Barcelona Supercomputing Center
Centro Nacional de Supercomputación

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Exploiting Task-Based Programming Models for Resilience

PhD defense 21/06/2019

Luc Jaulmes

Advisors: Marc Casas,
Miquel Moretó

Resilience trends

Error rates increase

- Hard faults: ageing, manufacturing variability
- Soft faults: particle strikes, voltage noise
- Memory subsystem most subject to faults

Known error rates

- Cielo supercomputer[1]: 1 error every 5h23min
- Mont-Blanc 1 prototype[2]: 1 error every 10min

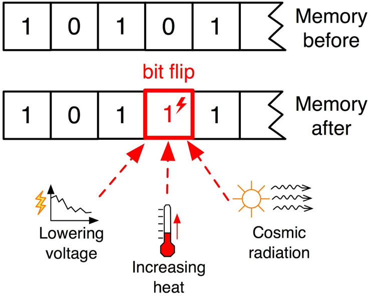


image dependablesystem.blogspot.com

[1] V. Sridharan et al. (2015). "Memory Errors in Modern Systems". In: ASPLOS XX, pp. 297–310.

[2] L. Bautista-Gomez et al. (2016). "Unprotected Computing". In: SC'16, 55:1–55–11.

Resilience trends

Main techniques today

- Sparing, adjusting refresh rate
  - Mitigate hard faults
  - Requires profiling
- Error Correcting Codes (ECC)
  - Detect unforeseen errors
  - Correct transparently
- Checkpointing-Rollback
  - Enable recovery from crash

Limitations

- Yield too low
- Current ECC unable to address projected fault rates
- High checkpoint overhead

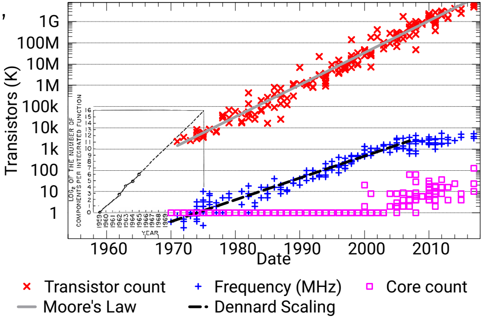
F. Cappello et al. (2014). "Towards Exascale Resilience: 2014 update". In: SuperFri 1.1, pp. 5–28.

Emergence of runtime systems

Programmability Wall

- Due to complex architectures

  E.g. large core counts, heterogeneous architectures, hybrid memory hierarchies, etc.
- Answer: new programming models
  - Take complexity away from programmer
  - Rely on supporting software: **the runtime system**



Runtime system role

- Initial focus: transparent task scheduling
- Opportunity for other optimizations

  E.g. accelerating critical tasks, enforcing a power budget, partitioning caches, managing scratchpad memories, etc.

Goal: exploiting runtime systems for cross-level resilience

- **Algorithm level (SC’15, TPDS 29:9)**
  - Novel forward recovery for a class of iterative solvers
  - Runtimes allow to overlap computation and recovery
- **OS level (Multiprog 2019, IOLTS’19)**
  - Define metric that improves correlation with error risk
  - Delay reporting errors to ignore non-consumed errors
- **Architectural level**
  - Sampling-based methodology to estimate vulnerability online
  - Implemented on real-world hardware with low overhead
  - Dynamically adjustable ECC guided allows to trade resilience for redundancy

State of the Art: Algorithmic Recoveries

Checkpointing / Rollback

A. Moody et al. (2010). "Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System". In: SC'10, pp. 1–11

L. Bautista-Gomez et al. (2011). "FTI: High performance Fault Tolerance Interface for hybrid systems". In: SC'11, 32:1–32:12

M. Bougeret et al. (2011). "Checkpointing strategies for parallel jobs". In: SC'11, 33:1–33:11

Algorithm-specific

- Restart methods

J. Langou et al. (2007). "Recovery Patterns for Iterative Methods". In: SIAM J. Sci. Comput. 30.1, pp. 102–116

E. Agullo et al. (2016). "Numerical recovery strategies for parallel resilient Krylov linear solvers". In: Numer. Linear Algebra Appl. 23.5, pp. 888–905

- Check invariants

Z. Chen (2013). "Online-ABFT". In: PPoPP'13, pp. 167–176

State of the Art: Vulnerability Metrics

Existing Metrics

- AVF, specialised for memory
- DVF
- *LD*/*ST*

S. S. Mukherjee et al. (2003). "A Systematic Methodology to Compute the Architectural Vulnerability Factors". In: MICRO 36, pp. 29–42
Y. Luo et al. (2014). "Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost". In: DSN 2014, pp. 467–478

L. Yu et al. (2014). "Quantitatively Modeling Application Resilience with the Data Vulnerability Factor". In: SC'14, pp. 695–706

M. Gupta et al. (2018). "Reliability-Aware Data Placement for Heterogeneous Memory Architecture". In: HPCA 2018, pp. 583–595

Error injection studies

G. Bronevetsky and B. R. de Supinski (2008). "Soft error vulnerability of iterative linear algebra methods". In: ICS'08, pp. 155–164
M. Casas et al. (2012). "Fault resilience of the algebraic multi-grid solver". In: ICS'12, pp. 91–100

Applications

- Reliability-driven memory mapping
- Selective protection (ABFT, pointer triplication, . . .)

