Parallelizing Online Error Detection in Many-core Microprocessor Architectures

Manolis Kaliorakis [1], Mihalis Psarakis [2], Nikos Foutris [1], Dimitris Gizopoulos [1]

[1] Dept. of Informatics & Telecomm., University of Athens, Greece
[2] Dept. of Informatics, University of Piraeus, Greece

Joint Euro-TM/MEDIAN Workshop on Dependable Multicore and Transactional Memory Systems (DMTM), Vienna, Austria, January 22, 2014

Motivation

- Many-core microprocessor era
  - deeper nanometer-scale technologies
  - extreme design complexity

  → unreliable many-core microprocessor chips

- Software-based testing:
  - a low-cost online error detection solution
  - must be accelerated to avoid scaling with the number of cores
  - <u>not</u> a conventional parallel programming problem
    - must be applied individually in *every* core

- Can be an online test program parallelized for many-core architectures?

Outline

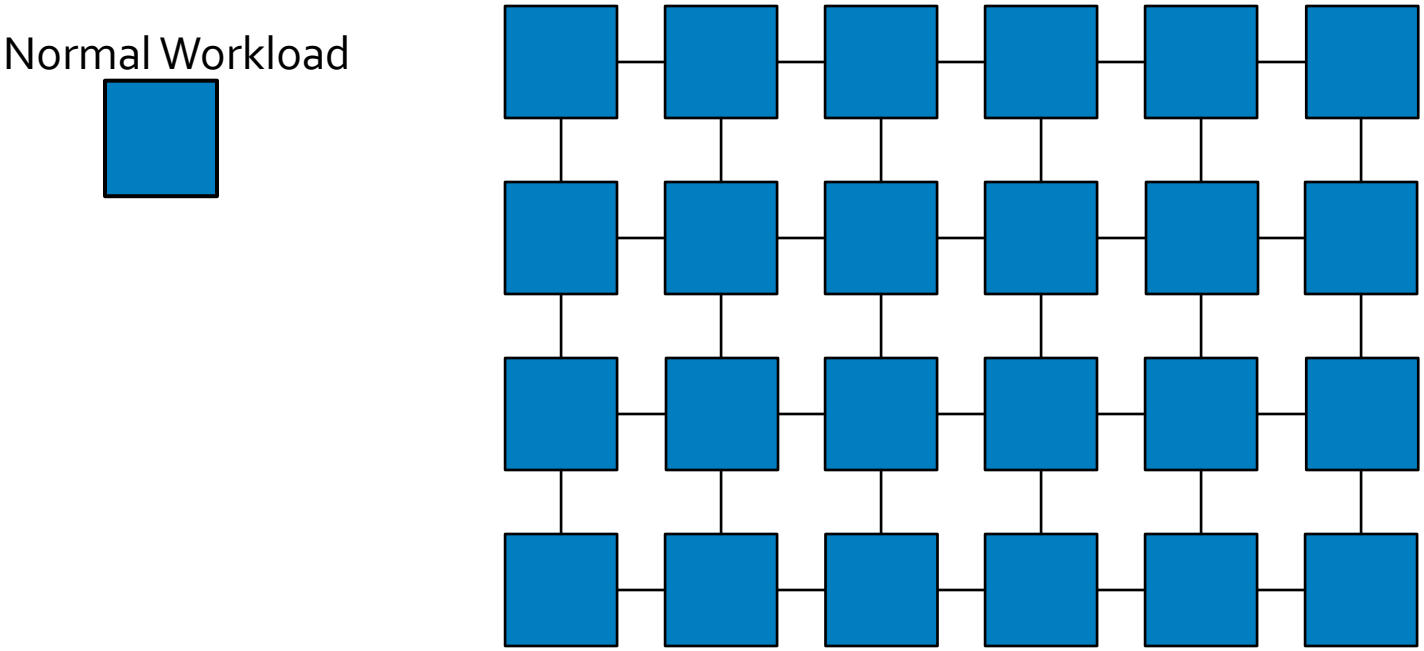
- Online error detection in many-core architectures
- Case study: Intel Single-chip Cloud Computer (SCC)
- Test program parallelization issues
- Proposed parallelization method
- Experimental results
- Conclusion

Online error detection in many-core architectures: requirements

- Redundant test execution
  - all processor cores must execute the same test program
- Small memory footprint
  - single copy of test program (code and data) in shared memory (main memory)
    - less storage requirements
- Reduced execution time
  - trade-off between reliability, performance overhead and availability

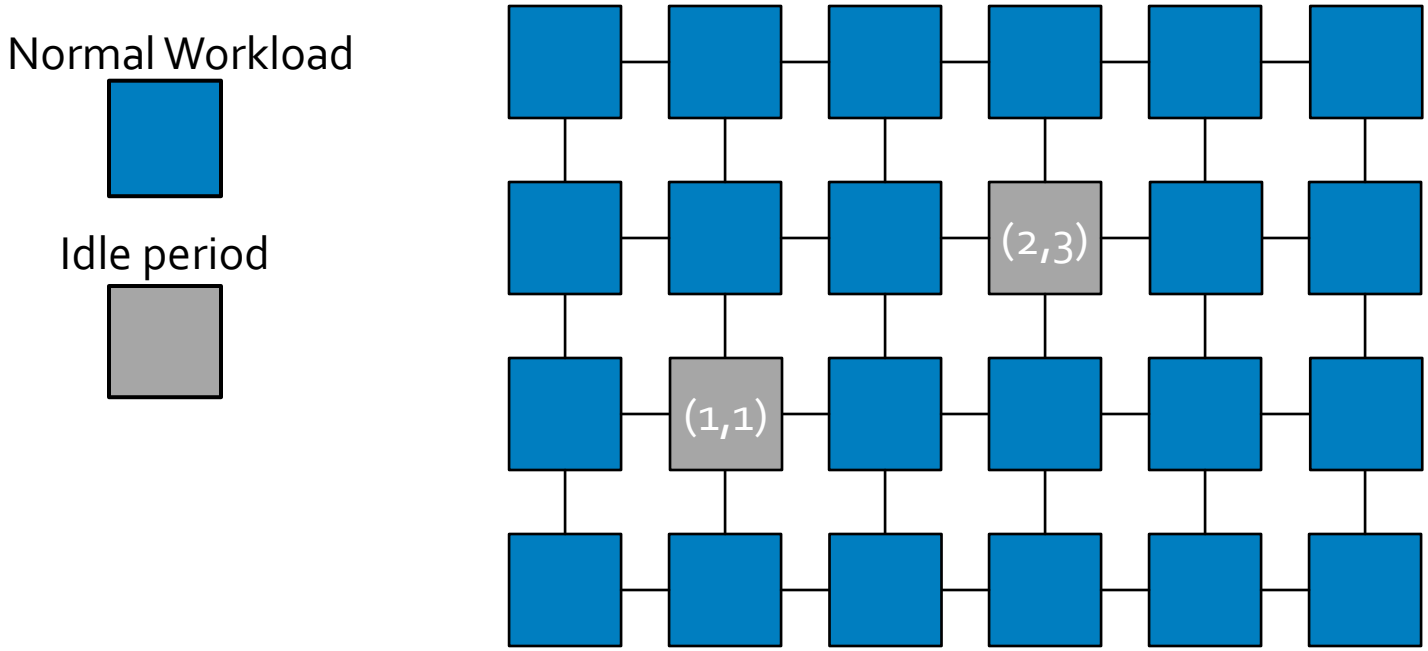
Non-intrusive approach

- Normal system operation not interrupted
  - test programs executed on idle periods



