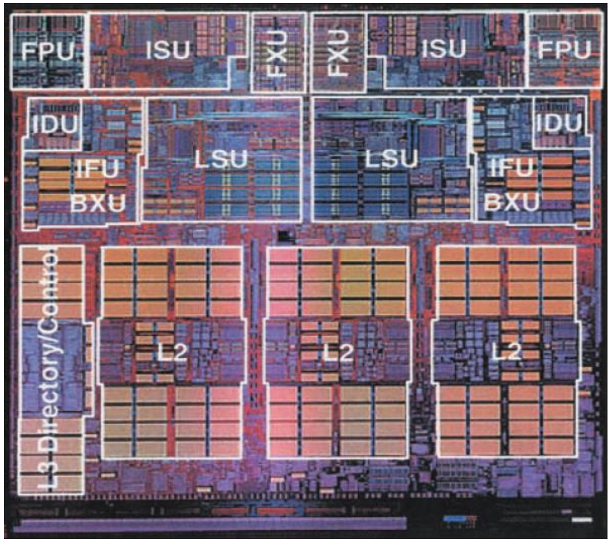
- 2 cores, ST
- 0.7 MB/core L2, 16MB/core L3 (off-chip)
- 115W TDP
- 10GB/s mem BW

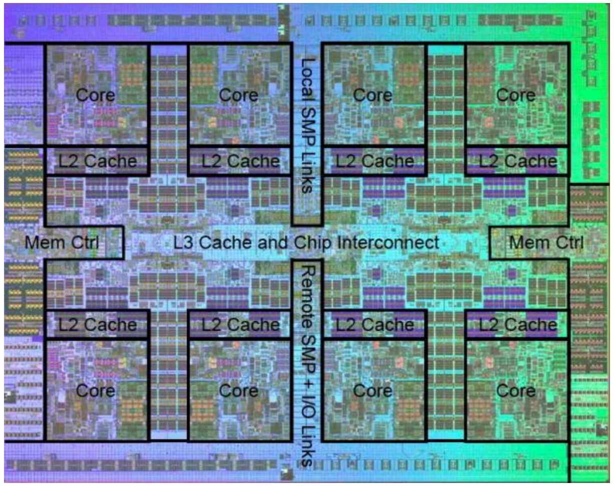
IBM Power7 (2010)

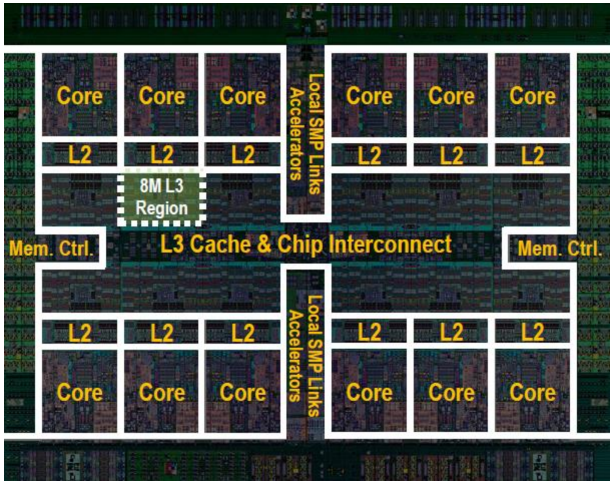
- 8 cores, SMT4
- 256 KB/core L2 16MB/core L3 (on-chip)
- 170W TDP
- 100GB/s mem BW

IBM Power8 (2014)

- 12 cores, SMT8
- 512 KB/core L2 8MB/core L3 (on-chip)
- 250W TDP
- 410GB/s mem BW

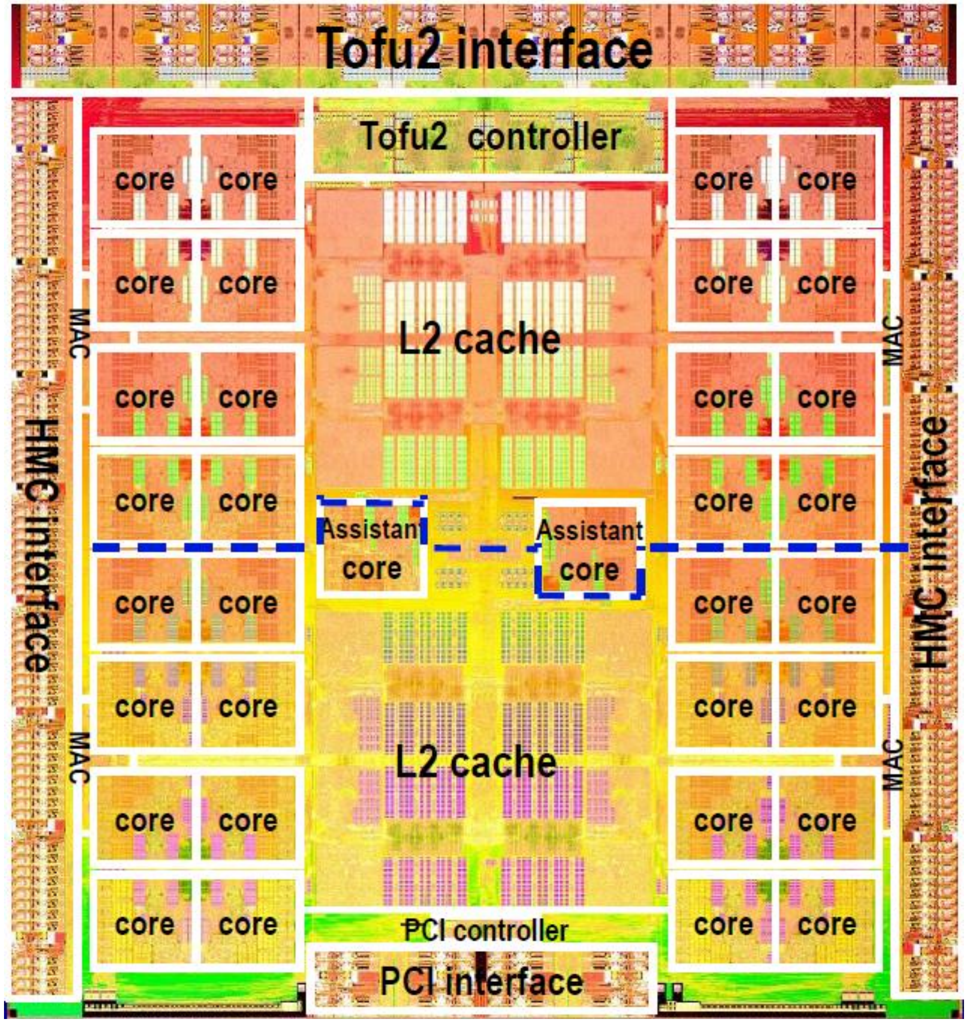




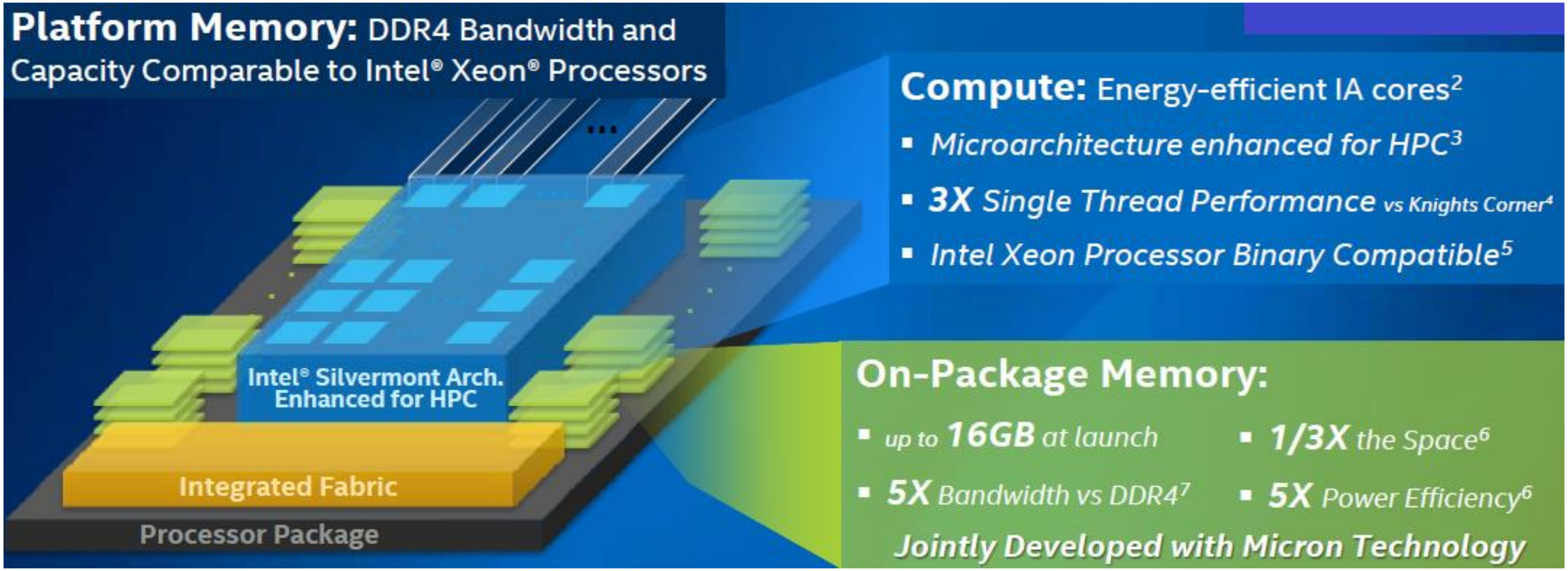


Fujitsu SPARC64 Xifx (2014)

- 32 computing cores (single threaded) + 2 assistant cores
- 24MB L2 sector cache
- 256-bit wide SIMD
- 20nm, 3.75M transistors
- 2.2GHz frequency
- 1.1TFlops peak performance
- High BW interconnects
  - HMC (240GB/s x 2 in/out)
  - Tofu2 (125GB/s x 2 in/out)



Intel Knights Landing (2016)