State of the Art: Dynamically Adaptable ECC

- **Two-level ECCs for DRAM**
  - Static: VS-ECC, Odd-ECC

D. H. Yoon and M. Erez (2010). “Virtualized and Flexible ECC for Main Memory”. In: ASPLOS XV, pp. 397–408
A. Malek et al. (2017). “Odd-ECC”. In: MEMSYS’17, pp. 96–111

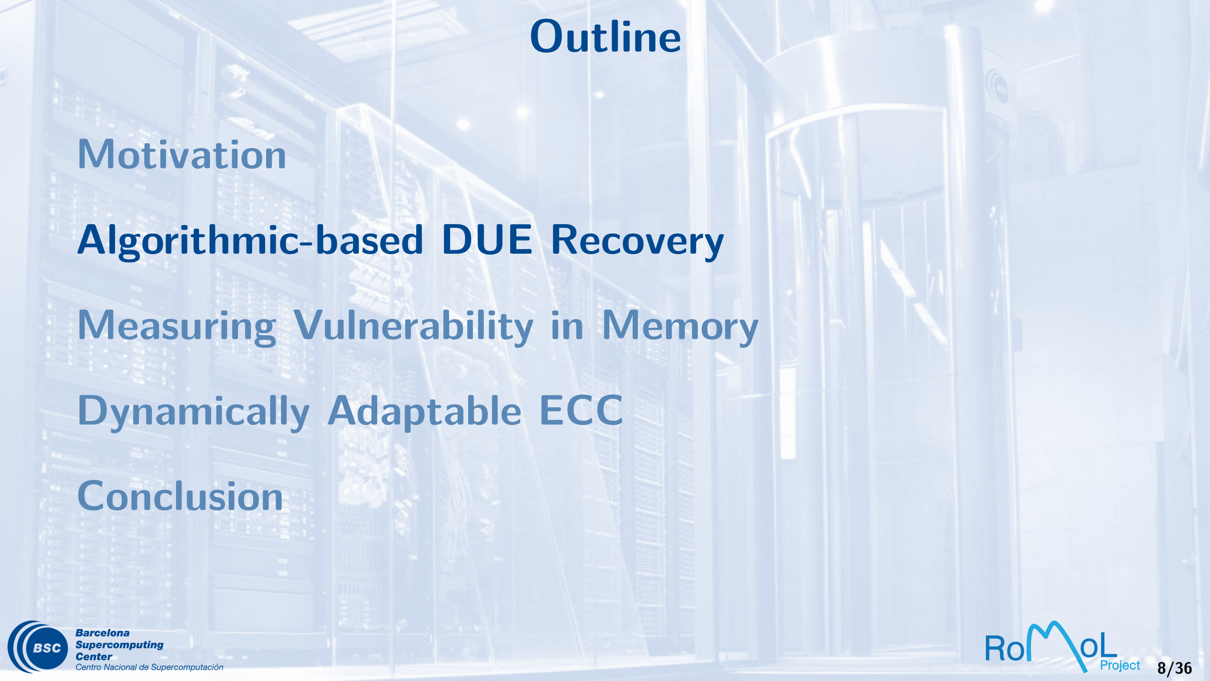
- **Sampling-based memory access patterns analysis**
  - Offline: bottleneck analysis. . .

Alfredo Giménez et al. (2014). “Dissecting On-node Memory Access Performance”. In: SC’14, pp. 166–176
H. Servat et al. (2014). “Identifying Code Phases Using Piece-Wise Linear Regressions”. In: IPDPS, pp. 941–951

  - Online: miss rate curves on POWER5

D. K. Tam et al. (2009). “RapidMRC”. In: ASPLOS XIV, pp. 121–132

Outline



Motivation

Algorithmic-based DUE Recovery

Measuring Vulnerability in Memory

Dynamically Adaptable ECC

Conclusion

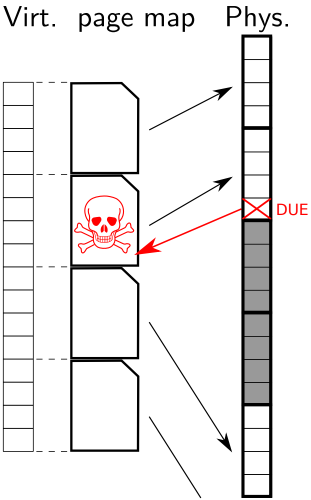
Error model: memory DUE

Fault in memory

- ECC performs detection, attempts correction
- Detected Uncorrected Errors (DUE) may happen
  - 27% of node down events caused by DUE[1]
- Reported in register: machine check exception

Application-level symptom

- Access DUE location ⇒ OS kills application
- Replace physical page: **data lost!**



[1] S. Levy et al. (2018). "Lessons Learned from Memory Errors Observed over the Lifetime of Cielo". In: SC'18, 43:1–43:12.