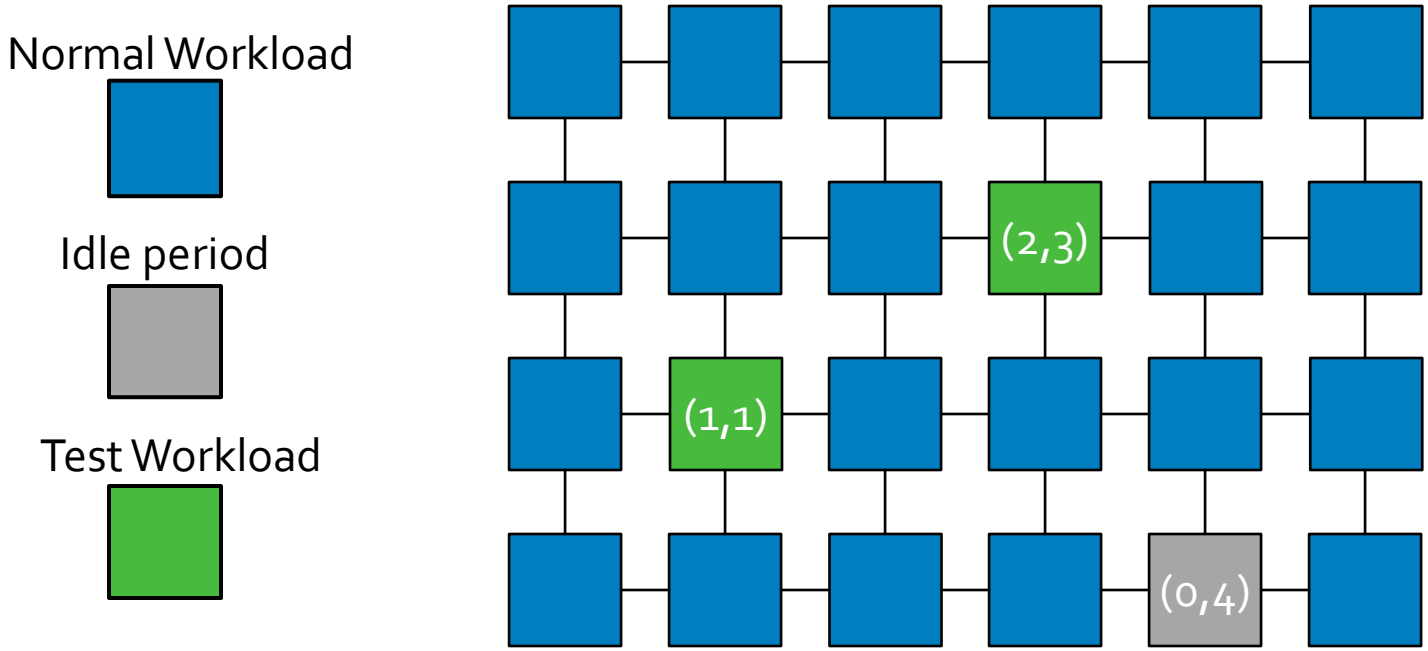
Non-intrusive approach

- Cores (1,1) and (2,3) in idle state



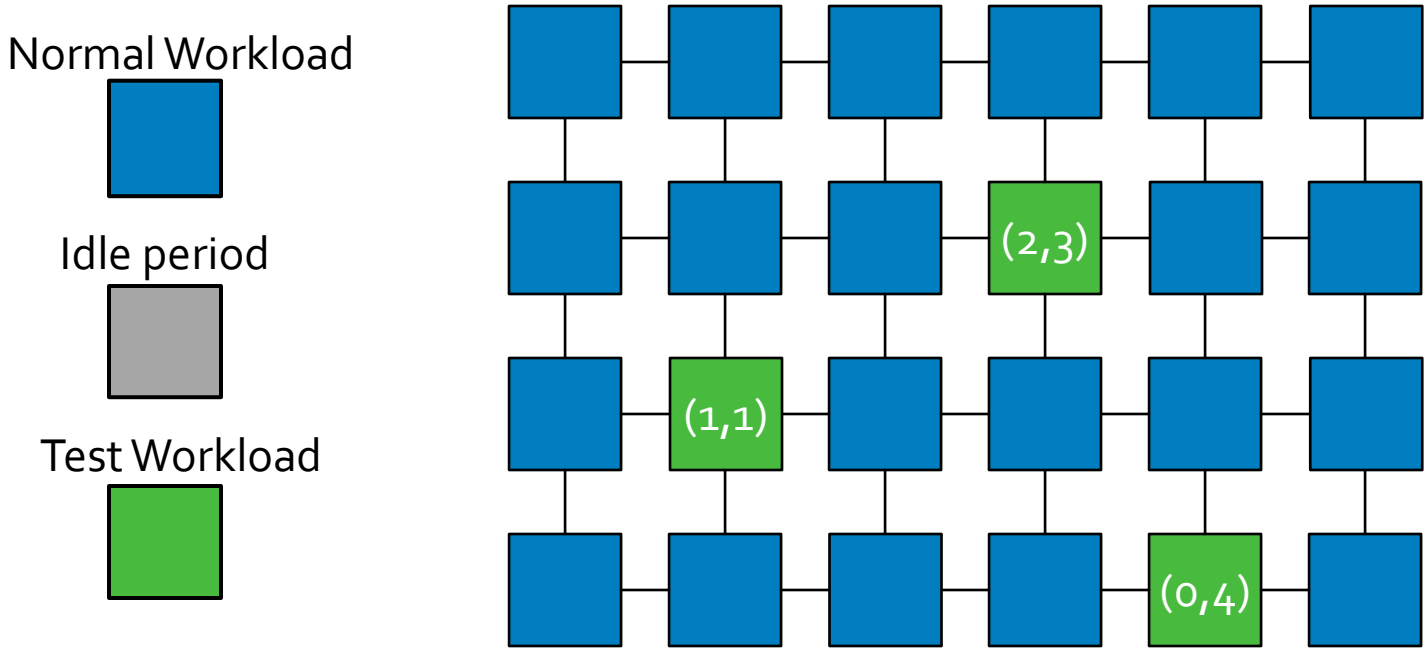
Non-intrusive approach

- Cores (1,1) and (2,3) run test programs
- Core (0,4) in idle state



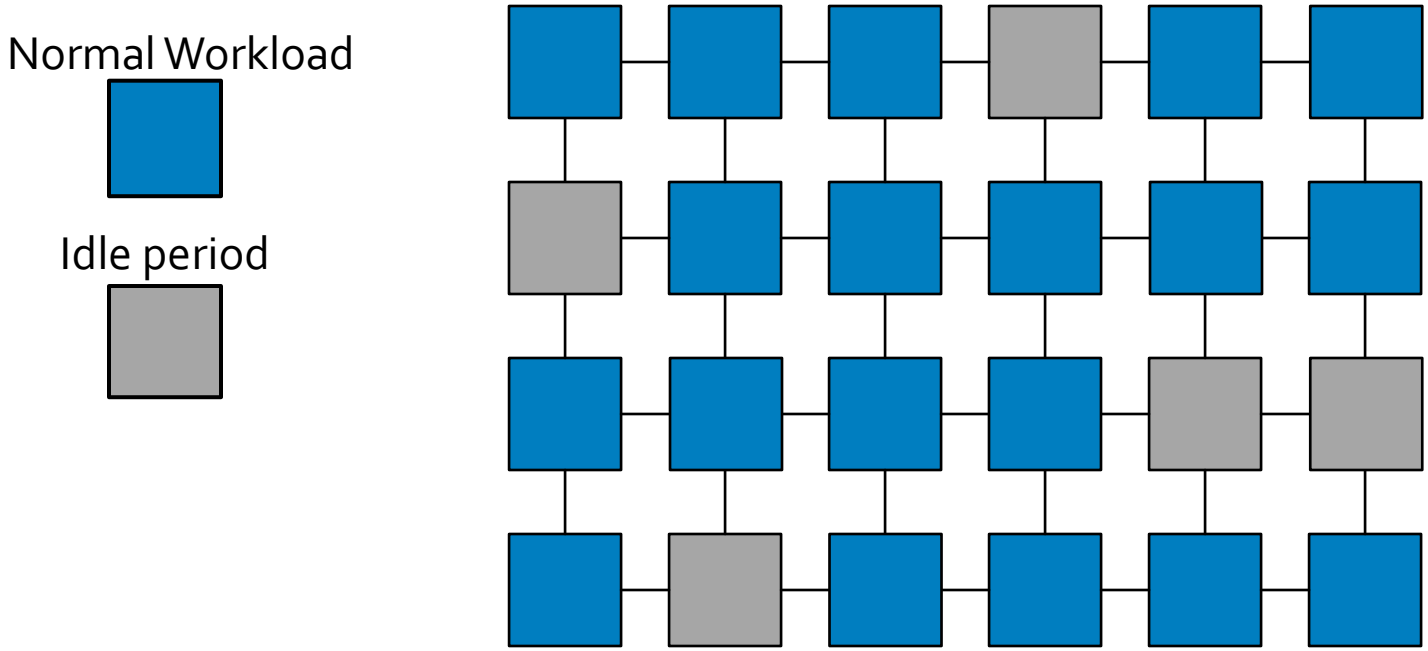
Non-intrusive approach