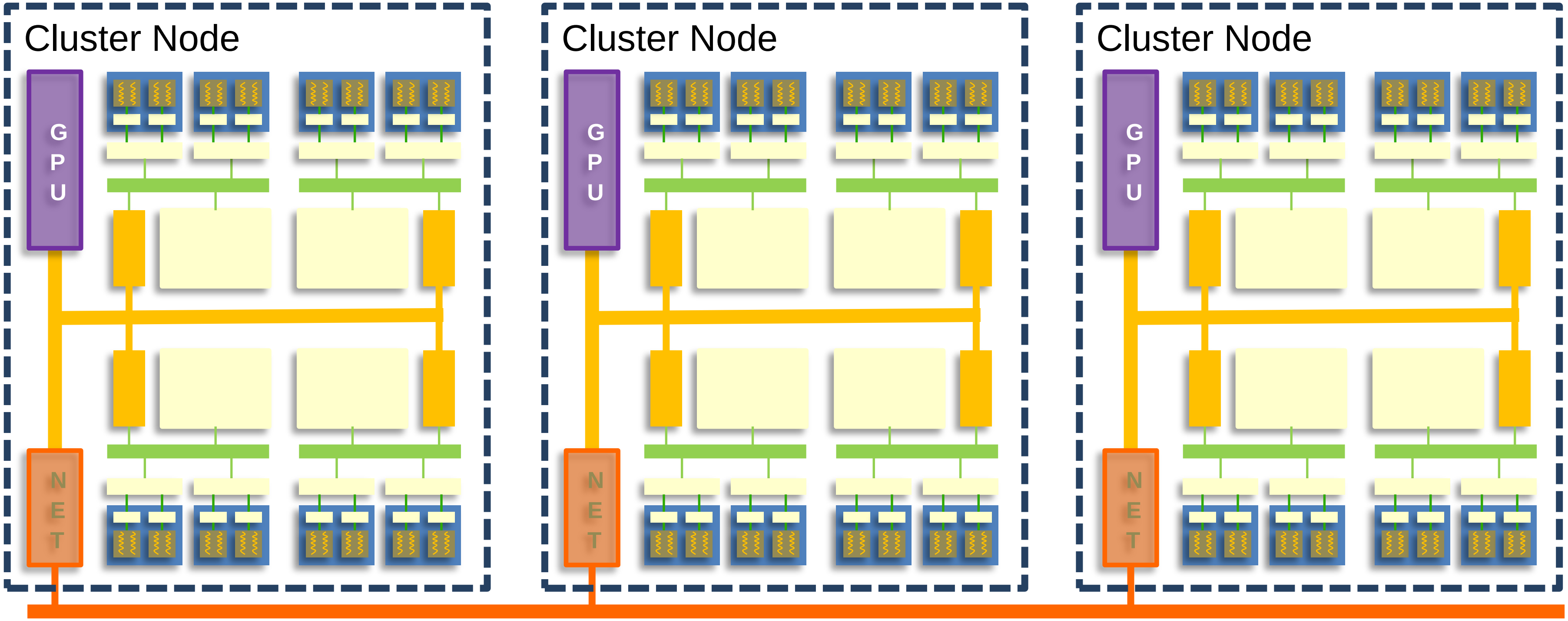


From Intel Corporation

- Intel's Knights Landing has a hybrid and reconfigurable memory hierarchy

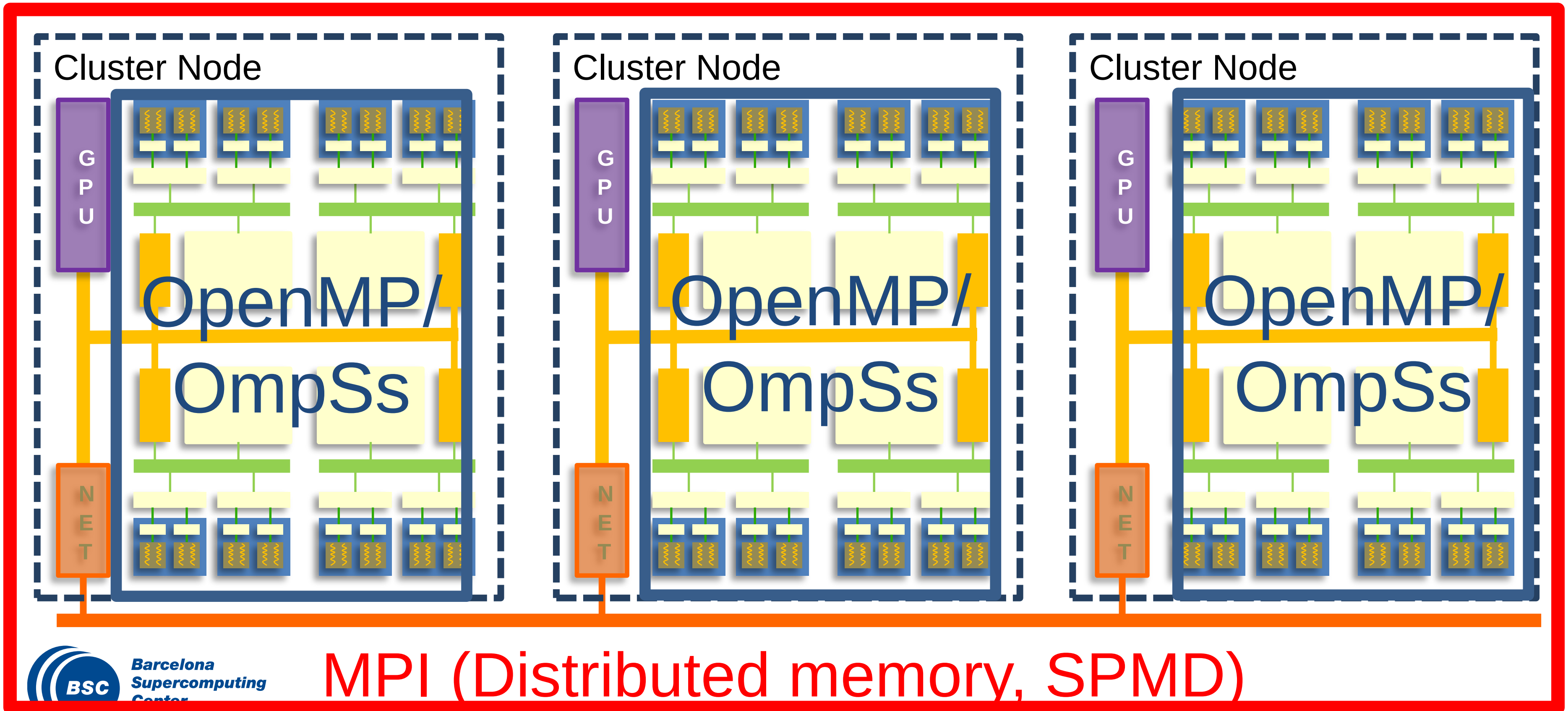
Cluster Machines

- SM or DSM machines interconnected
  - Distributed Memory → Multiple Address Spaces
  - Communication through interconnection network
- Usually allows multiple levels of parallelism



Cluster Machines

- SM or DSM machines interconnected
  - Distributed Memory → Multiple Address Spaces
  - Communication through interconnection network
- Usually allows multiple levels of parallelism



OmpSs: A Sequential Program …

```
void vadd3 (float A[BS], float B[BS],
            float C[BS]);

void scale_add (float sum, float A[BS],
               float B[BS]);

void accum (float A[BS], float *sum);
```

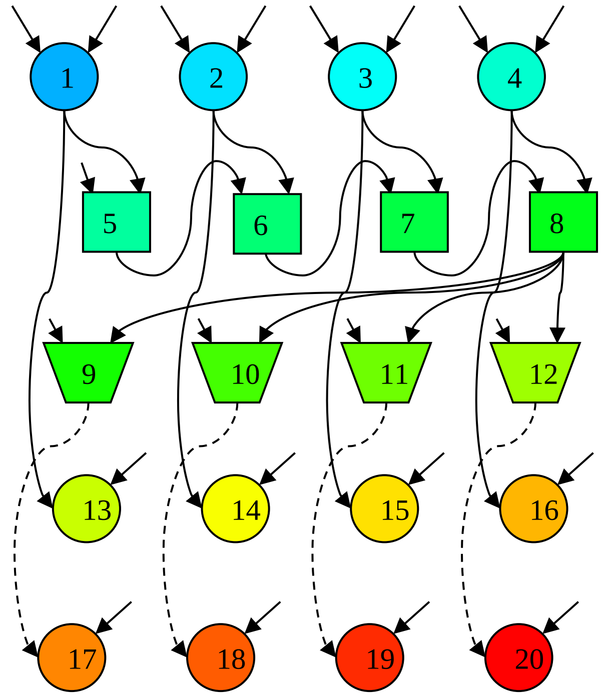
```
for (i=0; i<N; i+=BS)                  // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)                 //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)                 // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)                 // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)                 // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OmpSs: … Taskified …

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
               float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)                   // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)                  //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)                  // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)                  // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)                  // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```



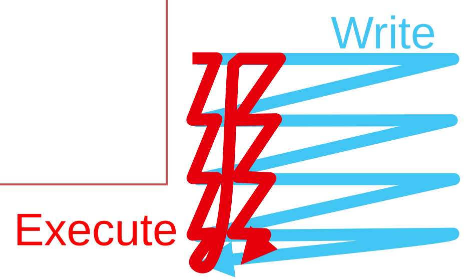
Color/number: order of task instantiation
Some antidependences covered by flow dependences not drawn

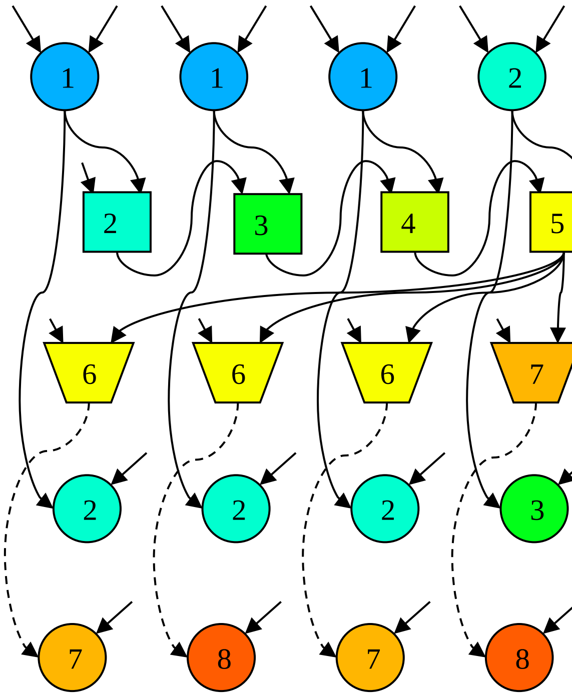
OmpSs: … and Executed in a Data-Flow Model

```
#pragma css task input(A, B) output(C)
void vadd3 (float A[BS], float B[BS],
            float C[BS]);
#pragma css task input(sum, A) inout(B)
void scale_add (float sum, float A[BS],
               float B[BS]);
#pragma css task input(A) inout(sum)
void accum (float A[BS], float *sum);
```