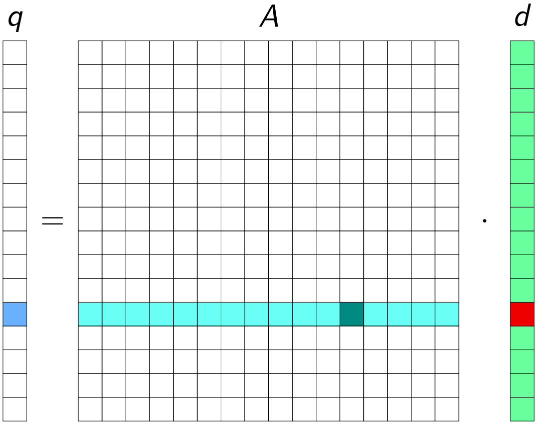
Extract redundancy from operations

Rewrite: 1 block per memory page

- Linear combination: $u_i = \alpha v_i + \beta w_i$
- Matrix-vector mult: $q_i = \sum_j A_{ij} d_j$
- 4KB page ⇒ 512 doubles

Reuse or “invert” relations

- $v_i = \frac{1}{\alpha}(u_i - \beta w_i)$
- Factorize diagonal blocks:
  $A_{ii} d_i = q_i - \sum_{j \neq i} A_{ij} d_j$



Fine-grain algorithm-based error correction for CG

```
$d' \Leftarrow 0$
$g \Leftarrow b - Ax$                        initial $x$
for $t$ in $0..t_{max}$
    $\epsilon \Leftarrow ||g||^2$                    $g = b - Ax$
    if $\epsilon < tol$ :
        break
    $\beta \Leftarrow \epsilon \ / \ \epsilon_{old}$
    $d \Leftarrow \beta d' + g$                  $d' = A^{-1}q$      $g = b - Ax$
    $q \Leftarrow Ad$                       $d = \beta d' + g$
    $\alpha \Leftarrow \epsilon \ / < q, d >$           $q = Ad$            $d = A^{-1}q$
    $x \Leftarrow x + \alpha d$                  $d = A^{-1}q$       $x = A^{-1}(b - g)$
    $g \Leftarrow g - \alpha q$                  $q = Ad$            $g = b - Ax$
    $\epsilon_{old} \Leftarrow \epsilon$
    swap($d$, $d'$)
```

colors: constant, dynamic

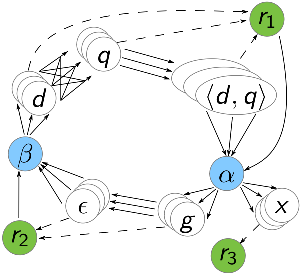
Recover lost data

- Recompute from redundancy existing in algorithm
  - Recompute past operations
  - Identify invariants, e.g. $g = b - Ax$
- No overhead for adding redundancy
- *Exact recovery*: converge as well as without faults
- *Forward recovery*: no work reverted

Fully covers Krylov subspace iterative solvers

- Popular family of solvers: CG, GMRES, BiCGStab...
- Also works with preconditioners

Task-based CG implementation



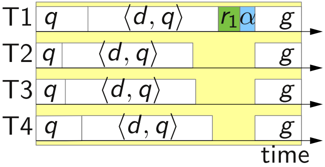
OmpSs

- Asynchronous tasks
- Dataflow dependencies

Adding recoveries

- Skip “corrupted” computations
- Recovery computations in tasks
  - before scalar tasks

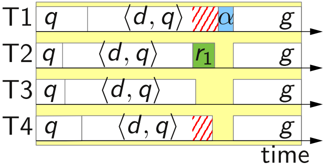
Strategies to schedule recoveries



Critical path

- Conservative approach

“Forward Exact Interpolation Recovery” (FEIR)



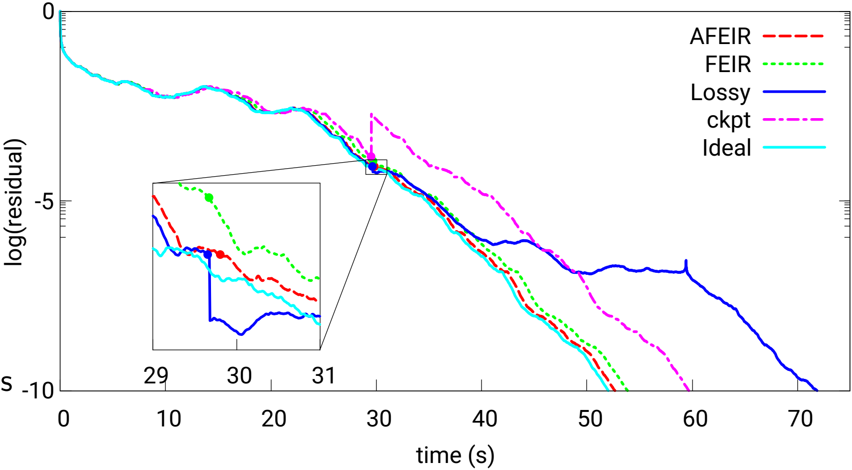
Overlap with work

- Reduce overhead
- Sacrifice error coverage

“Asynchronous FEIR” (AFEIR)