- Cores (0,4), (1,1) and (2,3) run test programs



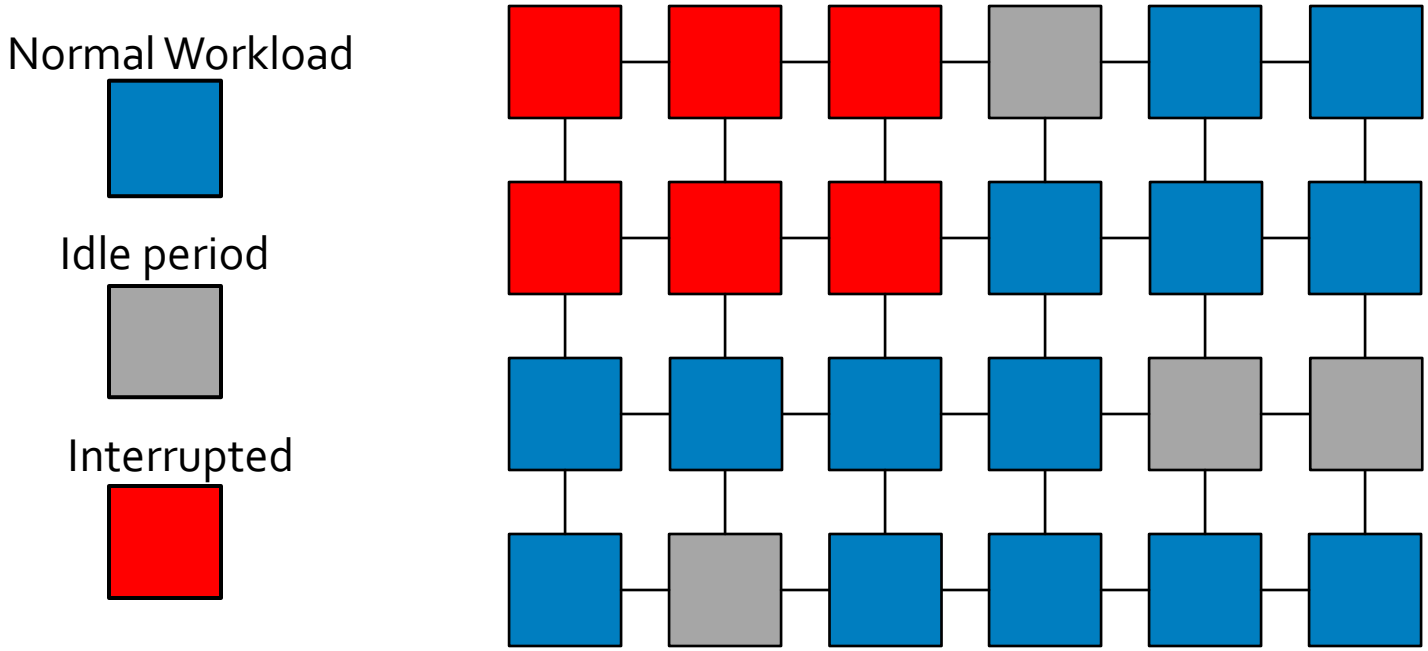
Intrusive approach

- Normal system operation is interrupted
  - test programs stop normal workload



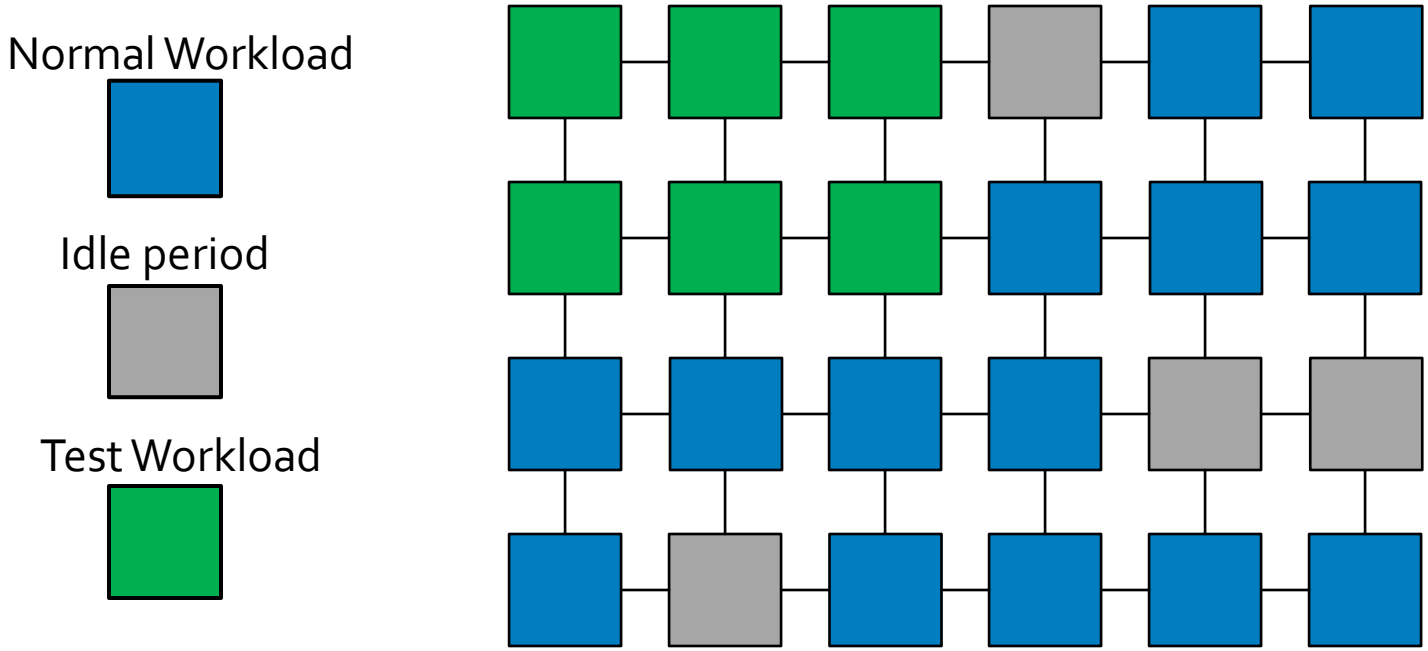
Intrusive approach

- All or a subset of cores is interrupted