```
for (i=0; i<N; i+=BS)                    // C=A+B
   vadd3 ( &A[i], &B[i], &C[i]);
...
for (i=0; i<N; i+=BS)                   //sum(C[i])
   accum (&C[i], &sum);
...
for (i=0; i<N; i+=BS)                   // B=sum*A
   scale_add (sum, &E[i], &B[i]);
...
for (i=0; i<N; i+=BS)                   // A=C+D
   vadd3 (&C[i], &D[i], &A[i]);
...
for (i=0; i<N; i+=BS)                   // E=G+F
   vadd3 (&G[i], &F[i], &E[i]);
```

Decouple
how we write
form
how it is executed





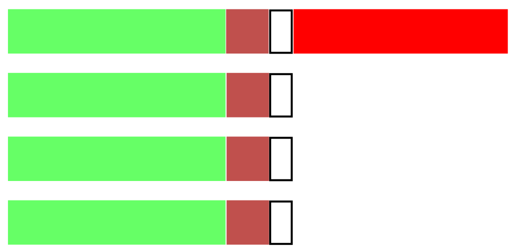
Color/number: a possible order of task execution

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

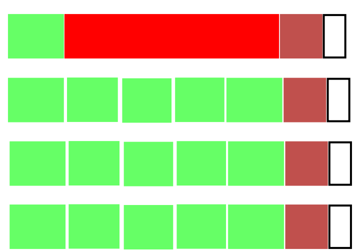
OmpSs/OpenMP4.0: Data-flow and asynchronous execution

Four loops/routines
Sequential program order

OpenMP 2.5
not parallelizing one loop



OmpSs/OpenMP4.0
not parallelizing one loop



PARSEC

Initial port from Pthreads to OmpSs and optimization

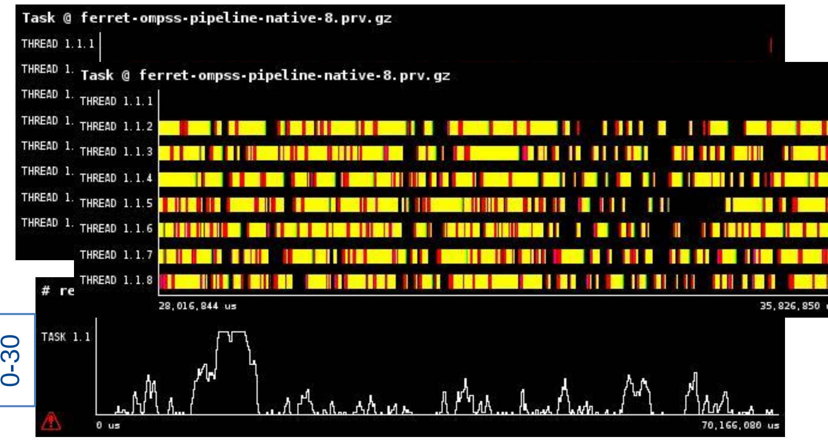
Bodytrack

Ferret

"Direct"

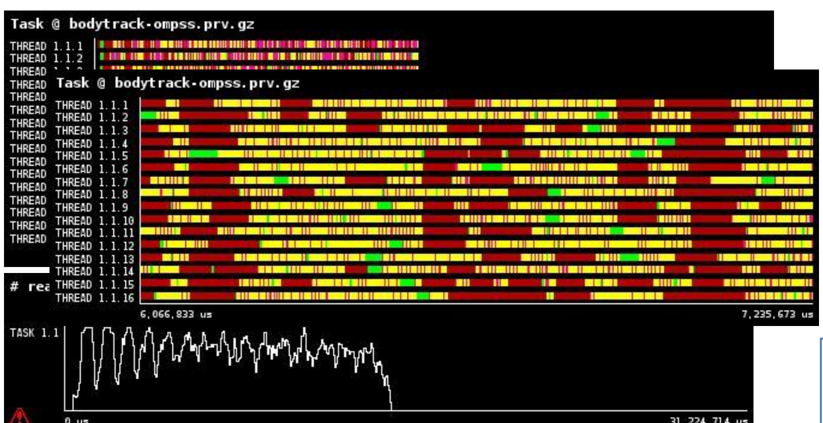
0-10

0-30

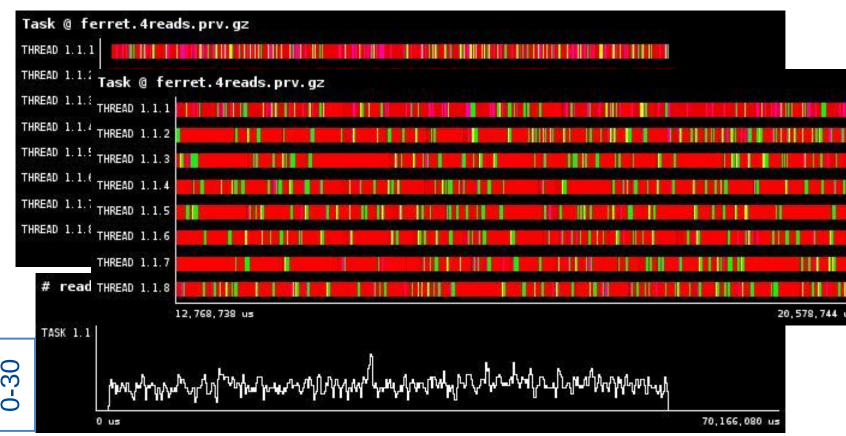


"optimized"

0-250



0-30



Barcelona Supercomputing Center
Centro Nacional de Superco

"Exploring the Impact of Task Parallelism in the PARSEC benchmark suite". TACO'2016

Outline

- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Conclusions

Motivation: Error Trends

- Increased number of memory errors
  - Error Correcting Codes (ECC) make correctable errors transparent
  - Checkpointing-Restart (CR) enables recovery from uncorrectable error
  - ECC may not be able to address failures projected for next generation systems
  - Frequent system failure and high CR overhead projected
- Soft Errors
  - Unpredictable transient errors
  - Expected decrease due technological trends
- Hard Errors
  - Recurring errors caused by aging of transistors, expected to be the dominant type
  - Symptom based techniques (ECC corrections) can be used to deal with hard errors

Source: IBM, SC'2014

Error detection and reporting

- Modern architectures discover data that is incoherent with memory ECC’s

- Faulty Memory Page Management
  - If number of errors exceeds a threshold, the OS relocates the page at another physical location
