SMCV: a Methodology for Detecting Transient Faults in Multicore Clusters

Authors: Diego Montezanti, Fernando Emmanuel Frati, Dolores Rexachs, Emilio Luque, Marcelo Naiouf and Armando De Giusti

24 de Julio de 2012

UAB
Universitat Autònoma de Barcelona

caos

III-LIDI

Projects

- It takes part of the projects accredited by UNLP within the “Programa de Incentivos”:
  - “Arquitecturas multiprocesador distribuidas. Modelos, Software de Base y Aplicaciones.”
  - “Procesamiento paralelo y distribuido. Fundamentos y aplicaciones en Sistemas Inteligentes y Tratamiento de imágenes y video.”

Agenda

- Work context
- SMCV: proposed methodology
- Background
- SMCV description
- Experimental validation
- Future work

Work Context (I)

Modern Processors:

Challenge → Improving computation performance

Multicores – Chip Multiprocessors (CMP's)

Increasing integration scale - Associated factors:

Increasement of power density

Raise in internal operation temperature

Decrease in supply voltage

Consequences:

Interferences from the environment that affect processors

Transient fault vulnerability

Work Context (II)

Transient Fault:

- Affects some hardware component of the system
  - Physical location: main memory, register file, buses, I/O devices.
  - Some hardware components are protected (ECC´s, parity)
  - Critical: processor registers, logics
- Short-lived . Do not affect the regular operation of the system
- Do not occur exactly the same way never again
- Can temporarily invert one or several bits of the affected hardware component *(single or mutiple bit flip)*.

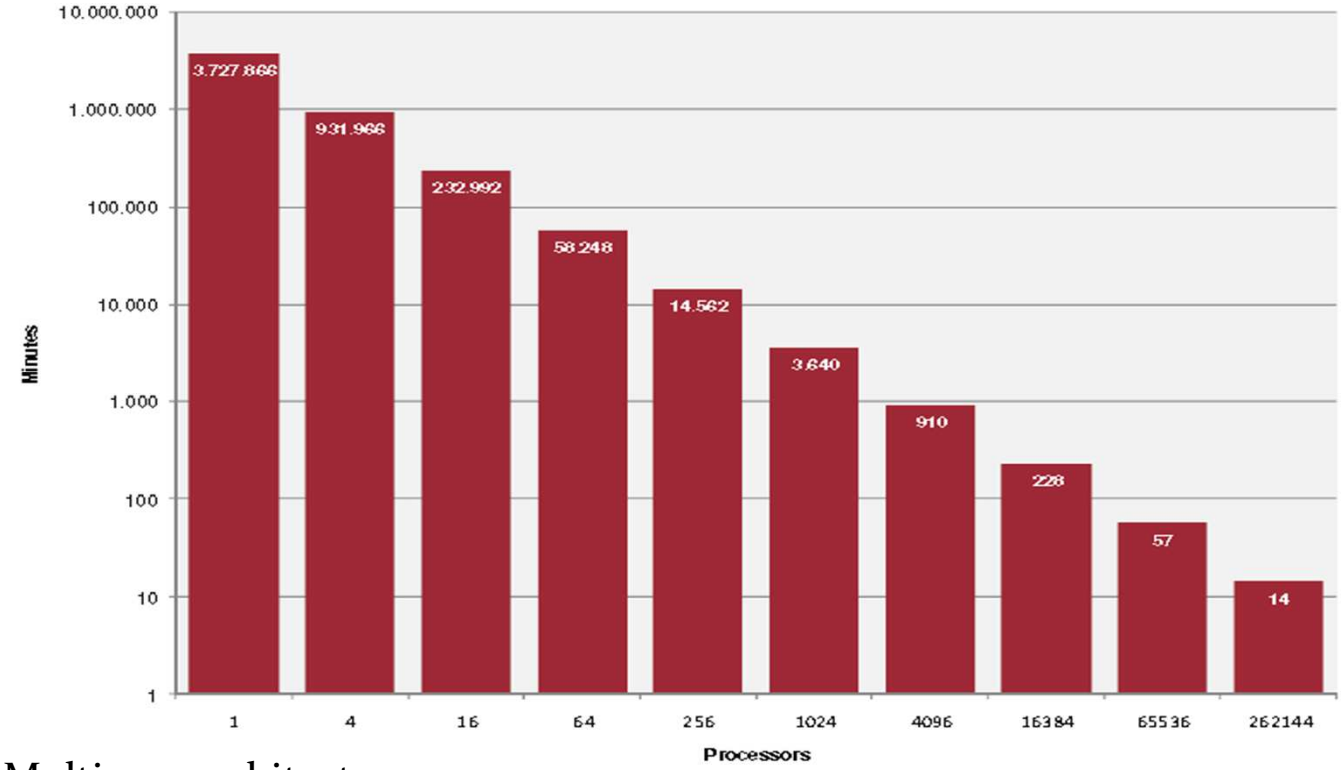
Application Perspective:

- Can alter important information (data, addresses, status information ,operation codes)
- May affect application behavior or results => **RELIABILITY**

Transient Faults in HPC systems

MTBF (commercial processor): about 2 years

Supercomputers: Frequent reports since 2000



Example IBM Power 4 (robust, monocore) hypothetical supercomputer MTBF [Mukherjee, S. et al. (2005)]

Multicore architectures

Large parallel applications ⟶ cost of relaunching

Strategies for reliability in HPC (detection first)

Universitat Autònoma de Barcelona

caos

III-LIDI

Techniques for Fault- Tolerance

Hardware-based

- Physical redundancy
- Most used in critical environments
- Inefficient in general-purpose computers

Software-based

- Low cost (achieved reliability vs involved resources)
- Flexibility, configuration
- Software redundancy
- Basic idea: DMR
- Workload (validation interval)

Serial programs

SMCV: proposed methodology

Detection strategy

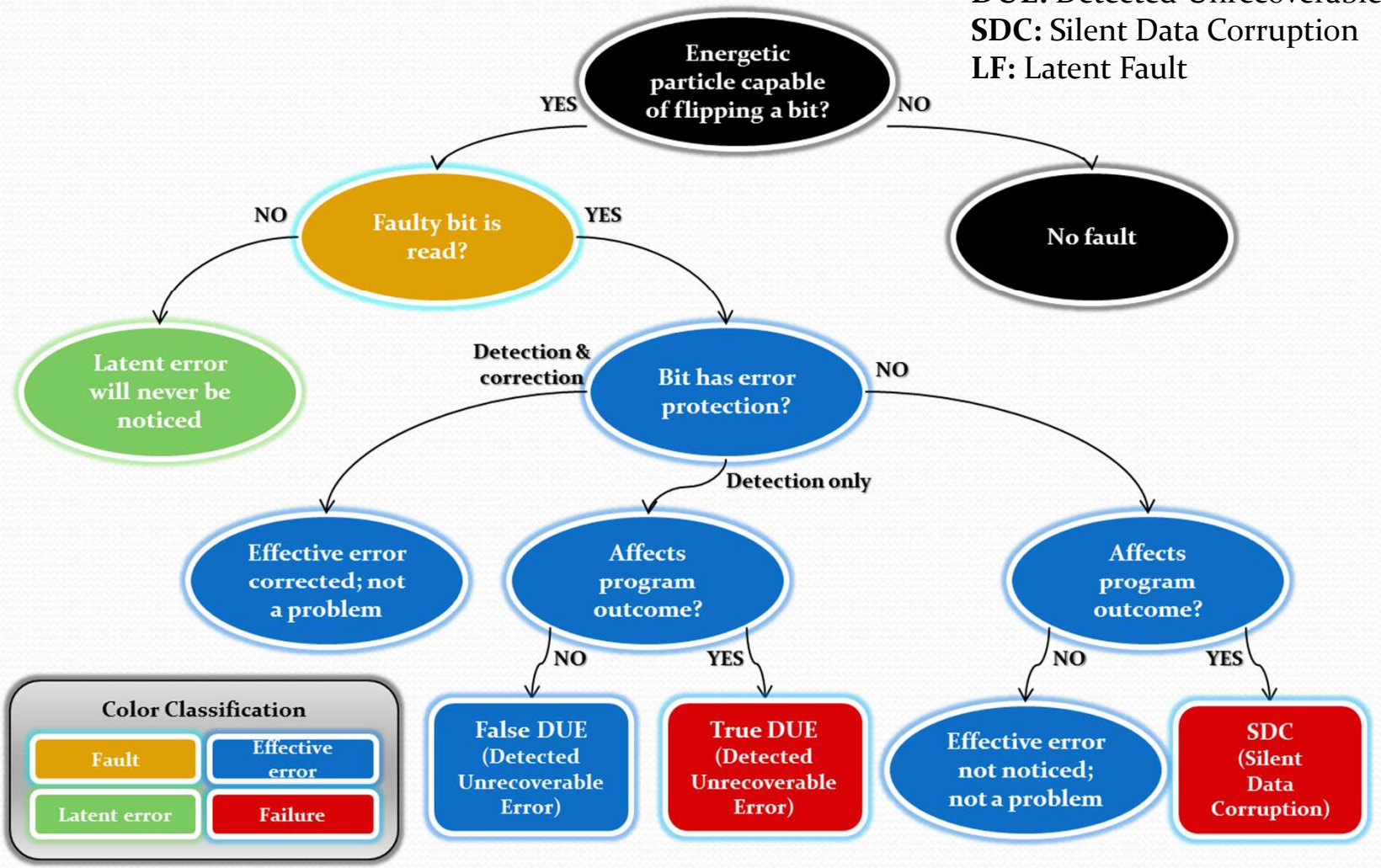
- ✓ Specific for scientific, message passing parallel applications
- ✓ Software only
- ✓ Leverages intrinsic redundancy in multicore clusters
- ✓ Validates contents of messages to be sent
  - ✓ Low overhead
  - ✓ Moderate validation interval
  - ✓ Reduced additional workload
- ✓ Fully distributed
- ✓ Prevents fault propagation to other application process