Compared methods

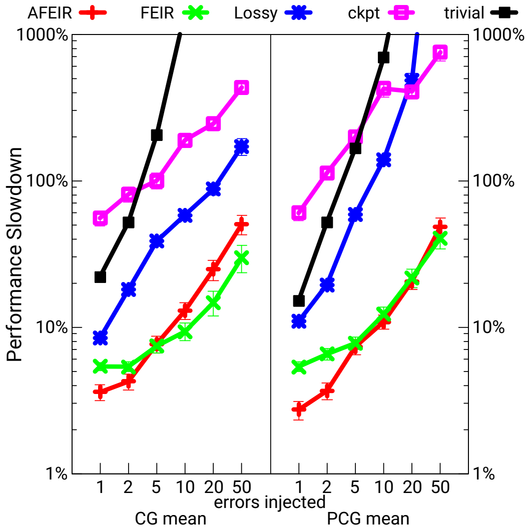
- **Ideal**
  - No faults baseline
- **Trivial**
  - No recovery, erratic
- **Checkpoint-Rollback**
  - Exact, backwards
- **Lossy Restart**[1][2]
  - Forward, lose some convergence guarantees
- **FEIR**
- **AFEIR**



[1] J. Langou et al. (2007). “Recovery Patterns for Iterative Methods”. In: SIAM J. Sci. Comput. 30.1, pp. 102–116.

[2] E. Agullo et al. (2016). “Numerical recovery strategies for parallel resilient Krylov linear solvers”. In: Numer. Linear Algebra Appl. 23.5, pp. 888–905.

Results with fault injection



Methodology

- Error injection: trigger signal on access
  - For application, strictly identical
  - MTBE scaled to baseline's run time
- Single 8-core socket
- Geometric mean of overheads

CG and PCG evaluated

- 9 matrices[1]
- Block-Jacobi preconditioner

[1] T. A. Davis and Y. Hu (2011). "The University of Florida Sparse Matrix Collection". In: ACM Trans. Math. Softw. 38.1, 1:1–1:25.

Summary

Algorithm level recovery

- Fine-grain recovery
- Desirable properties: Forward & Exact
- Uses inherent redundancy, easy to identify

Runtimes allow to overlap computation and recovery

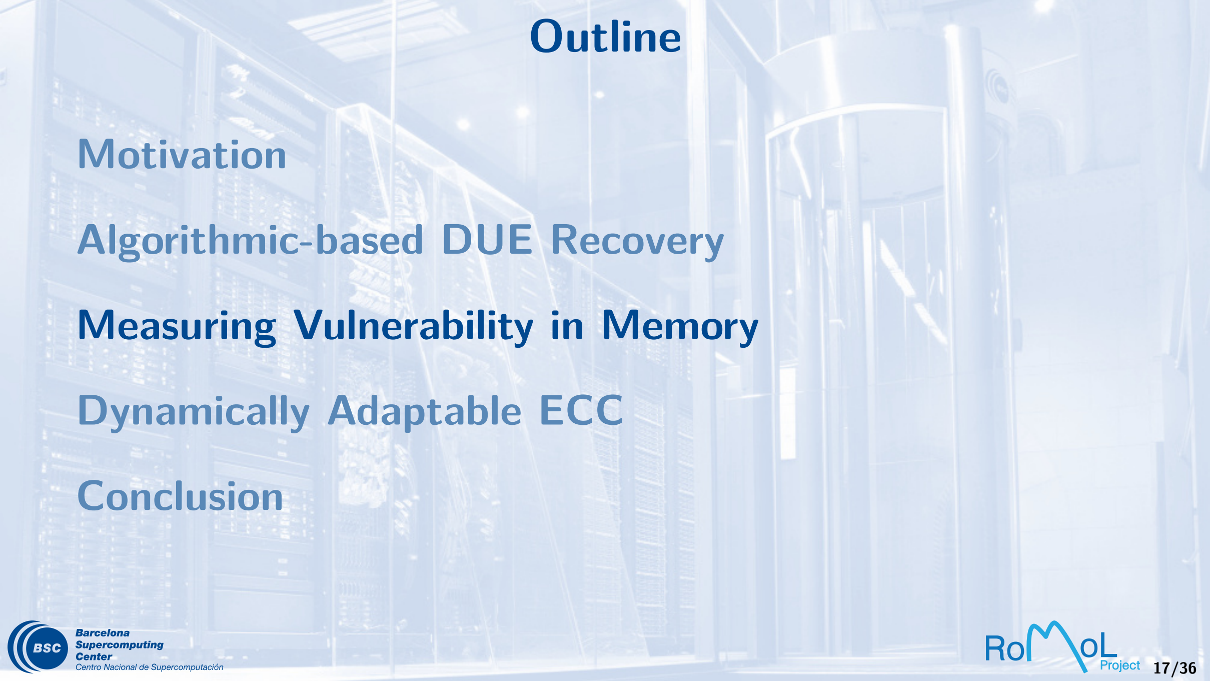
- Mask overhead
- Trade-off with error coverage