    - Memory Page Retirement in Solaris
    - Page off-lining in Linux Kernels
  - If a Detected and Uncorrectable Error (DUE) is reported,
    - The OS kills the affected process via a signal that also specifies the faulty page

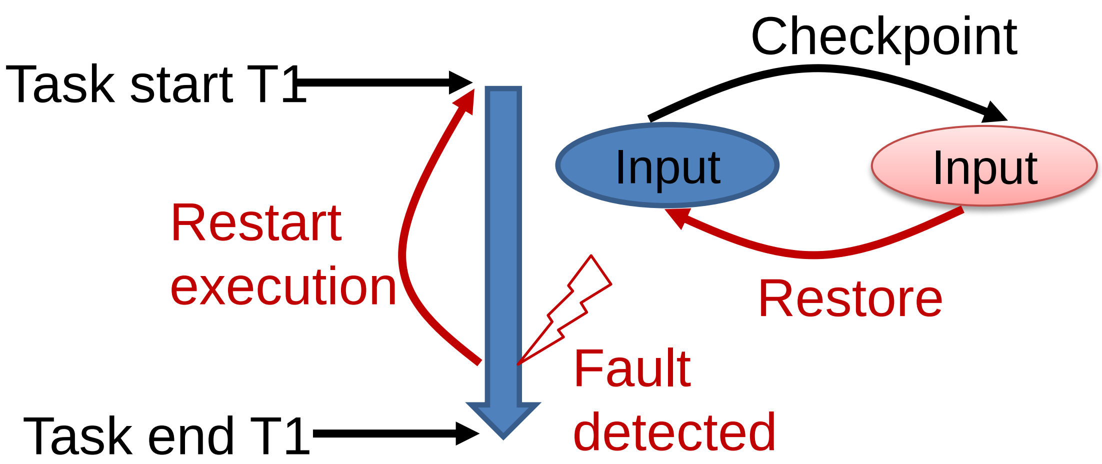
- Thus, to be resilient against memory DUE, an application “simply” has to replace data contained in the faulty page

Outline

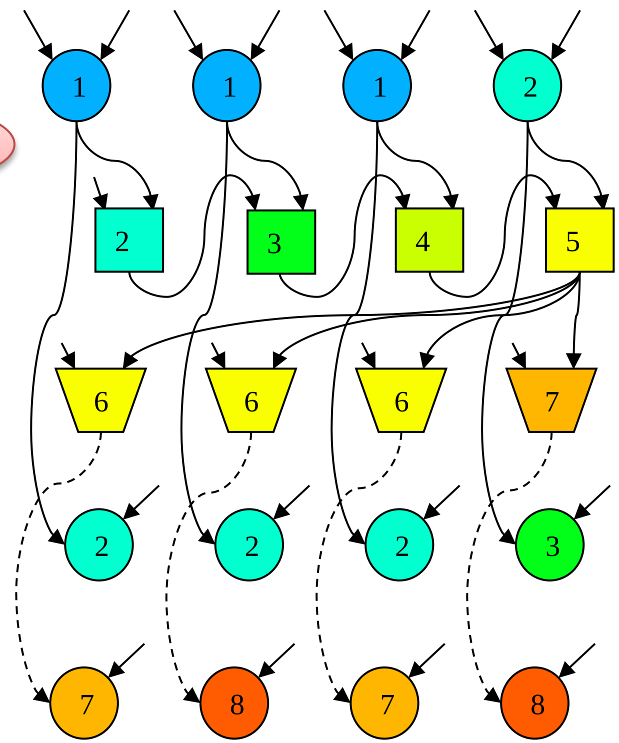
- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Evaluation
- Conclusions

Our Design: Checkpoint and Restart of Tasks

- Recover task execution from the detected errors.
- Contain errors within the task boundaries



- Inputs are known at runtime through programmer annotations.
- Recovery is asynchronous



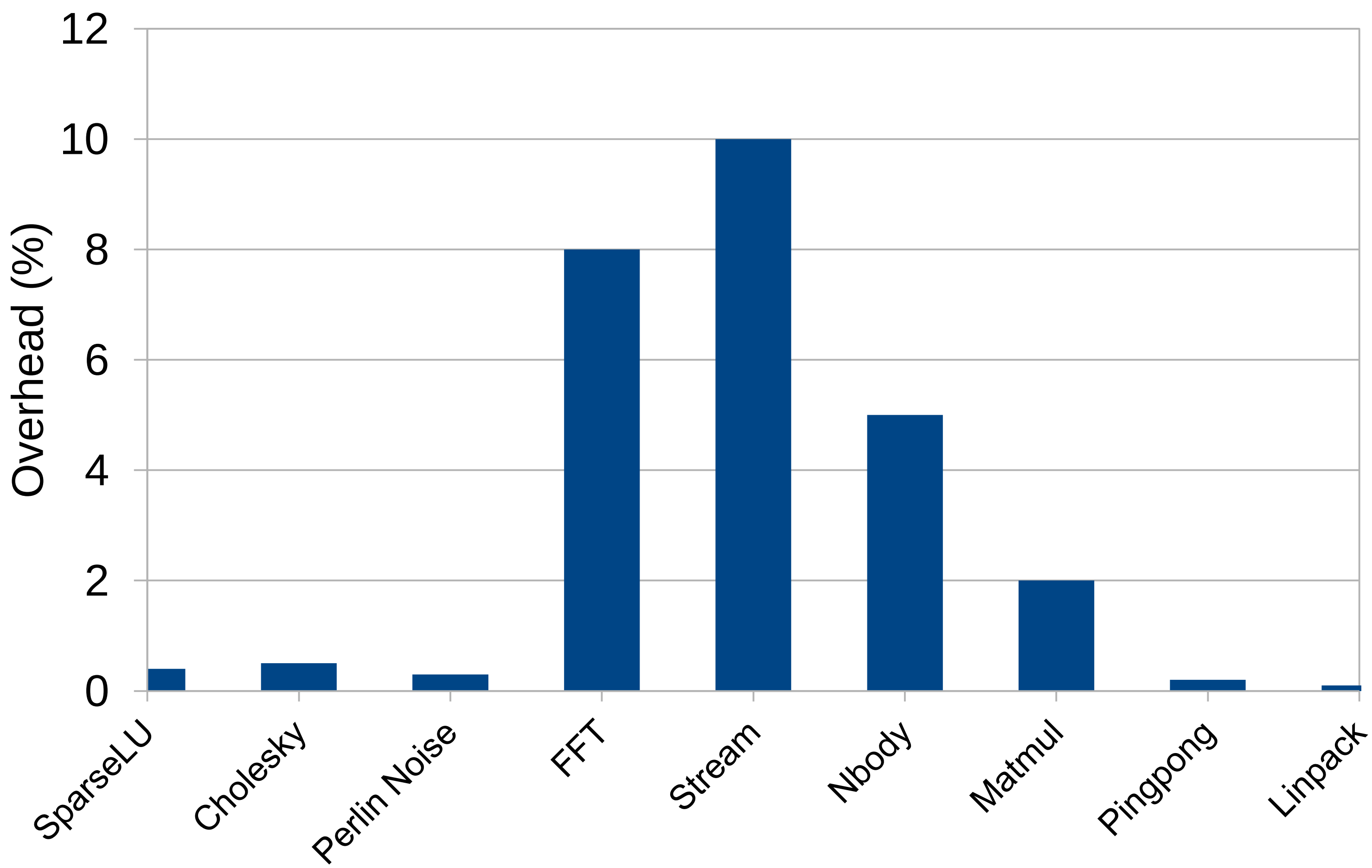
Experimental Setup

- Marenostrum supercomputer at Barcelona Supercomputing Center
- Two sets of benchmarks:
  - Task-parallel SMP (Shared memory)
  - Hybrid OmpSs+MPI
- Fault injection to evaluate the overhead of recovery and stress NanoCheckpoints

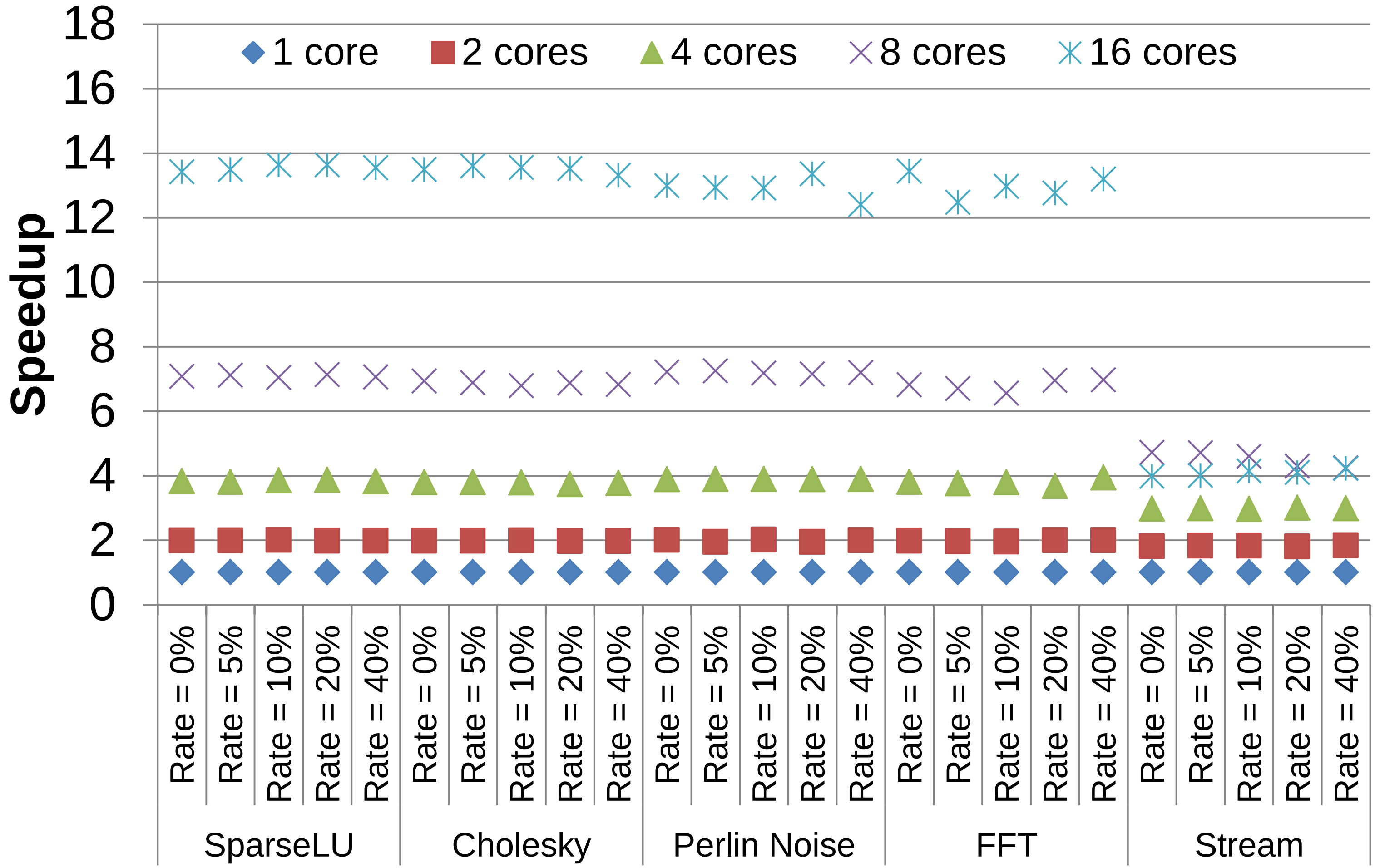
Benchmarks

Task-parallel SMP (Shared memory)	
Sparse LU	Matrix size 6400x6400, block size 100x100
Cholesky	Matrix size 16384x16384, block size 512x512
FFT	Array size 16384x16384 (complex doubles), block size 16384x128
Perlin Noise	Array of pixels with size of 65536 (1500 iterations), block size
Stream	Array size 2048x2048 (doubles), block size 32768
Hybrid OmpSs+MPI	
Nbody	65536 bodies, block size depends on #nodes
Matrix Multiplication	Matrix size 9216x9216 and block size 1024x1024
Pingpong	Array size 65536, block size 1024
Linpack	Matrix size 131072, block size 256x256

Fault-Free Overheads

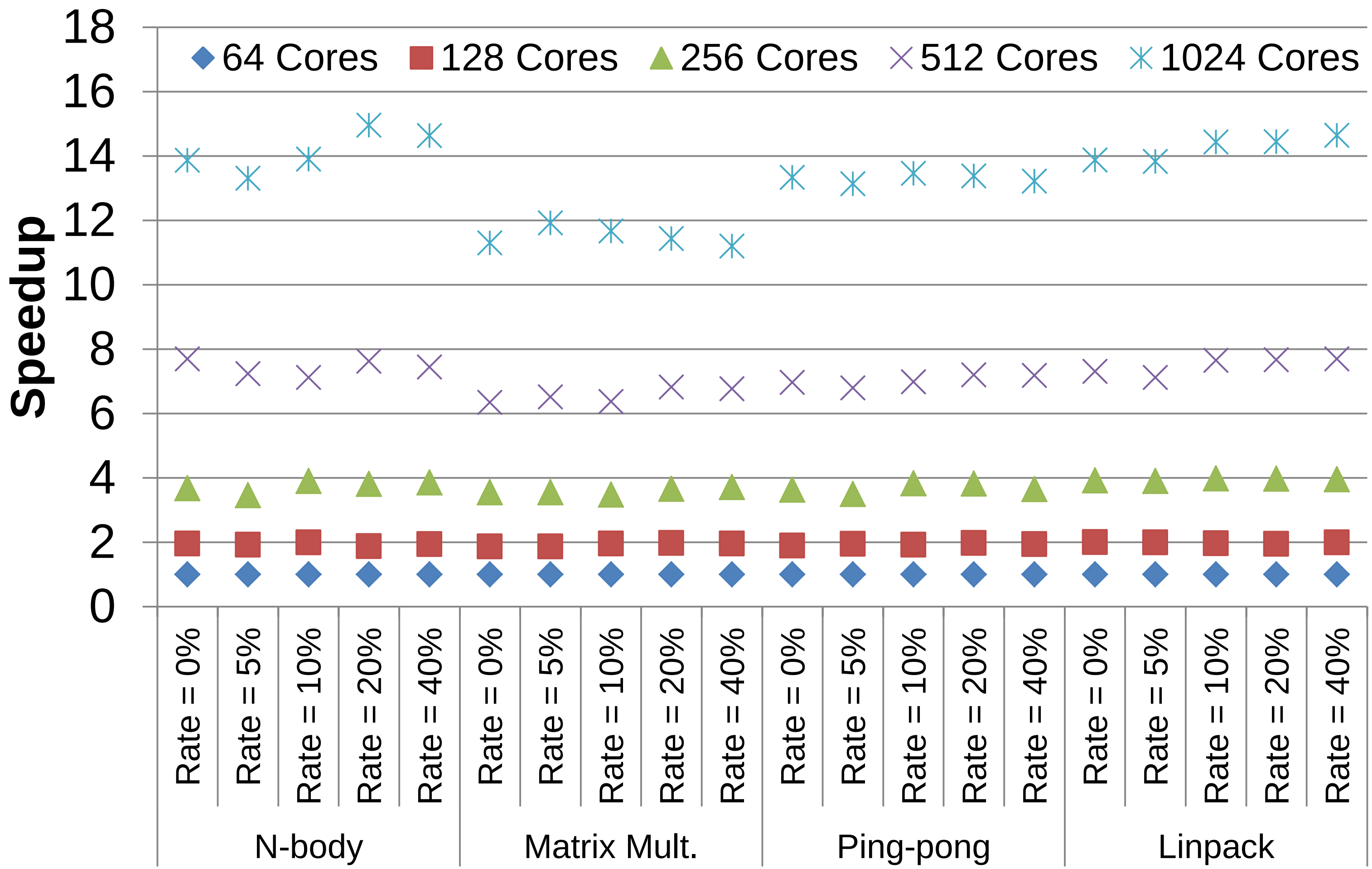


Results: Scalability (1)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Results: Scalability (2)



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Outline

- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Conclusions

Error detection and reporting

- Modern architectures discover data that is incoherent with memory ECC's

- Faulty Memory Page Management
  - If number of errors exceeds a threshold, the OS relocates the page at another physical location
    - Memory Page Retirement in Solaris