  - Upper-left quadrant is interrupted



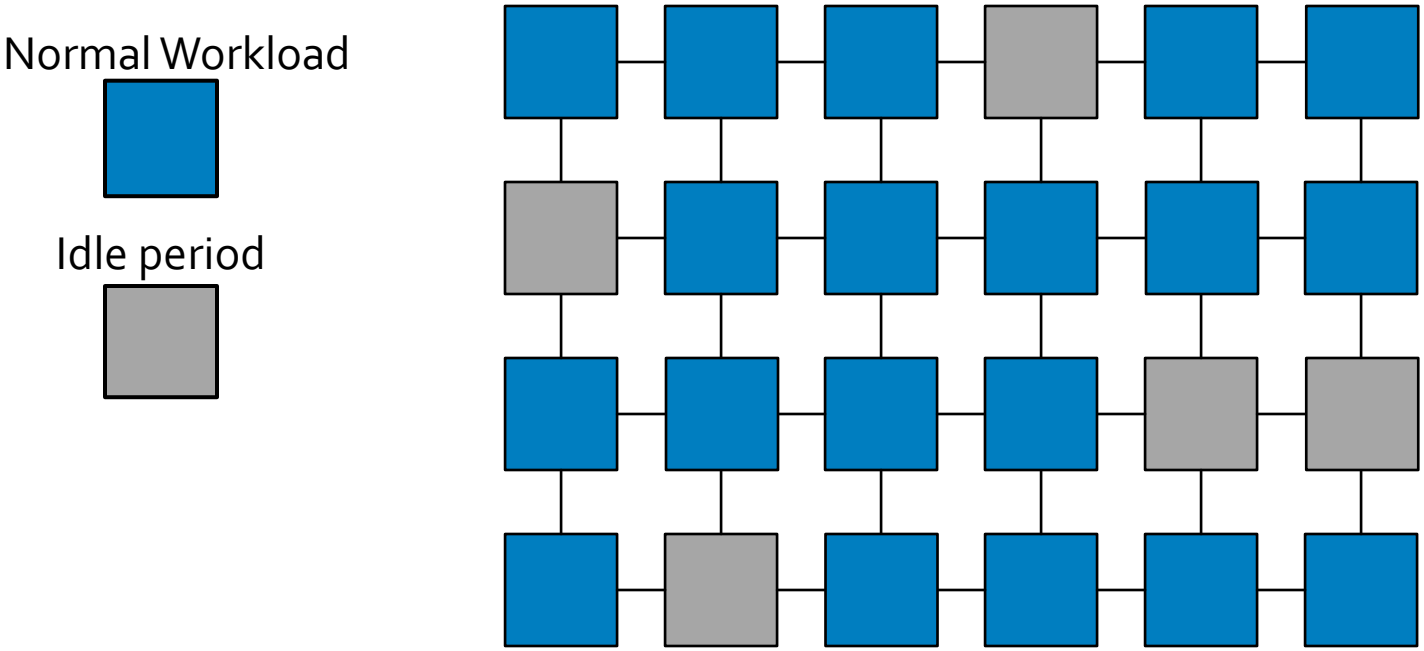
Intrusive approach

- and runs the test programs



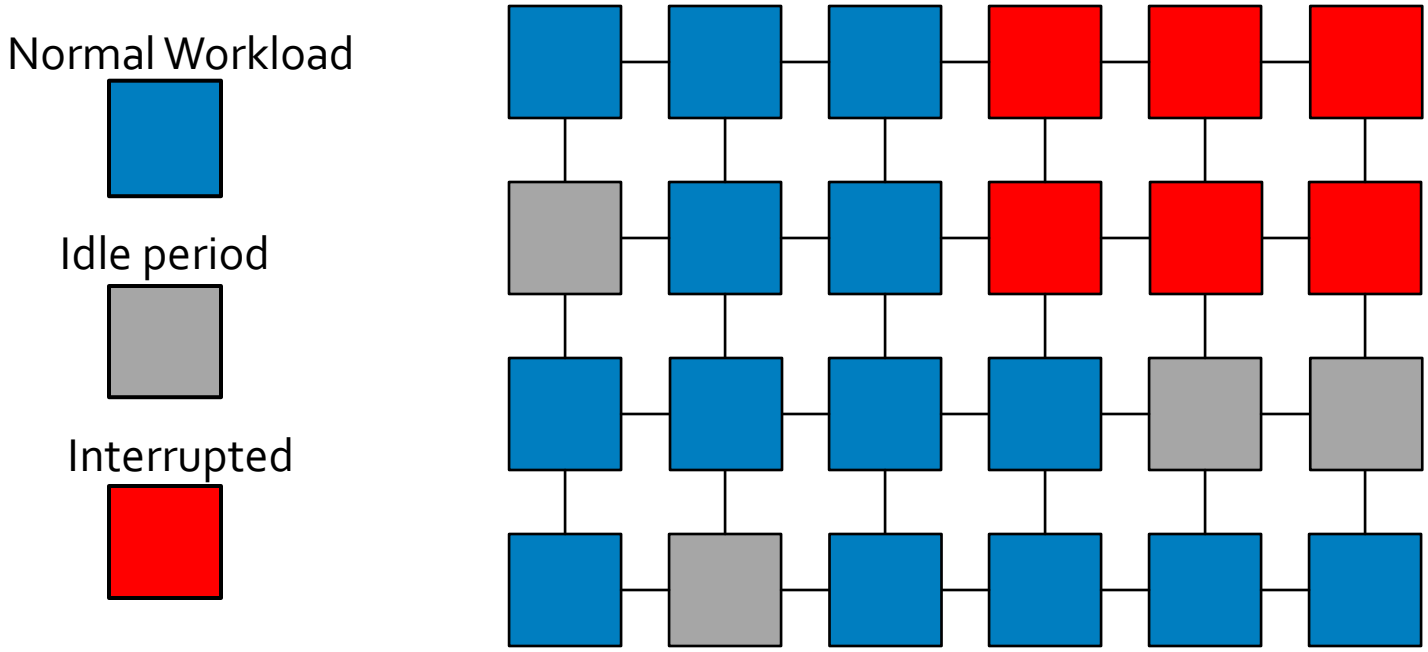
Intrusive approach

- Upon the completion of test programs, interrupted cores resume their normal operation