Further results

- Redundancy relations used for BiCGStab, GMRES, preconditioned solvers
- Strong scaling (up to 1024 cores)
- Impact of memory page size

L. Jaulmes et al. (2015). "Exploiting Asynchrony from Exact Forward Recovery for DUE in Iterative Solvers". In: SC'15, 53:1–53:12

L. Jaulmes et al. (2018). "Asynchronous and Exact Forward Recovery for Detected Errors in Iterative Solvers". In: IEEE Trans. Parallel Distrib. Syst. 29.9, pp. 1961–1974



Outline

Motivation

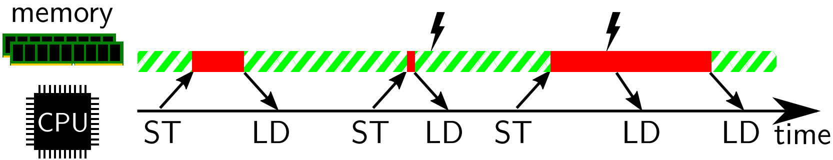
Algorithmic-based DUE Recovery

Measuring Vulnerability in Memory

Dynamically Adaptable ECC

Conclusion

Intuition: faults only matter if consumed



Memory Vulnerability Factor (MVF)

- Architectural Vulnerability Factor[1] for memory
- Error overwritten by ST

  ST ⇒ data safe

  LD ⇒ data vulnerable
- MVF = fraction of vulnerable time (before LD)
- Upper bound on failure probability

[1] S. S. Mukherjee et al. (2003). "A Systematic Methodology to Compute the Architectural Vulnerability Factors". In: MICRO 36, pp. 29–42.

Errors in data fetched from memory, but not consumed

Data fetched but *unused*

- Loaded speculatively
- Neighbour data (e.g. same cache line)
- Write-allocate cache on ST miss

Errors have no impact on the program

- Reporting them causes *False Errors*
- Solution: delay until data consumed

New Metric: False-Error Aware vulnerability (FEA)

- Data vulnerable only if consumed:

$$FEA = \frac{\text{time before data consumed}}{\text{total time}} = MVF - \frac{\text{time before ``unused fetch''}}{\text{total time}}$$

- Still upper bound on failure probability

Vulnerability metrics in literature

Safe Ratio[3]

$$sr = \frac{\text{safe time}}{\text{total time}} = 1 - MVF$$

Store ratio (proxy for MVF)[4]

As $\frac{LD}{LD+ST}$ to fit in $[0,1]$ and correlate positively

Data Vulnerability Factor[5]

$$DVF = \sum_{d \in \text{data structures}} \text{error rate} \times \text{size}_d \times \text{execution time} \times \text{memory accesses}_d$$

- Detailed model for memory accesses, based on access pattern, cache sizes.

[3] Y. Luo et al. (2014). “Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost”. In: DSN 2014, pp. 467–478

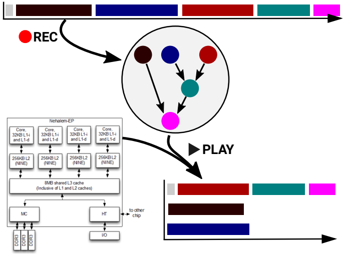
[4] M. Gupta et al. (2018). “Reliability-Aware Data Placement for Heterogeneous Memory Architecture”. In: HPCA 2018, pp. 583–595

[5] L. Yu et al. (2014). “Quantitatively Modeling Application Resilience with the Data Vulnerability Factor”. In: SC’14, pp. 695–706

Measure metrics and impact of errors

TaskSim[1] to measure vulnerability

- Architectural simulator
  - Simple core model
  - Full cache hierarchy
  - Ramulator[2] for memory
- Uses real runtime
  - Schedule tasks onto simulated cores
  - Simulate tasks in detail
- Based on task traces
  - Traced with DynamoRIO
  - Records memory accesses, basic blocks, runtime calls