    - Page off-lining in Linux Kernels
  - If a Detected and Uncorrectable Error (DUE) is reported,
    - The OS kills the affected process via a signal that also specifies the faulty page

- Thus, to be resilient against memory DUE, an application "simply" has to replace data contained in the faulty page

Extracting Trivial Redundancies of Linear Operations

- Linear transformations consists in operations like:
  - Matrix-vector multiplication $q = Ap$
  - Linear combinations of vectors $u = \alpha v + \beta \omega$
  - Gradient or residual $g = b - Ax$
  - …

- It is possible to decompose these relationships into several blocks

$q_i = \sum_{j=0}^{n-1} A_{ij} p_j$	$A_{ii} p_i = q_i - \sum_{j \neq i} A_{ij} p_j$
$u_i = \alpha v_i + \beta w_i$	$w_i = (u_i - \alpha v_i)/\beta$
$g_i = b_i - \sum_{j=0}^{n-1} A_{ij} x_j$	$A_{ii} x_i = b_i - g_i - \sum_{j \neq i} A_{ij} x_j$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Forward Recoveries Based on Trivial Redundancies

- In many cases, the left hand side (LHS) and the right hand side (RHS) of these operations coexist
- Recoveries:
  - LHS is trivial
  - RHS involves inverting A_{ii}, only possible if block size is small
- The block size that we use for recovery is a memory page of 4K bytes, that is, 512 double precision floating-point values

Block relation, recover lhs	Inverted relation, recover rhs
$q_i = \sum_{j=0}^{n-1} A_{ij} p_j$	$A_{ii} p_i = q_i - \sum_{j \neq i} A_{ij} p_j$
$u_i = \alpha v_i + \beta w_i$	$w_i = (u_i - \alpha v_i)/\beta$
$g_i = b_i - \sum_{j=0}^{n-1} A_{ij} x_j$	$A_{ii} x_i = b_i - g_i - \sum_{j \neq i} A_{ij} x_j$

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Dealing with multiple faults by using Forward Recoveries

- Multiple faults in a single vector
  - Trivial for vectors recovered from linear relationships
  - For sub-matrix inversion relations

$$\begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix} \begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} b_i - g_i - \sum_{k \neq i,j} A_{ik} x_k \\ b_j - g_j - \sum_{k \neq i,j} A_{jk} x_k \end{pmatrix}$$

- For multiple faults in related data, i. e., q_i and $p_j$in $q = Ap$
  - Alternative relationship that allows to recover each piece of lost data separately