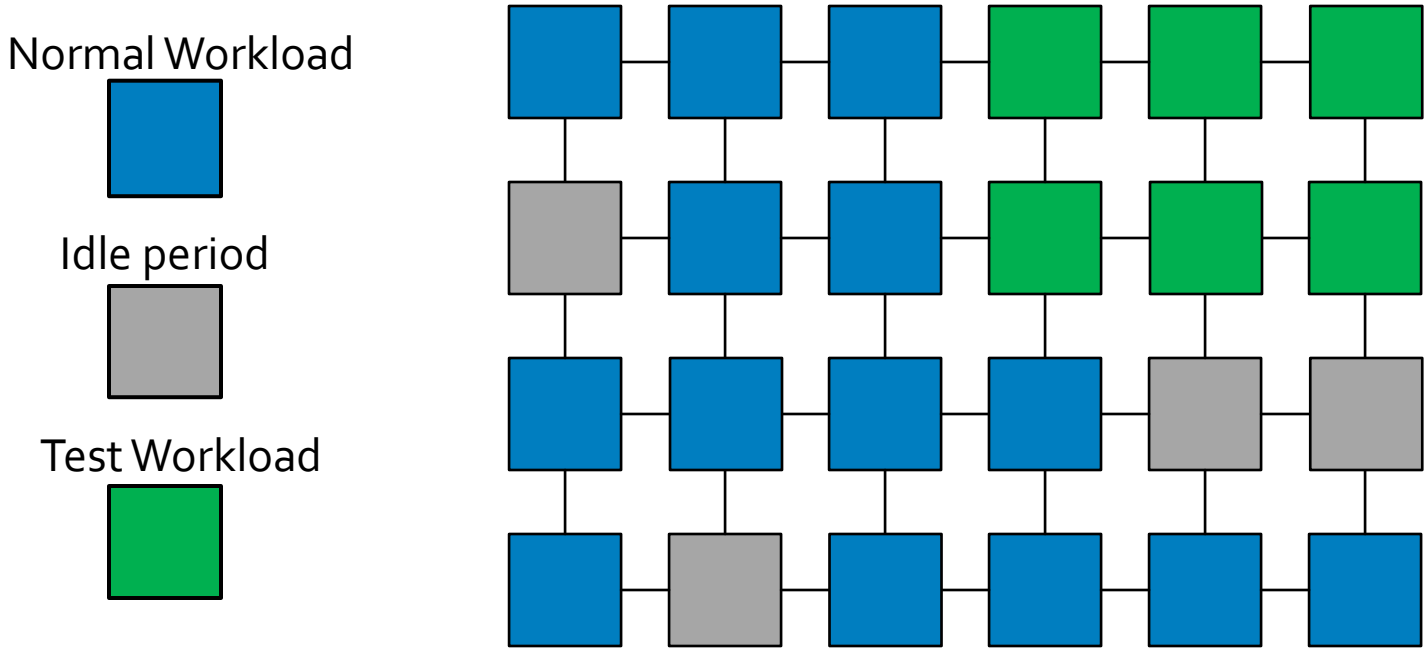
Intrusive approach

- Periodically, another subset of cores is interrupted



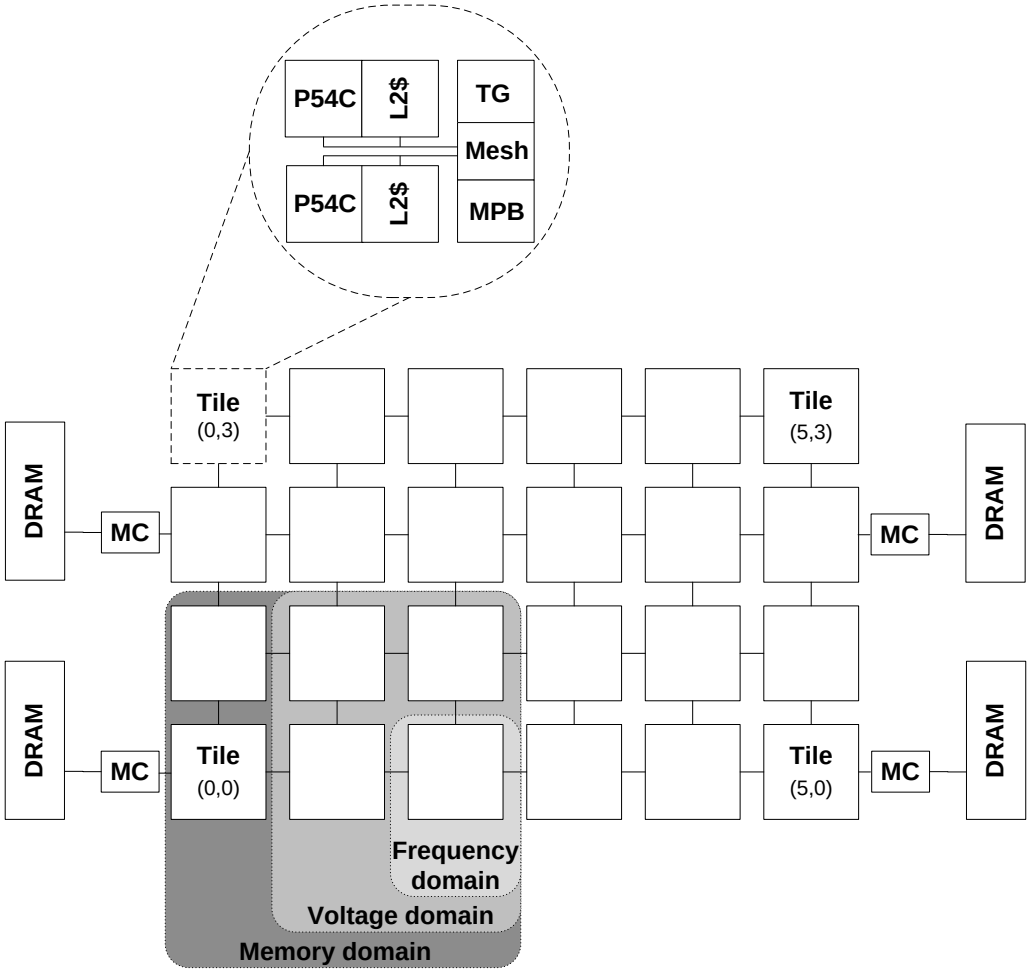
Intrusive approach

- and runs the test programs, and so on



Single-Chip Cloud Computer (SCC)

- Intel's research many-core chip
- 48 P54C cores (24 tiles)
- 4 DDR3 memory controllers (up 64GB)
- 16KB+16KB L1 cache, 256KB L2 cache per core
- 16KB message passing buffer (MPB) per tile



Typical test programs for the SCC

- Load-Apply-Accumulate (LAA) tes
  - ATPG-generated test patterns stored in the off-chip DRAM
  - 192KB or 384KB test patterns per
  - memory-intensive program
- Linear-Feedback-Shift-Register (L
  - pseudorandom patterns generated by a 32-bit LFSR
  - 10 times more test patterns than LAA (3840KB)
  - CPU-intensive program

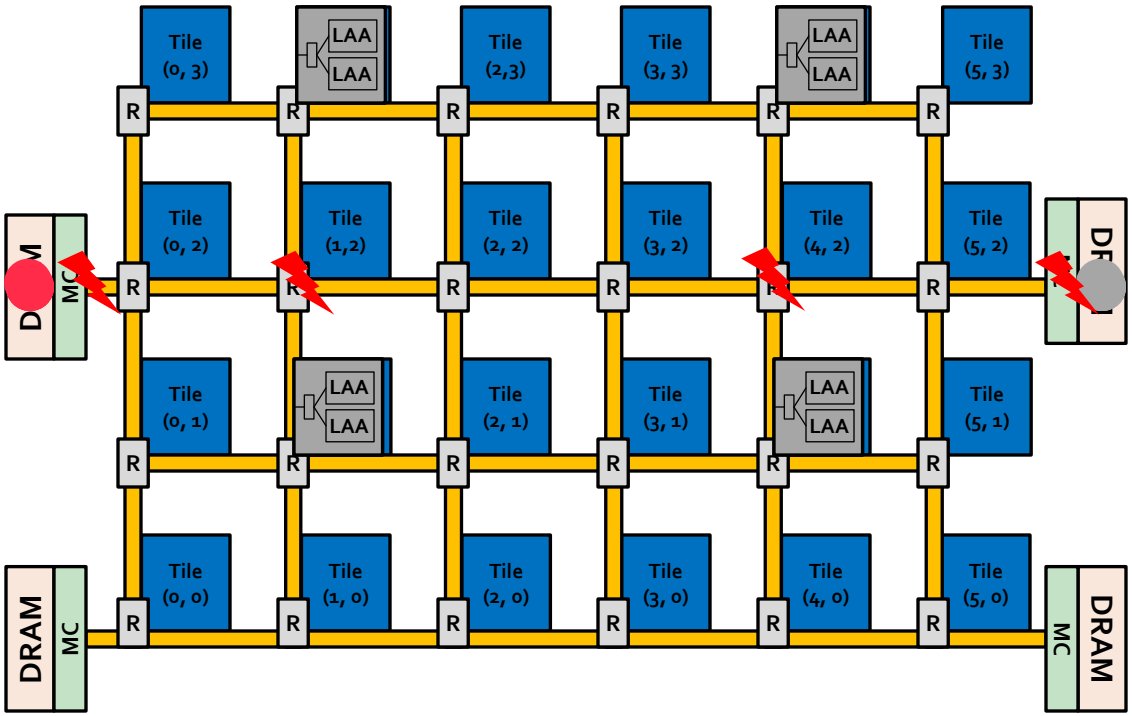
```
int   *Response;
Loop