The application that finishes has correct results

Possible outcomes of a transient fault

SER = DUE + SDC + LF

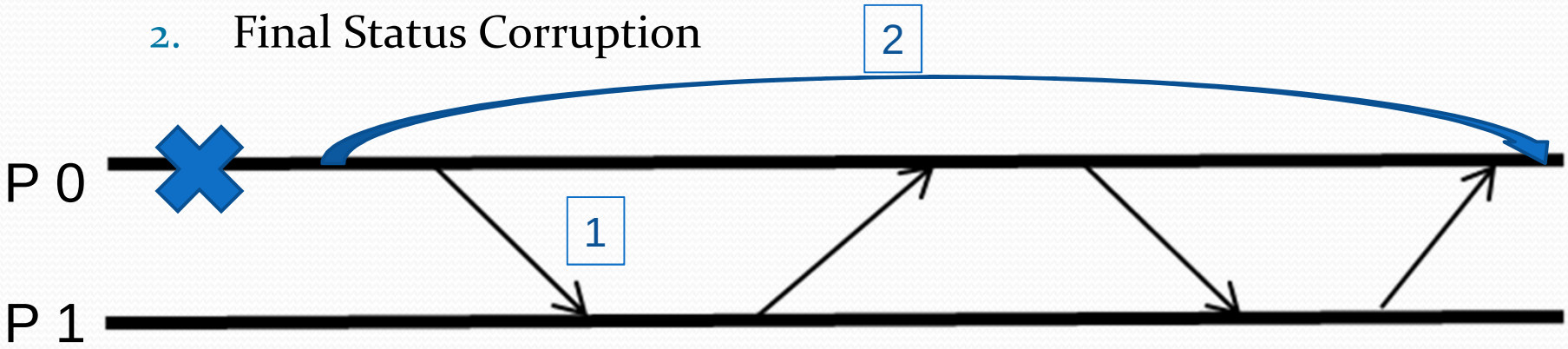
SER: Soft Error Rate
DUE: Detected Unrecoverable Error
SDC: Silent Data Corruption
LF: Latent Fault