  - Rollback mechanism

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Extracting Forward Recoveries from CG

- The relation that last produced data is used
- The invariant $g = b - Ax$ makes possible to protect x and g updates
- When updating d, we can not use $d = A^{-1}q$ to recover d_i, because in the block formulation

Listing 1: CG pseudo code

$\epsilon_{old} \Leftarrow +\infty$	
$g \Leftarrow b - Ax$	
for t in $0..t_{max}$	
$\quad \epsilon \Leftarrow \|\|g\|\|^2$	$g = b - Ax$
$\quad$ if $\epsilon < tol$: break	
$\quad \beta \Leftarrow \epsilon / \epsilon_{old}$	
$\quad \boxed{d \Leftarrow \beta d + g}$	$\boxed{d = A^{-1}q \quad g = b - Ax}$
$\quad q \Leftarrow Ad$	
$\quad \alpha \Leftarrow \epsilon / < q, d >$	$q = Ad \quad d = A^{-1}q$
$\quad x \Leftarrow x + \alpha d$	$d = A^{-1}q \quad x = A^{-1}(b - g)$
$\quad g \Leftarrow g - \alpha q$	$q = Ad \quad g = b - Ax$
$\quad \epsilon_{old} \Leftarrow \epsilon$	

$$d_i = A_{ii}^{-1}\left(q_i - \sum_{j \neq i} A_{ij} d_j\right)$$

the parameters $d_0, \ldots, d_{i-1}$ are at iteration k+1 while $d_{i+1}, \ldots d_{n-1}$ are at iteration k

Effective Protection of Iterative Solvers

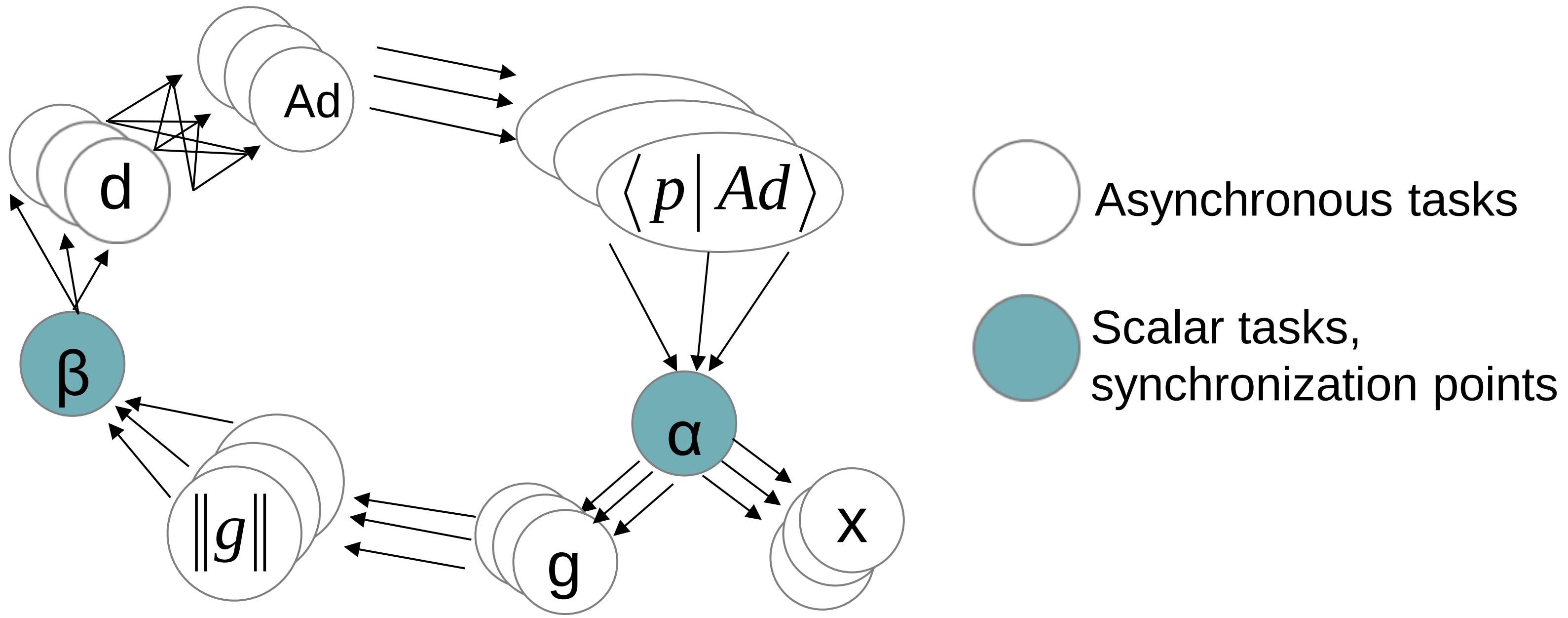
- Double buffering to protect d update in CG
- Trivial linear relations to protect the other CG transformations
- The invariant $g = b - Ax$ plays a fundamental role
- General scheme in iterative solvers:
  - Bi-Conjugate Gradient Stabilized (BiCGStab) requires 1 variable double buffered plus trivial linear relations
  - Generalized Minimal Residual (GMRES) does not require double buffering

Double buffered CG

for t in $0..t_{max}$
    $\ldots$
    $d_1 \Leftarrow \beta d_2 + g$
    $q \Leftarrow A d_1$
    $\alpha \Leftarrow \epsilon / < q, d_1 >$
    $x \Leftarrow x + \alpha d_1$
    $\ldots$
    $t{+}{+}$
    $d_2 \Leftarrow \beta d_1 + g$
    $q \Leftarrow A d_2$
    $\alpha \Leftarrow \epsilon / < q, d_2 >$
    $x \Leftarrow x + \alpha d_2$

Conjugate Gradient

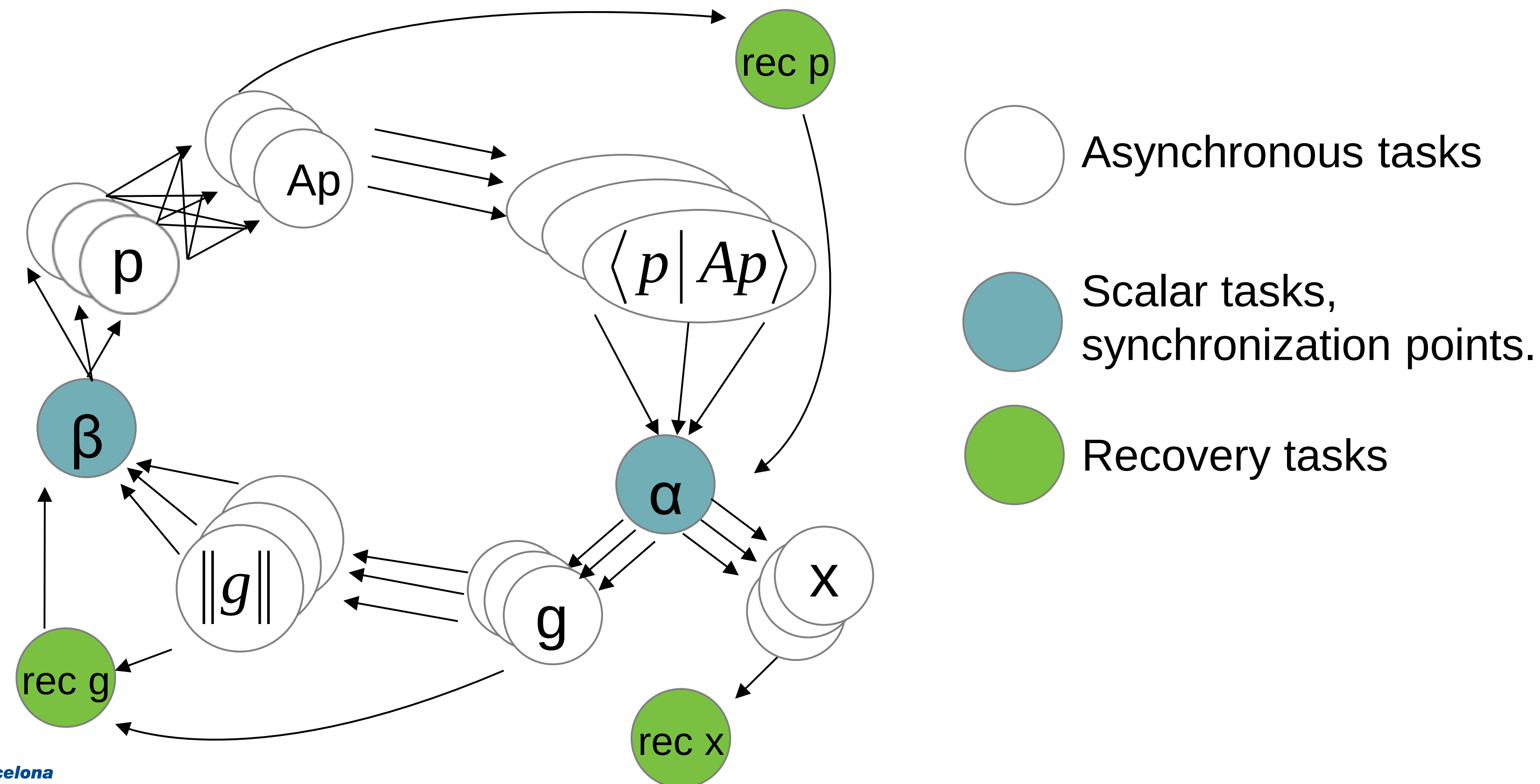
- Representation of one iteration:



- Errors need to be corrected before scalar tasks to avoid propagation

Detecting and Correcting Memory Errors in CG

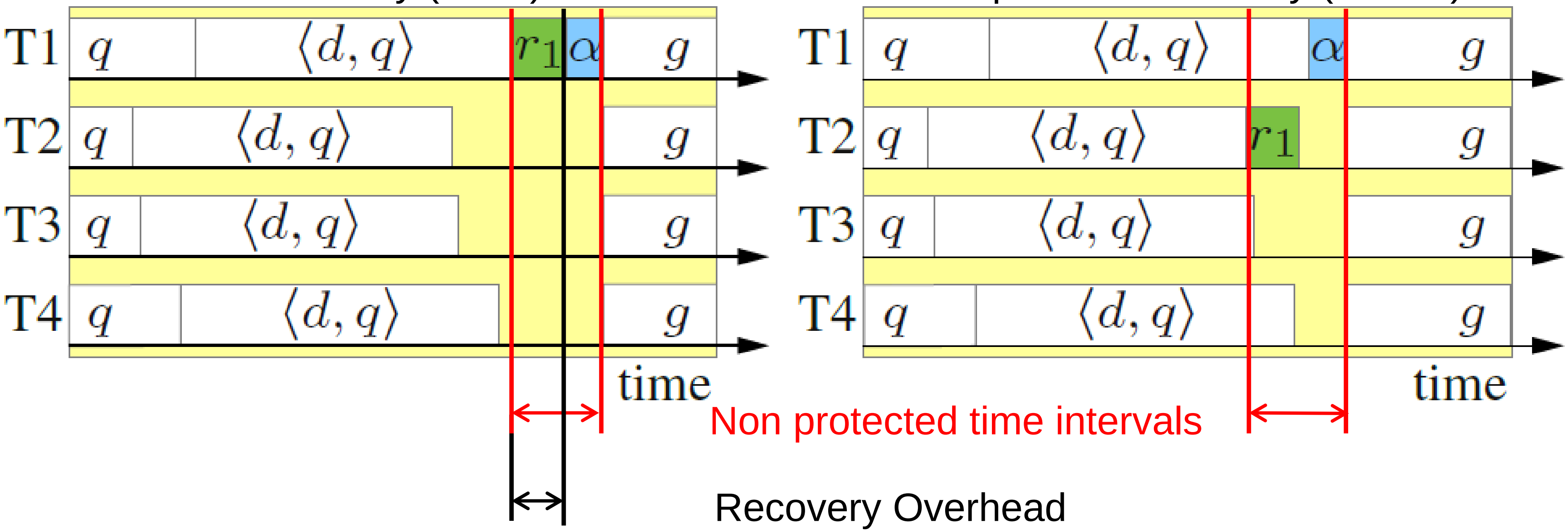