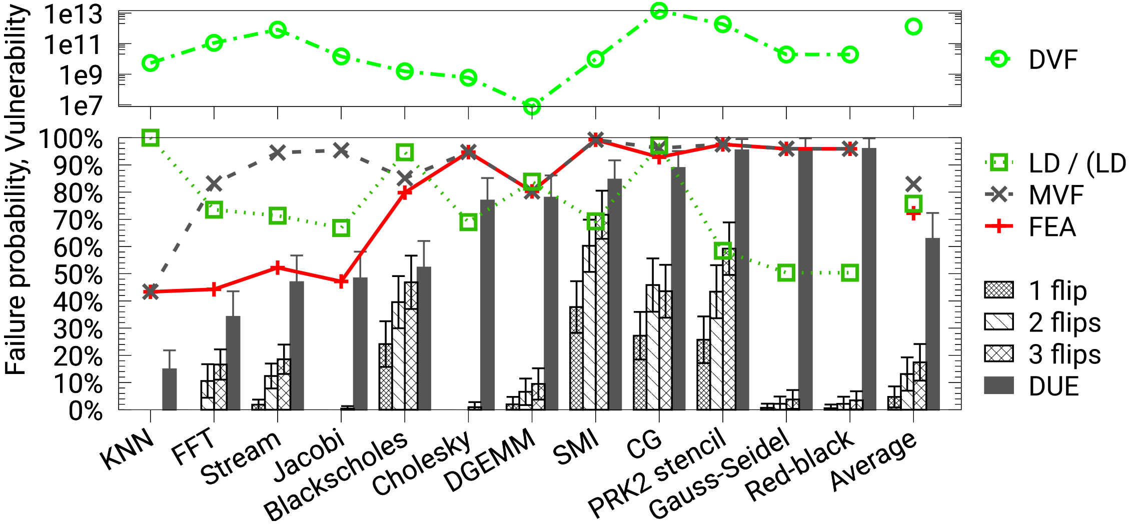
Native runs to measure impact of errors

- Flip 1, 2, 3 bits or DUE (e.g. `NaN`)
- outcome **crash, hang, wrong, slow** or **ok**
- On Intel Xeon Platinum 8160

[1] A. Rico et al. (2011). “Trace-driven simulation of multithreaded applications”. In: ISPASS, pp. 87–96.

[2] Y. Kim et al. (2016). “Ramulator: A Fast and Extensible DRAM Simulator”. In: IEEE Comput. Archit. Lett. 15.1, pp. 45–49.

Comparing metrics over various benchmarks



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Failure = **crash + hang + wrong + slow** outcomes, success = **ok** outcomes

RoMoL Project

Summary

False-Error Aware metric

- Identify errors that will not be consumed
- Best metric correlation with error risk
- Remains upper bound on error risk

Further results

- Metric comparison at memory page granularity
- Correlation coefficients
- DRAM refresh savings

L. Jaulmes et al. (2019b). “Memory Vulnerability: A Case for Delaying Error Reporting”. In: Multiprog 2019

L. Jaulmes et al. (2019a). “Memory Vulnerability for ECC-protected Memory”. In: IOLTS 2019

Outline

Motivation

Algorithmic-based DUE Recovery

Measuring Vulnerability in Memory

Dynamically Adaptable ECC

Conclusion

Dynamically adapting ECC online

- **Estimate FEA metric online: a methodology**
  - Identify memory access patterns in real time
  - Compute fraction of time before LD
  - Uses sampling of instructions

- **Dynamically adjustable ECC scheme**
  - ECC scheme with 2 protection levels
  - Increase protection for pages identified as most vulnerable

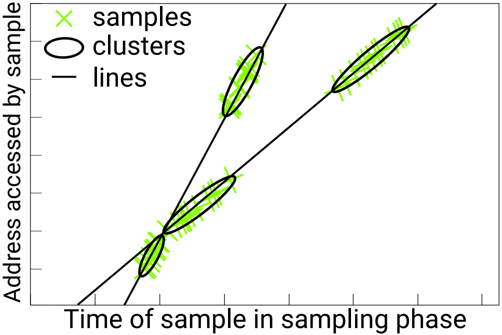
Runtime vulnerability methodology

Gather information from hardware sampling

- Sample LD and ST target address and time
  - PMU randomly selects instructions
  - Record 1 in N selected instructions
- Only during “sampling phases”
  - Control overhead: 0 cost when disabled

Extrapolate memory access pattern from sampled instructions

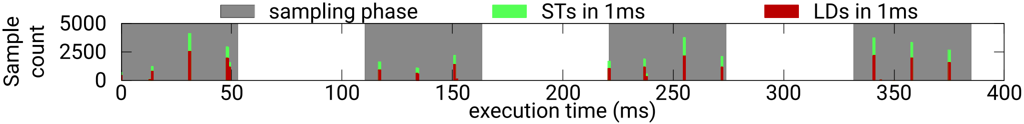
- Aligned points ⇒ streaming patterns
  - Kernel Hough Transform[1]
- Extrapolate streams to memory region
- Average time before LD as vulnerability



[1] L. A. F. Fernandes and M. M. Oliveira (2008). “Real-time line detection through an improved Hough transform”. In: Pattern Recog. 41.1, pp. 299–314.