   // Generate test patterns
   RA = LFSR();
   RB = LFSR();

   // Apply test operation
   RC = RA op RB;

   // Accumulate responses
   Acc = Acc + RC;
end loop

 // Store final signature
 *Response = Acc;
```

Is an online test program embarrassingly parallel ?

- Online test program must be applied individually in every core
- Test program parallelization seems to be effortless
  - all processor cores run the test program in parallel
  - no dependencies, no communication
- Naïve parallelization method in the SCC achieves high speedup for the CPU-intensive test program (LFSR)
  - i.e. 45X compared to the serial execution

Naïve parallelization method

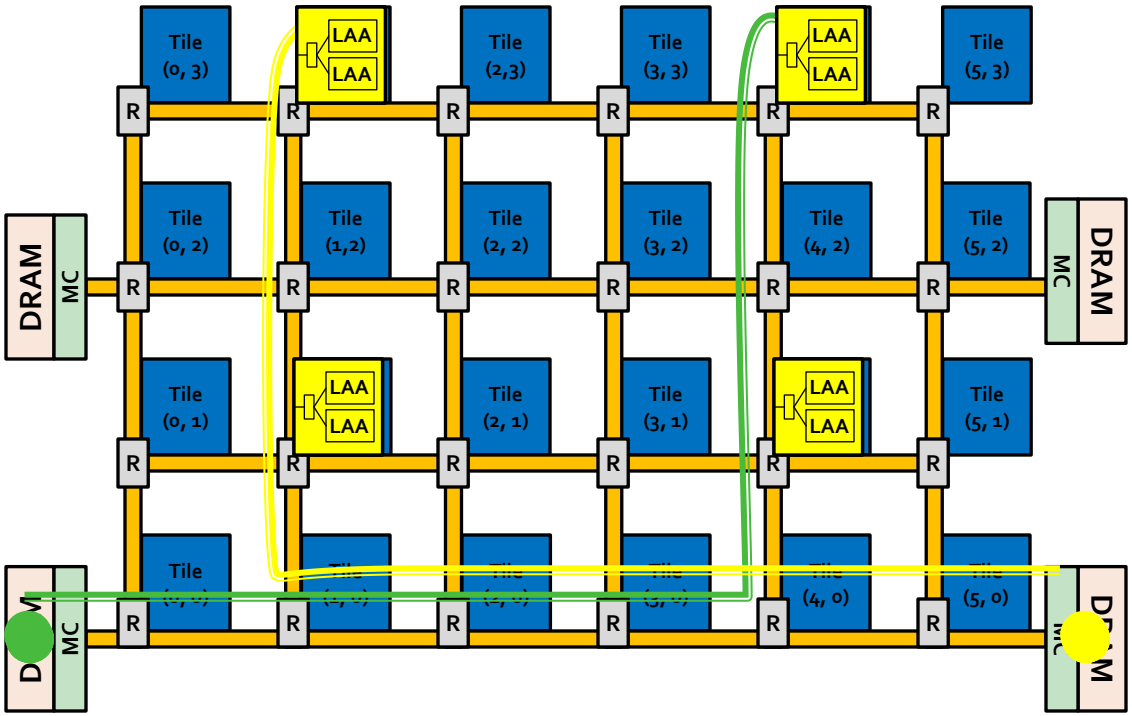
- Low speedup for the memory-intensive test program (LAA)
  - i.e. up to 11X
- Due to the excessive traffic in the interconnection network and MCs
  - during the test load phase