- Retired memory pages are marked as “failed” inside application and skipped in subsequent computations
- Recovery tasks are executed to fill in missing data and complete partial computations before synchronization points



Deployment of the Forward Recovery

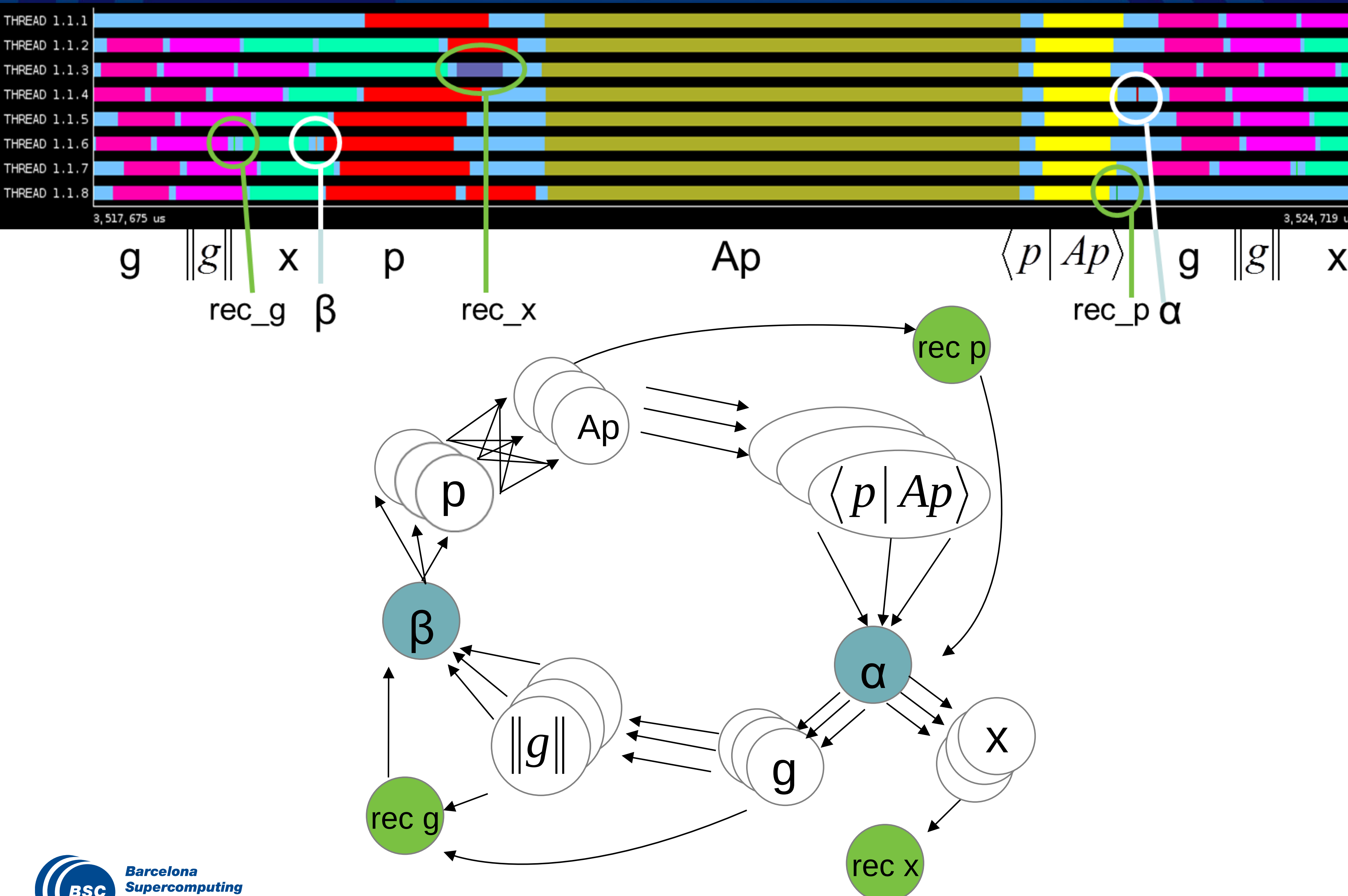
Forward Exact Interpolation Recovery (FEIR)

Asynchronous Forward Exact Interpolation Recovery (AFEIR)



- Trade-off: Recovery Overhead vs Errors Coverage

OmpSs makes the implementation easy



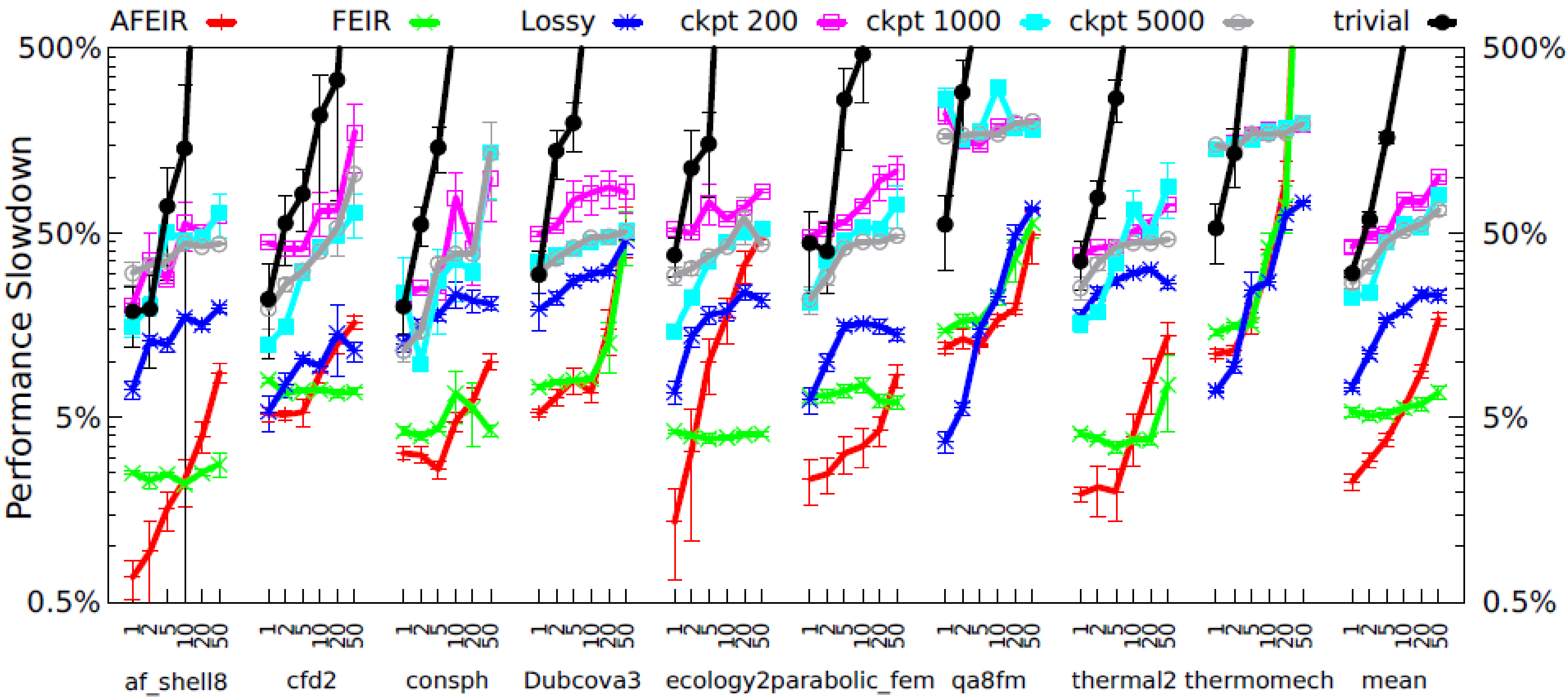
Experimental Evaluation

- FEIR/AFEIR approaches
- Checkpoiting the x vector to disk every:
  - 200 CG iterations
  - 1000 CG iterations
  - 5000 CG iterations
- Lossy Restart Approach
- Trivial

J. Langou, Z. Chen, G. Bosilca and J. Dongarra, SIAM Journal of Scientific Computing, 2007
E. Agullo, L. Giraud, et al, "Towards Resilient Parallel Krylov Solvers", 2013

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

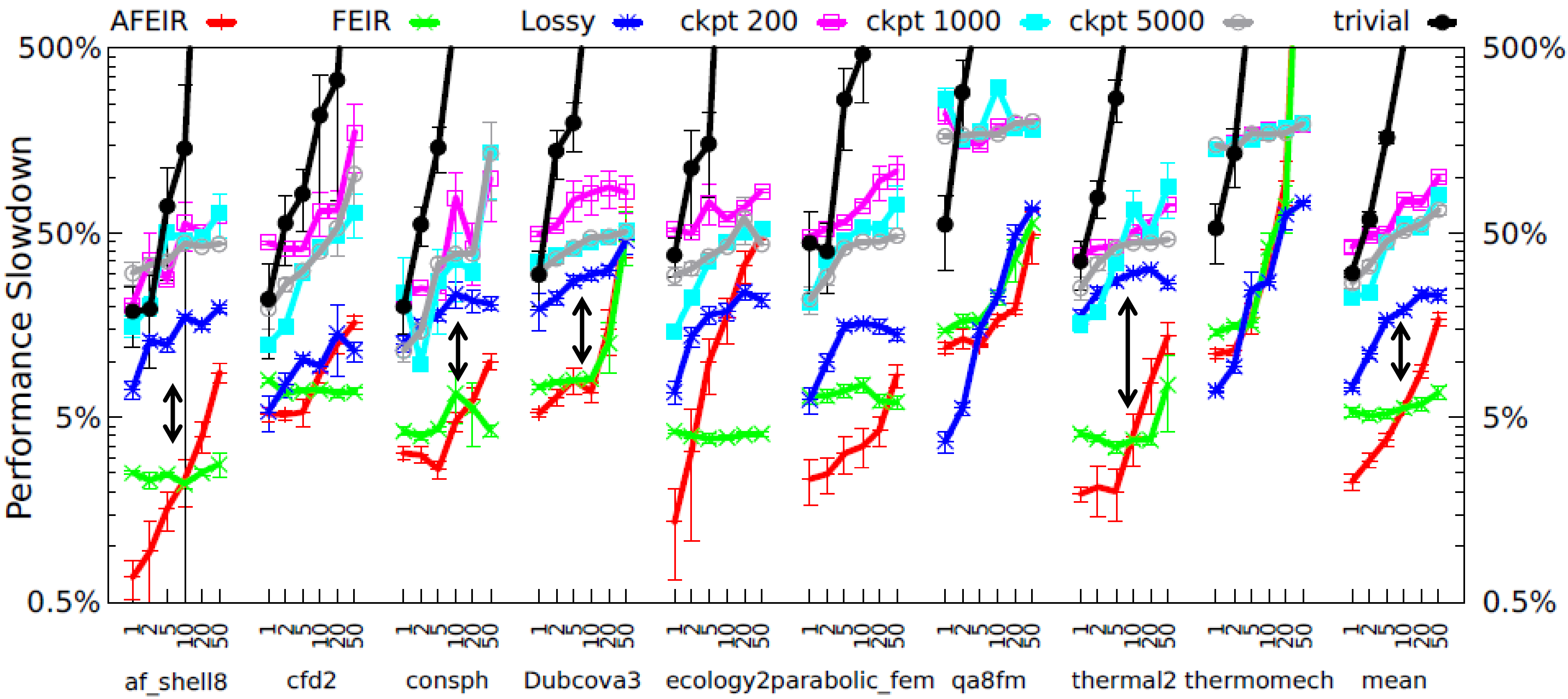
Evaluation Experiments



- 7 different resilience techniques, 9 test matrices, 6 different fault rates per matrix
- 20 8-cores runs per experiment with faults injected randomly via the mprotect system call
- For each experiment we report harmonic mean and standard deviation

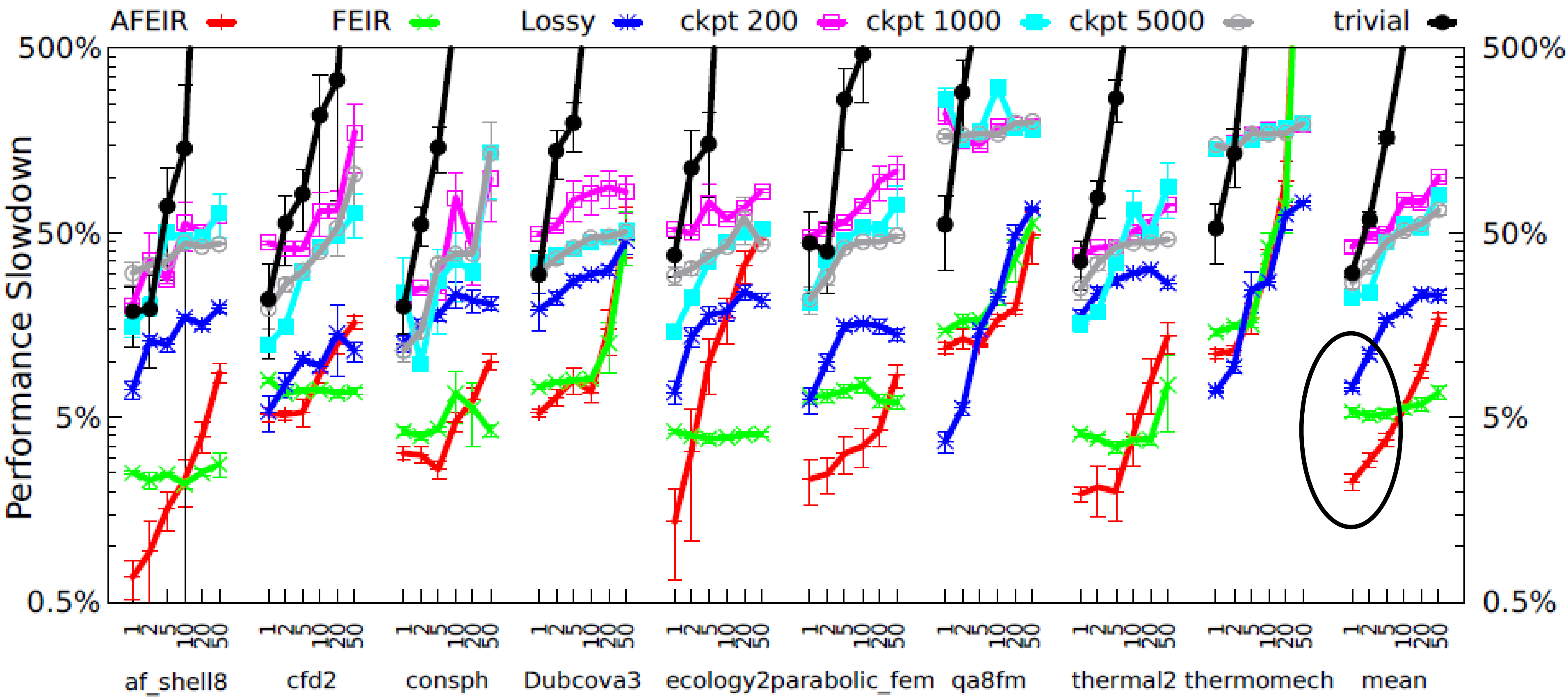
method	Lossy	Trivial	AFEIR	FEIR	ckpt 5000	ckpt 1000	ckpt 200
overhead	0.00%	0.00%	0.23%	2.73%	8.21%	17.62%	46.20%

Evaluation Experiments



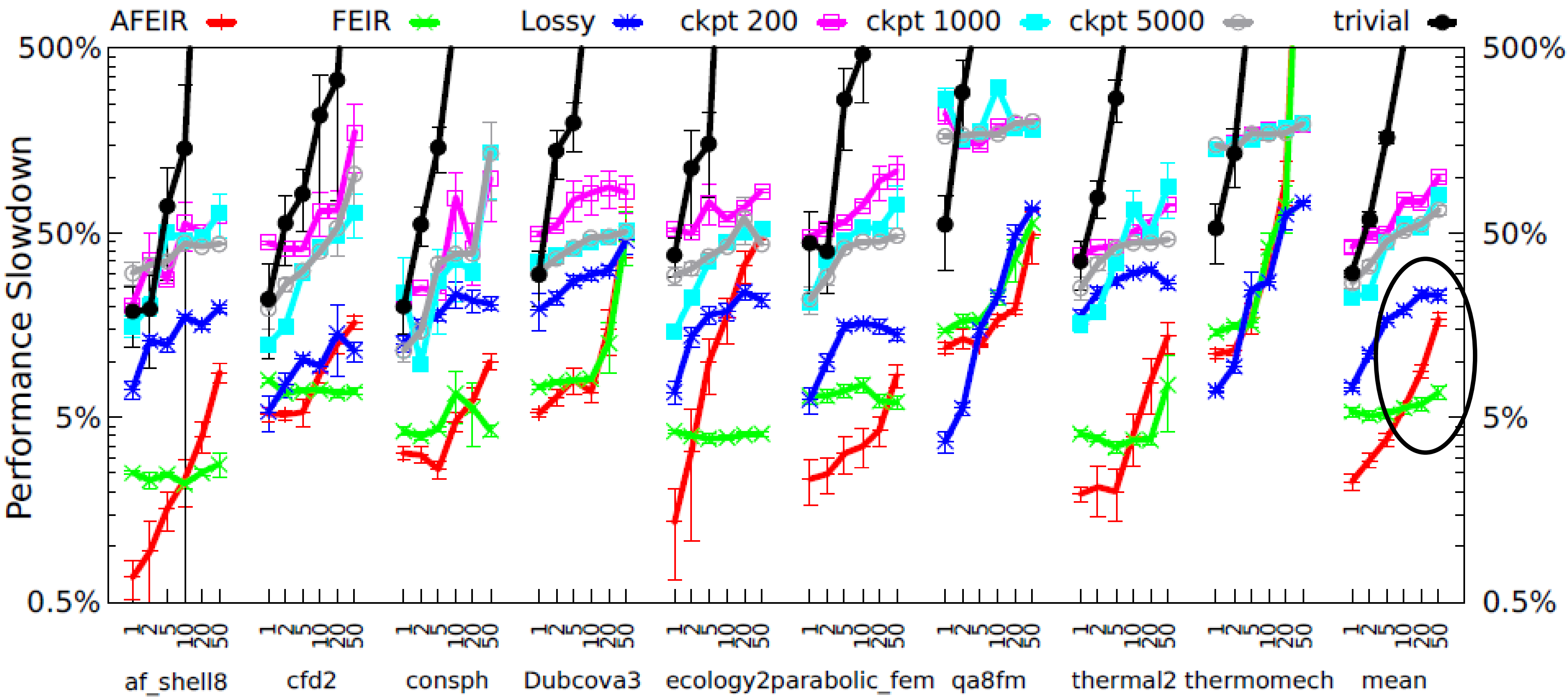
Exact recovery performs better than the lossy restart approach

Evaluation Experiments



Overlapping the exact recovery with computation is the best option if less than 10 errors per run are injected (0.1 error/second)

Evaluation Experiments



Recovery/computation overlap stops paying off when more than 10 errors per run are injected (0.1 error/second)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Outline

- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Conclusions

Hybrid MPI + OmpSs

- Shared + Distributed Programing Model
  - Analogous to MPI+OpenMP

- Intra-node communications can be encapsulated into a task

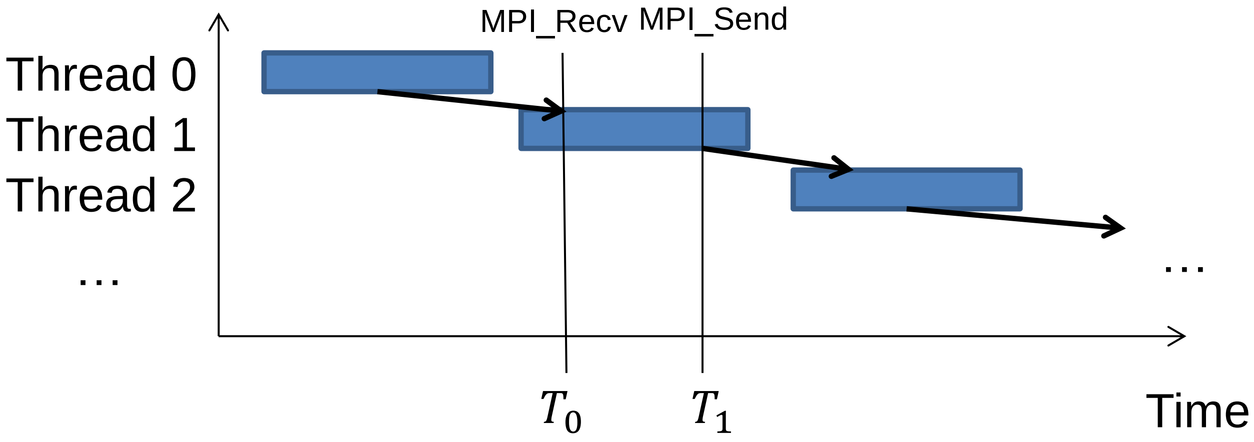
- Communication/Computation overlap is the goal