Implemented on Power8

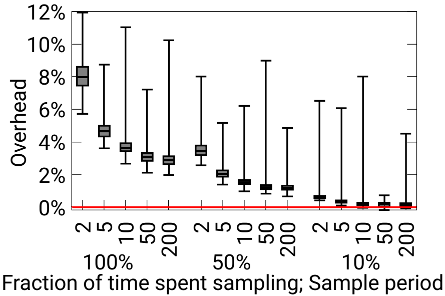
“Randomly” selected instructions



User-level interrupt to record

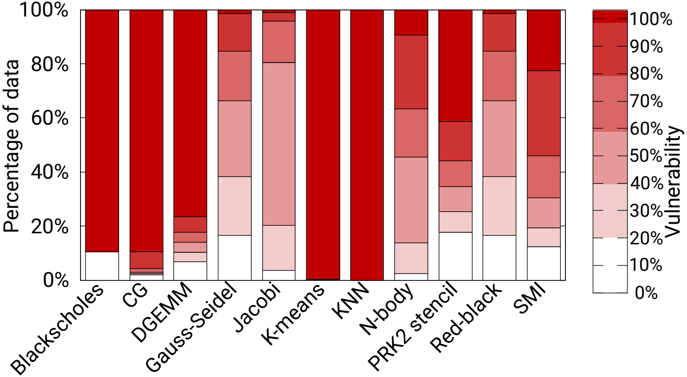
- Hand-written assembly routine
- Log to thread-local buffers

Technique overhead: sampling + analysis



Sampling period of 2, enabled 50% of the time: 3.47% overhead

Online vulnerability values



Varied Vulnerability Behaviour

- To be used for stronger protection targeting

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

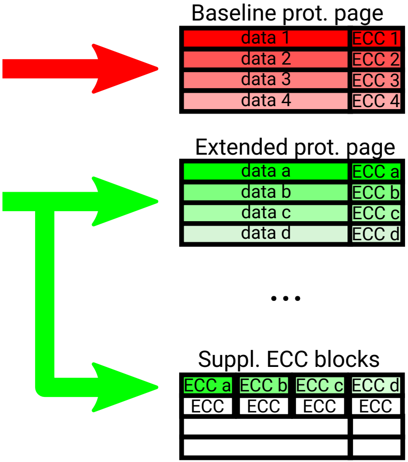
Witsec Is Targeted Strong Error Correction

Core ideas

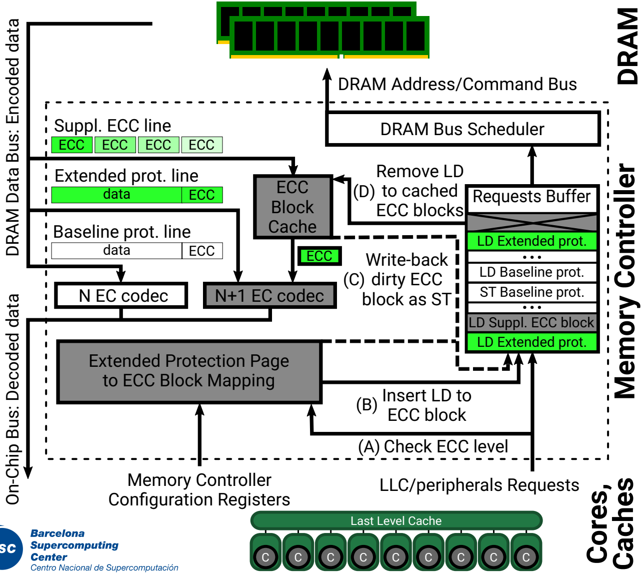
- Apply strong ECC where needed
  - Trade-off reliability vs. redundancy
- Dynamically switch normal ↔ strong ECC
- Extra redundancy in addressable memory

ECC Model

- Baseline: N bit flips ($0 \leq N < 3$)
- Selected pages: N+1 bit flips



WITSEC implementable in the memory controller



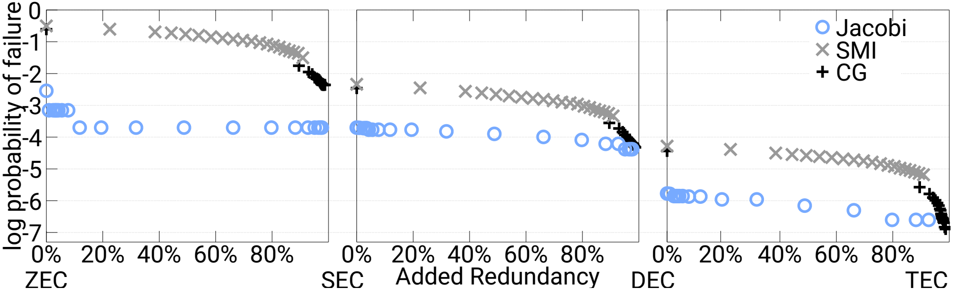
Necessary extensions

- Strong (N+1) EC codec
- Extended-protection pages
  - Physical address, size
  - Corresponding supplementary ECC blocks