Transient faults in message-passing parallel applications

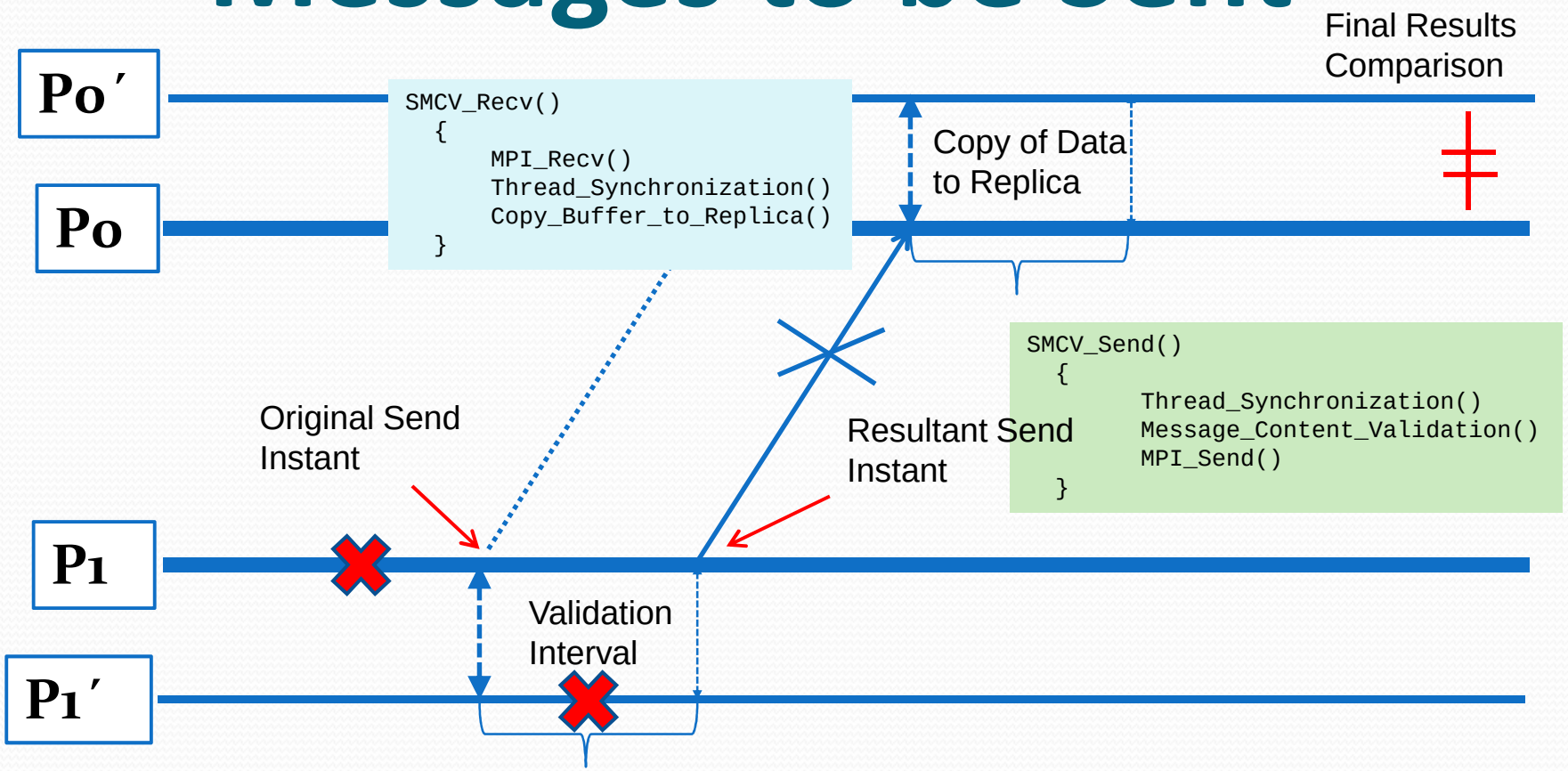
SDC// = TDC + FSC

1. Transmitted Data Corruption
2. Final Status Corruption



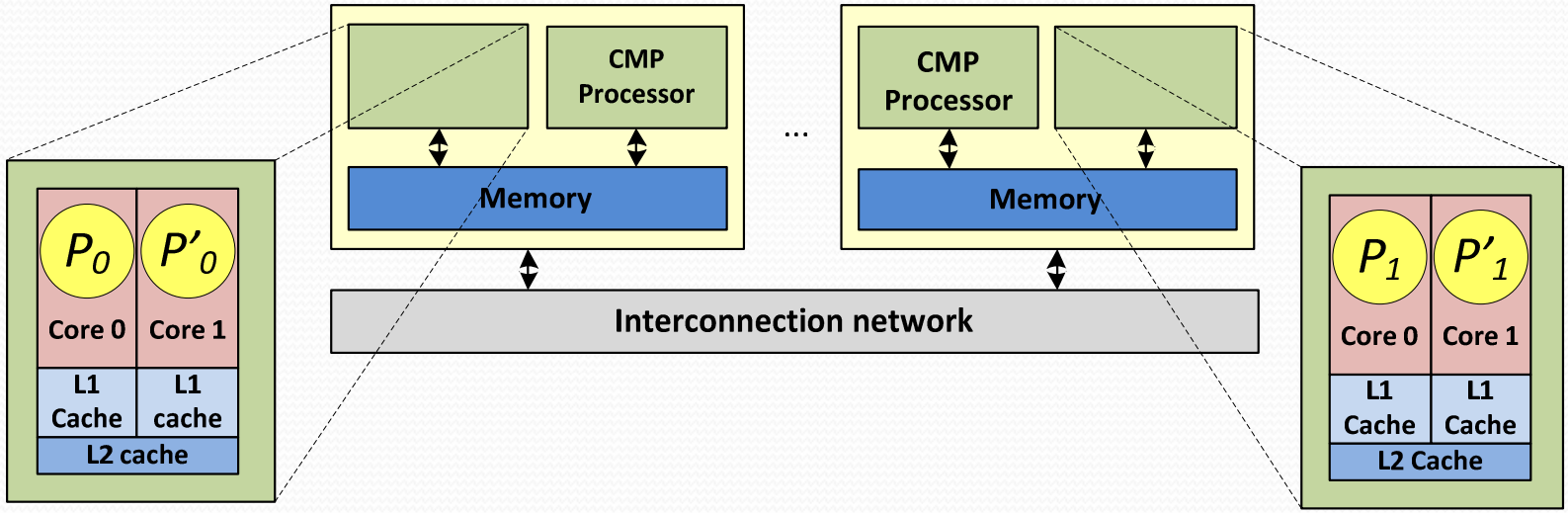
- Most SDC are TDC. They have a significant impact on end results
- FSC are similar to serial applications
- SMCV focuses in TDC portion and includes a final comparison for FSC