- Tiles (1,1), (1,3), (4,1) and (4,3) running red and gray test patterns

Naïve parallelization method

- Low speedup for the memory-intensive test program (LAA)
  - i.e. up to 11X
- Due to the higher DRAM memory latency
  - to load test patterns from remote MCs



- Tiles (1,1), (1,3), (4,1) and (4,3) running green and yellow test patterns

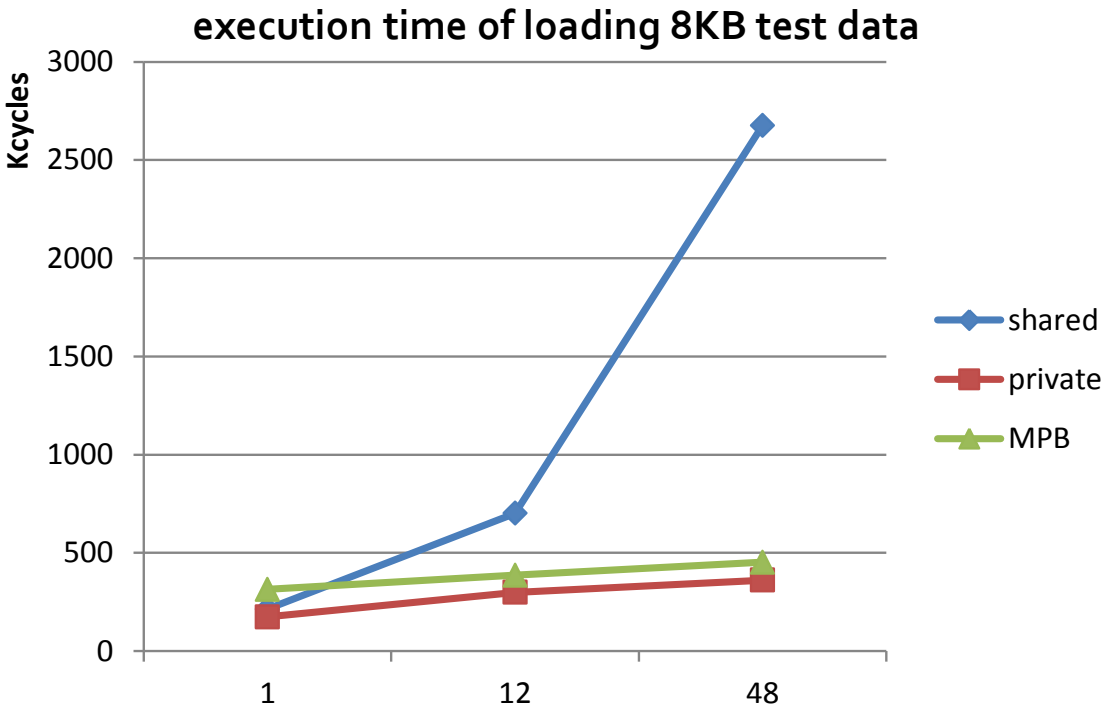
Memory performance issues

- In a previous work (*), we measured the performance degradation when multiple cores run in parallel
  - up to 27% when all cores of a memory domain run in parallel the same memory-intensive test programs

(*) M.Kaliorakis, et al., "Online error detection in multiprocessor chips: A test scheduling study.", *IEEE On-Line Testing Symposium (IOLTS)*, 2013.

Memory performance issues

- Use private memory instead of shared memory to store test patterns
  - private is much faster than the non-cacheable shared
- Use message passing instead of off-chip DRAM to share data
  - to achieve high throughput
    - route of test data must conform to the XY routing rule
    - test data exchange must be done in chunk sizes that fully exploit the MPB size