```
#pragma omp task input(A, B)
inout(C)
void mxm (double A[N], double
B[N],double C[N]);

#pragma omp task input(A)
output(receivebuf)
void SendRecv (double A[N],double
receivebuf[N]);
```

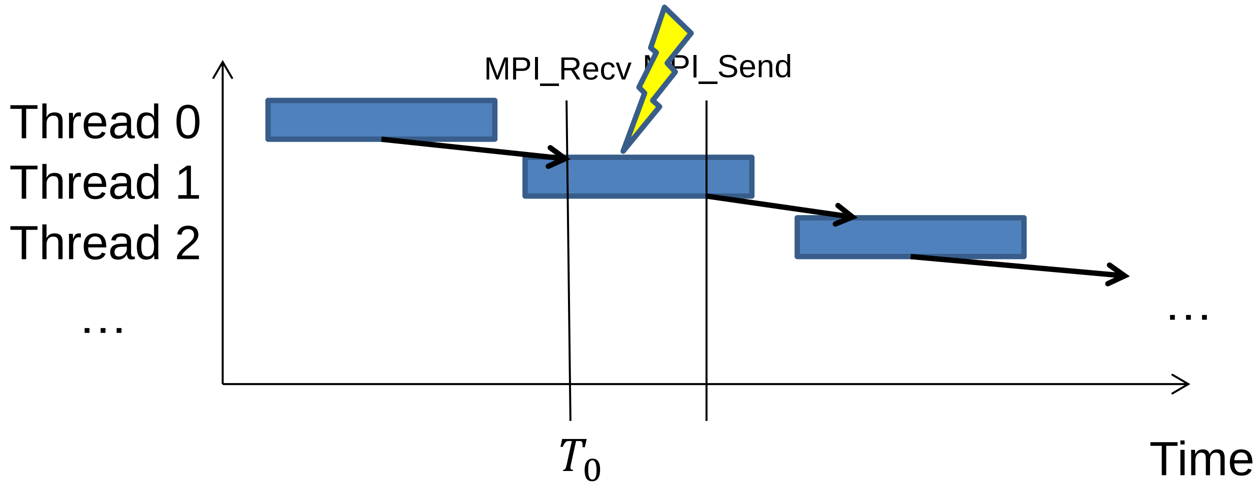
Impact of Memory DUE in MPI+OmpSs

- Remember, when we do not consider MPI, the checkpointing mechanism just has to:
  - Catch Exception
  - Restore Input Parameters
  - Re-execute
- Calling MPI inside the tasks makes things more problematic:



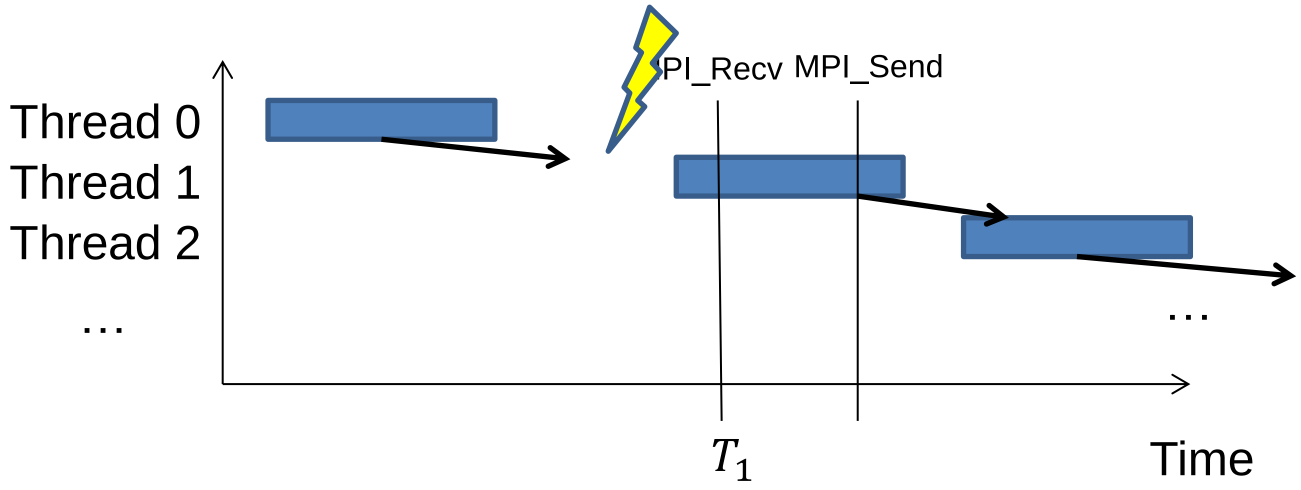
Impact of Memory DUE in MPI+OmpSs

- Remember, when we do not consider MPI, the checkpointing mechanism just has to:
  - Catch Exception
  - Restore Input Parameters
  - Re-execute
- Calling MPI inside the tasks makes things more problematic:



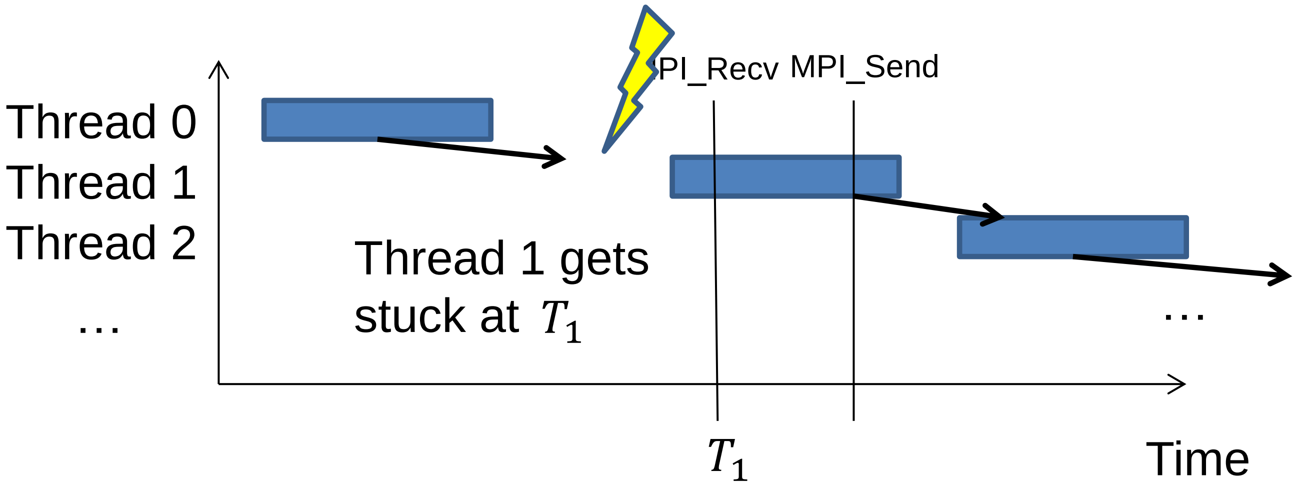
Impact of Memory DUE in MPI+OmpSs

- Remember, when we do not consider MPI, the checkpointing mechanism just has to:
  - Catch Exception
  - Restore Input Parameters
  - Re-execute
- Calling MPI inside the tasks makes things more problematic:



Impact of Memory DUE in MPI+OmpSs

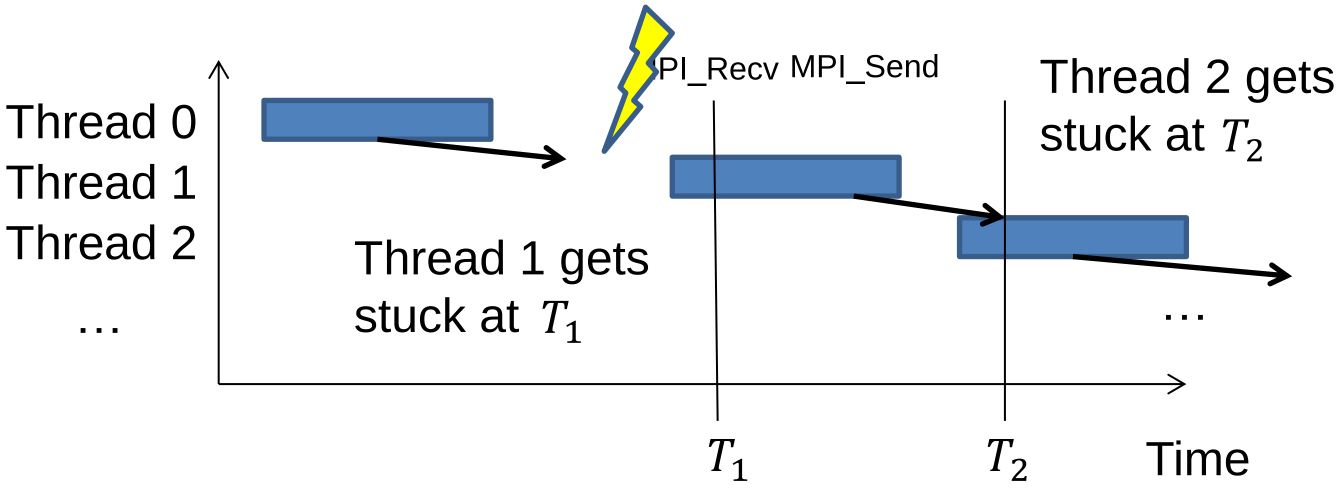
- Remember, when we do not consider MPI, the checkpointing mechanism just has to:
  - Catch Exception
  - Restore Input Parameters
  - Re-execute
- Calling MPI inside the tasks makes things more problematic:



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

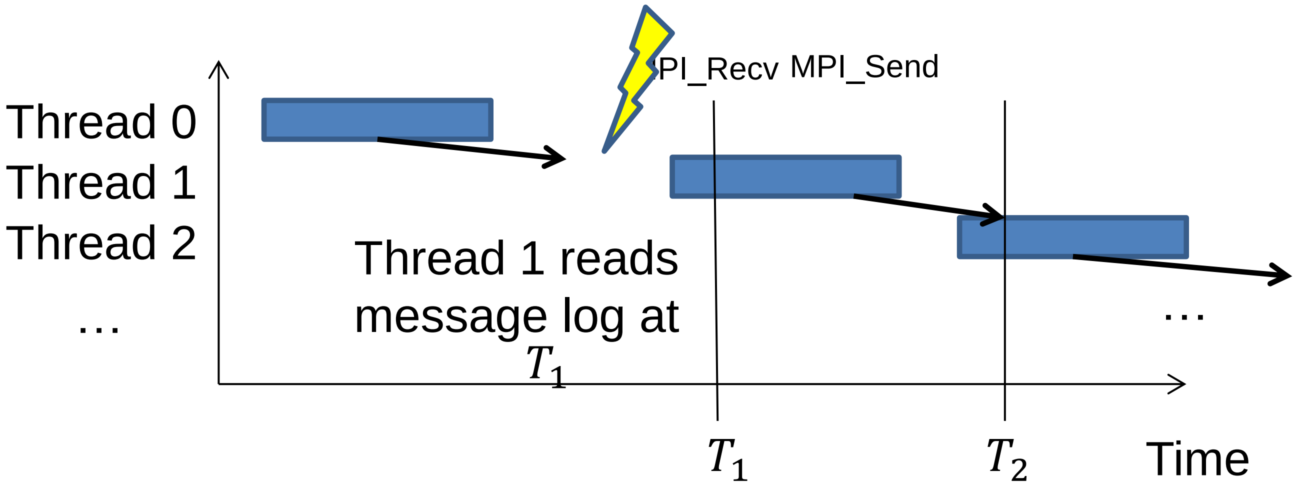
Impact of Memory DUE in MPI+OmpSs

- Remember, when we do not consider MPI, the checkpointing mechanism just has to:
  - Catch Exception
  - Restore Input Parameters
  - Re-execute
- Calling MPI inside the tasks makes things more problematic:



Solution: Message Logging

- Log incoming messages in per each task
- Task Log is deleted after task completion
- If the sequential order of incoming task is an invariant, it is straightforward to apply per-task message loging



Evaluation

	Checkpoint Size (MB)	Messages Logged (MB)	Messages Log Peak Size (MB)	Fault-free Overhead (%)
HPL	627,0	4268,0	34,0	7,3
Heat	129,0	5,0	0,5	1,0
Dense Matrix Multiply	512,0	8064,0	128,0	6,1
N-Body	0,2	0,8	0,3	0,3

Outline

- Introduction: HPC trends and Task-based Parallelism
- Motivation: Error Trends and Detection of Memory Errors
- Opportunities for Resilience Enabled by Task-based Parallelism:
  - Rollback Checkpointing/Restart Mechanisms
  - Linear Forward Recoveries for Iterative Solvers
- Resilience for Codes Combining MPI + Tasking
- Conclusions

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Conclusions

- Software Perspective:
  - Task Parallelism Enables Asynchronous Recoveries:
    - Forward ABFT techniques can effectively correct DUE’s in iterative solvers.
    - Rollback CheckPointing Strategies
  - Overlapping recoveries with computation:
    - Trivial to implement with OmpSs
    - Trade-off’s between overhead and fault coverage

- Hardware Perspective:
  - Precise memory ECC are required
  - How precise? Do we really need memory-page level detection accuracy?
  - Can ECC’s be relaxed?

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

QUESTIONS?