Supplementary extension

- ECC blocks cache

Evaluation: trade-off reliability vs. redundancy



Measurement

- Inject single, double and triple bit flips
  - ▶ Vulnerability at injection time
  - ▶ Outcome at the end of the run
- Failure probability per vulnerability threshold:
  - ▶ Injection target above threshold ⇒ **ok**
  - ▶ Otherwise use outcome: **failure** or **ok**

Trade-off applications

- Given a global reliability target, apply optimal level of redundancy
- Given a redundancy budget, get maximal reliability

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RoMoL Project

Summary

Sampling-based methodology to estimate vulnerability online

- Generic: only requires sampling capability
- Uses real-time algorithms to identify streaming memory access patterns

Evaluated on real-world hardware

- Using POWER8 performance monitoring unit
- Low overhead: 3.47%

WITSEC dynamically adjustable ECC

- Implementable in memory controller
- Allows to dynamically trade resilience for redundancy

Further results

- CPU vs Memory Vulnerability Comparison
- Event-Based Branch details

L. Jaulmes et al. (est. 2020). *Adapting ECC Protection Dynamically using Online Estimation of Memory Vulnerability*. under preparation

L. Jaulmes (2018). *Online sampling-based vulnerability estimator*. GitHub. URL: `https://github.com/lucjaulmes/online_vulnerability`

Outline

Motivation

Algorithmic-based DUE Recovery

Measuring Vulnerability in Memory

Dynamically Adaptable ECC

Conclusion

Conclusion

Fine-grain algorithm-level recovery techniques

- Take full advantage of OS and HW support for DUE
- Overlap work and recover: never stop solving

False-Error Aware Metric

- Consistent upper bound, correlates best with failure risk
- Highlights opportunities from dead data in memory

WITSEC

- Dynamically adjust ECC to protect more the most vulnerable regions
- Detect most vulnerable regions online using sampling-based instrumentation

**Runtime systems help optimise recoveries,
detect redundant data, and manage added redundancy.**

Where to go from here

Extend existing work

- More algorithms, access patterns, different hardware. . .
- Using task dependency graphs for vulnerability estimation

Use “dead data” insight from FEA

- Skip DRAM refreshes, critical for future DRAM technologies
- Candidates for algorithmic optimisation, e.g. loop fusion for stencil codes

Runtime-aided error detection

- High DUE tolerance ⇒ lower SDC rate
- Data dependency information, task profiling
  ⇒ detect and contain errors on shared memory systems
- Programming-model support to express resilience

List of publications

First author publications:

- **L. Jaulmes**, M. Casas, M. Moretó, E. Ayguadé, J. Labarta, and M. Valero (2015). “Exploiting Asynchrony from Exact Forward Recovery for DUE in Iterative Solvers”. In: SC’15, 53:1–53:12. **Nominated for the best paper award.**
- **L. Jaulmes**, M. Moretó, E. Ayguadé, J. Labarta, M. Valero, and M. Casas (2018). “Asynchronous and Exact Forward Recovery for Detected Errors in Iterative Solvers”. In: IEEE Trans. Parallel Distrib. Syst. 29.9, pp. 1961–1974
- **L. Jaulmes**, M. Moretó, M. Valero, and M. Casas (2019b). “Memory Vulnerability: A Case for Delaying Error Reporting”. In: Multiprog 2019
- **L. Jaulmes**, M. Moretó, M. Valero, and M. Casas (2019a). “Memory Vulnerability for ECC-protected Memory”. In: IOLTS 2019

Other publications:

- M. Casas, M. Moreto, L. Alvarez, E. Castillo, D. Chasapis, T. Hayes, **L. Jaulmes**, O. Palomar, O. Unsal, A. Cristal, Eduard Ayguadé, J. Labarta, and M. Valero (2015). “Runtime-Aware Architectures”. In: Euro-Par 2015, pp. 16–27
- D. Richards and **L. Jaulmes** (2014). “CoMD in Chapel: The Good, the Bad, and the Ugly”. In: Chapel Lightning Talks, Birds-of-a-Feather session at SC’14

Publicly available code:

- **Luc Jaulmes** (2016). *Resilient CG implementation*. GitHub. URL: `https://github.com/lucjaulmes/resilient_cg` (visited on 01/23/2019)
- **L. Jaulmes** (2018). *Online sampling-based vulnerability estimator*. GitHub. URL: `https://github.com/lucjaulmes/online_vulnerability` (visited on 11/01/2018)
- **Luc Jaulmes** (2019). *OmpSs Fault Tolerance Benchmarks*. GitHub. URL: `https://github.com/lucjaulmes/ompss_fault_tolerance_benchmarks` (visited on 01/23/2019)