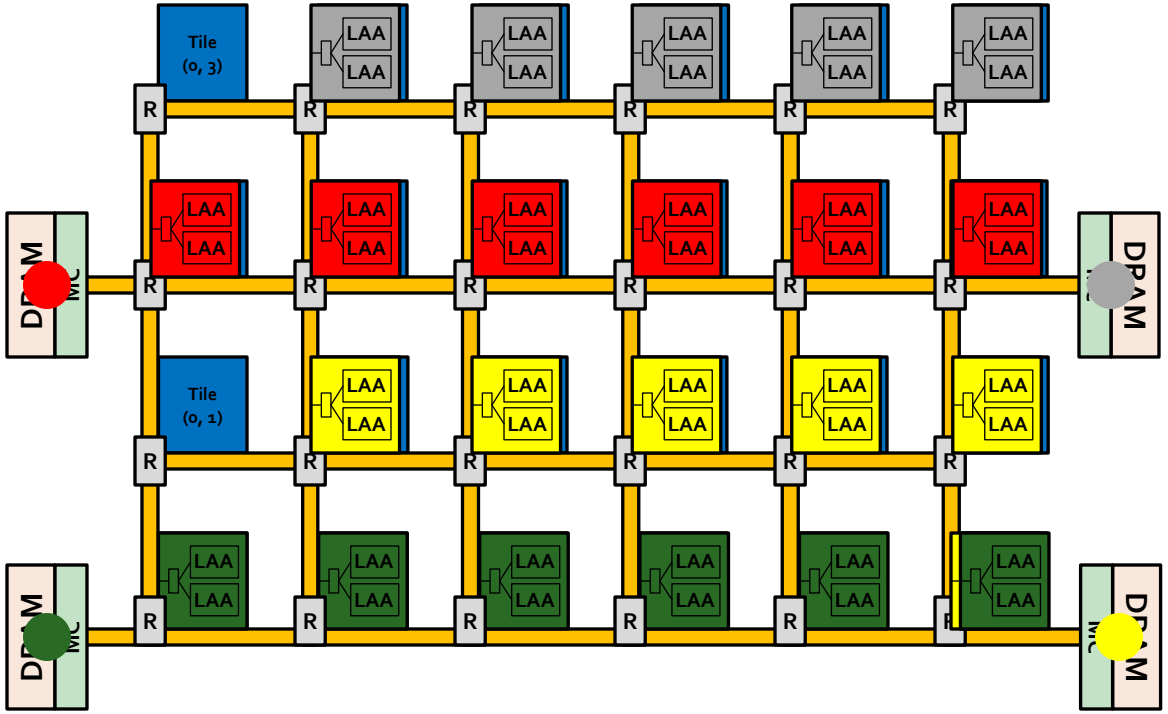


Test program parallelization

- Test *application* and test response *compaction* cannot be parallelized
  - apply all test patterns to every core and
  - compact core's responses separately
- Test *preparation* can be performed *only once* and thus can be parallelized
  - can be divided and executed in parallel in a many-core architecture balancing the test preparation workload

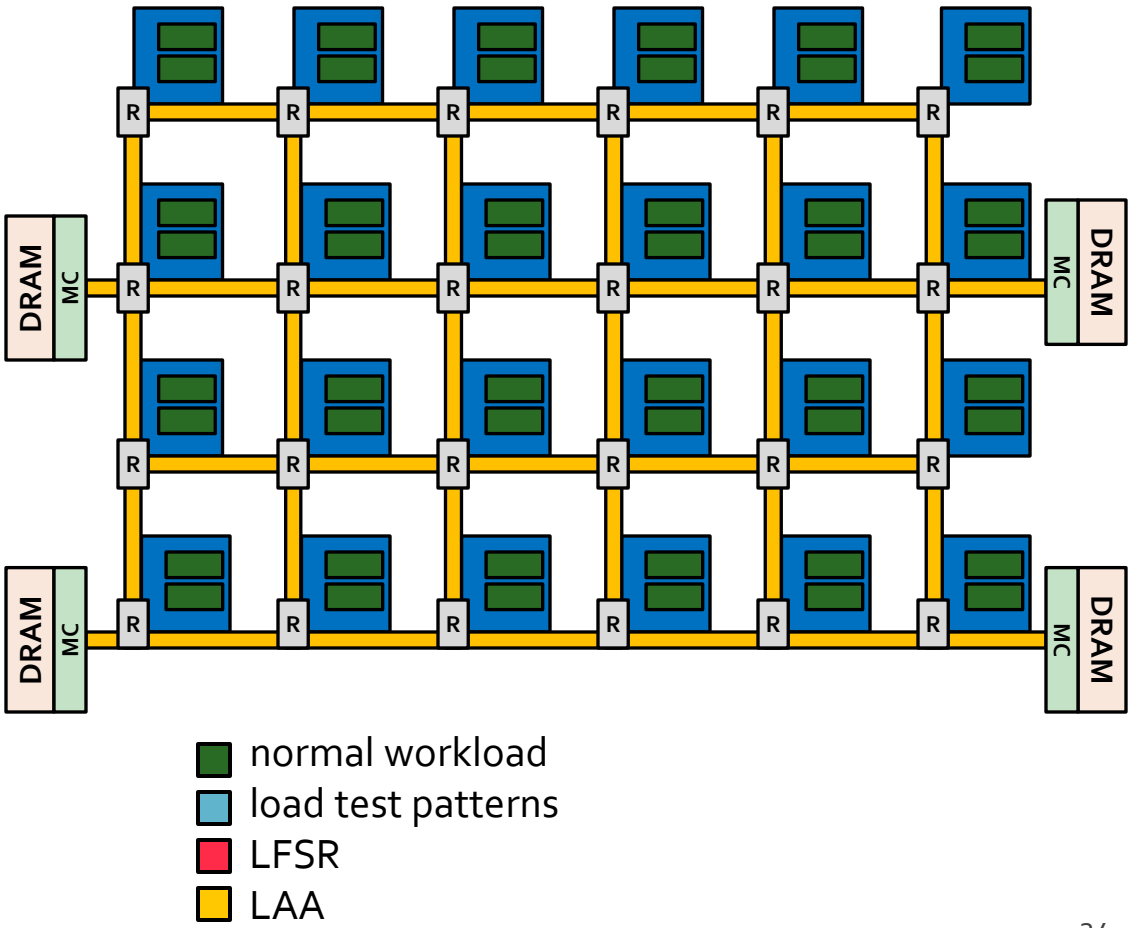
Proposed parallelization method

- Parallel execution of the test preparation phase
- Test patterns are divided into 48 segments
  - each segment assigned to the private memory region of a core
- LAA test program is divided into two phases
  - **Phase A**: all cores load in parallel test patterns from private memories, apply the tests and accumulate the responses
  - **Phase B**: each core copies the test patterns from the local MPBs of the other 47 cores and applies the tests
    - Second phase lasts 47 cycles: in each cycle each MPB serves the memory requests of only one core



Proposed parallelization method

- Concurrent execution of memory-intensive test programs (LAA test) and CPU-intensive test programs (LFSR test)
  - to avoid high traffic congestion in the mesh and the routers.
- First, all cores load in parallel the test patterns
- Next, one core of the tile runs the LAA while the other core runs the LFSR
- Finally, the two cores change roles



Experimental results (1)

SPEEDUP OF THE PROPOSED PARALLEL METHOD OF LAA TEST PROGRAM

Cores/ Test data	Chunk size	Speedup (Serial/Naïve)	Speedup (Serial/Proposed)
12 cores/ 192KB	1KB	4.6X	9.3X
	2KB	4.6X	**9.9X**
	4KB	4.6X	9.5X
	8KB	4.6X	9.2X
48 cores/ 384KB	1KB	10.4X	34.3X
	2KB	10.4X	**38.4X**
	4KB	10.4X	36.9X
	8KB	10.4X	36.0X

Experimental results (2)

TEST EXECUTION TIME FOR THE SERIAL, NAÏVE PARALLEL AND PROPOSED PARALLEL APPROACHES (EXECUTION TIME IN 10^6 CYCLES AND SPEEDUP AGAINST THE SERIAL EXECUTION)

Test Program	Serial	Naïve Parallel	Proposed Parallel
LAA	145.8	14.0 (10.4X)	3.8 (38.4X)
LFSR	1047.8	23.1 (45.4X)	-
LAA+LFSR	1193.6	37.1 (32.2X)	26.9 (44.4X) 25.1 (47.6X)

Conclusion

- Online error detection for many-core architectures
  - Intel's SCC architecture
- Exploit high-speed Message Passing Buffer
  - Memory-intensive test routines
- Accelerate total test execution time **47.6 times**

- Thank you for your attention!

QUESTIONS?