Validating Contents of Messages to be Sent



- **All faults that cause TDC are detected**
- **No fault is propagated to other application process**
- **Final comparison ensures correct results**
- **No additional network bandwidth is consumed**

Leveraging Redundant Hardware Resources



- Each application process is replicated in another core of the same CMP
- Half of the resources are used for redundancy
- The redundant core shares some cache level with the one that runs the application process
- Most comparisons are resolved at LLC, minimizing main memory access
- Using cores for FT is beneficial for the system

SMCV´s additional features

- Decentralized: each process and its replica are locally validated.
- Low overhead in execution time: only one comparison is added for each byte of each outgoing communication and the end result.
- Lightweight technique (versus a conservative strategy for sequential programs, in which each memory write operation is checked before being written).
- When a fault is detected, the application is safe-stopped, narrowing error latency (important in scientific applications that can run for several days).
- Trade-off between detection latency, additional workload and involved resources.

SMCV´s additional workload

Versus a conservative every-write validation technique (analyitical modeling)

$$\frac{W_{MV}}{W_{WV}} = \frac{M \,.\, C_{sync} + M \,.\, k \,.\, C_{comp}}{S \,.\, (C_{sync} + C_{comp}) + M \,.\, k \,.\, C_{sync} + M \,.\, k \,.\, C_{comp}} < 1$$

W_{MV}: workload introduced with message validation

W_{WV}: workload introduced with message validation

C_{sync}: cost of synchronization operation

C_{comp}: cost of comparison operation

S: store operations (writes), excluding those of messages

M: messages sent by the application

k: average size of a message

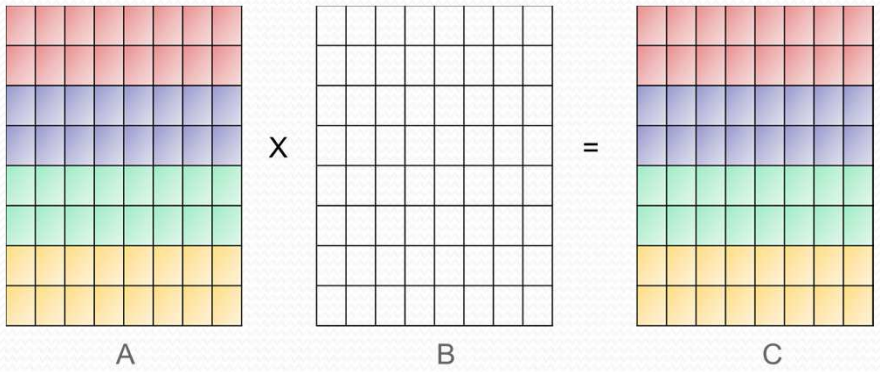
Experimental environment

Experimental platform:

- Cluster BLADE (LIDI - UNLP)
- 16 servers
- 2 Quad Core Intel Xeon 5405 @ 2.0 GHz, 2 x 6MB cache L2, shared between pairs of cores
- 2 GB RAM each server
- Fedora 12 (64 bits)
- OpenMPI library

Test application:

- Parallel matrix multiplication
- Master/Worker (Master also computes)
- Non-blocking messages



Two experiments:

1. **Detection efficacy**
2. **Overhead measurements**

Detection Efficacy

Setup of the environment:

- 4 application processes (Master and 3 Workers)
- Mapping: 4 application processes and 4 thread-based replicas to 8 cores of the blade

Fault Injection Experiments

- Debugging tool
- Breakpoint is inserted (any application process)
- Modification of a variable´s value
- Computation is resumed (simulates a transient fault)
- Both TDC and FSC were injected at certain instants during execution

✓ **TDC were detected by message content validation**
✓ **FSC were detected by final results comparison**

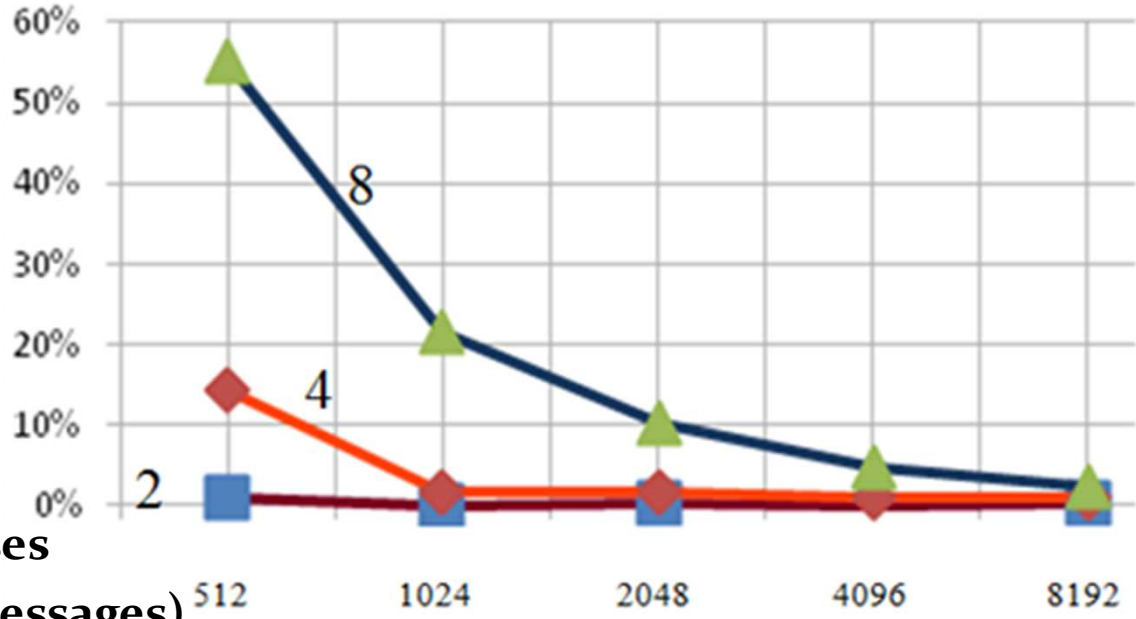
Overhead measurements

Differences in execution times determined in absence of faults

Setup of the environment:

- Application processes: 2, 4 and 8 (and its replicas)
- Matrix sizes: 512, 1024, 2048, 4096 and 8192
- Mapping that ensures the same conditions of execution with and without SMCV (2 blades for 8 processes)

	Processes		
Size (N)	2	4	8
512	0,87%	14,24%	55,11%
1024	0,01%	1,63%	21,40%
2048	0,39%	1,61%	10,05%
4096	-0,14%	0,91%	4,74%
8192	0,17%	0,92%	2,45%



- **With constant processes, it decreases with problem size.**
- **With constant size, it increases with amount of processes (messages).**

Future Work

General goal: providing transient fault tolerance for systems formed by scientific, message-passing parallel applications that are run on multicore cluster architectures.

Fault Tolerance = Detection + Protection + Recovery

- Integrating transient fault detection methodology to the protection and recovery strategies available for permanent faults to provide transient fault tolerance (checkpoint, log, rollback-recovery).
- No need of TMR + voting.

In this way...

1. Perfecting detection strategy
   - Expanding experimental validation: HPL, NAS
   - Integration with fault injection tools
   - Achieving transparency for the application (source-code required, thread-based)
   - Optimizing the methodology. Configuration of robustness level

2. Providing full tolerance to SDC
   - Integration with RADIC

Questions?

Thank you for your time!