A Resilient Architecture for Low Latency Communication in shared-L1 processor clusters

Mohammad Reza Kakoee
DEIS, University of Bologna
Email: m.kakoee@unibo.it

Igor Loi
DEIS, University of Bologna
Email: igor.loi@unibo.it

Luca Benini
DEIS, University of Bologna
Email: luca.benini@unibo.it

***Abstract*—A reliable and variation-tolerant architecture for shared-L1 processor clusters is proposed. The architecture uses a single-cycle mesh of tree as the interconnection network between processors and a unified Tightly Coupled Data Memory (TCDM). The proposed technique is able to compensate the effect of process variation on processor to memory paths.**

By adding one stage of controllable pipeline on the processor to memory paths we are able to switch between two modes: with and without pipeline. If there is no variation, the processor to memory path is fully combination and we have single-cycle read and write operations. If the variation occurs, the controllable pipeline is switched to pipeline mode and by increasing the latency of the read/write operation we mitigate the effect of the variations. We also propose a configuration-time approach to conditionally add the extra pipeline state based on detection of timing-critical paths.

Experimental results show that our speed adaptation approach is able to compensate up-to 90% degradation in the request path with less than 1% hardware overhead for a shared-L1 CMP with 16 processors and 32 memory banks. We show that even if variation occurs on all processor to memory paths, our approach can mitigate it with an average overhead of 20% on the application's runtime.

I. Introduction

Recently, several many-core architectures have been proposed that leverage tightly-coupled clusters as a building block. Examples include the HyperCore Architecture Line (HAL) processors from Plurality [2], ST Microelectronics Platform 2012 [3], or even GPGPUs like NVIDIA Fermi [4]. In a shared memory paradigm, these designs try to overcome the scalability limitations encountered when increasing the number of processing elements (PEs) that share a unique interconnection and memory system [5] by creating a hierarchical design where PEs are clustered into small-medium sized subsystems. The small number of PEs makes it possible to design a low-latency interconnect between processors and L1 (in-cluster) memories, while scaling to larger system sizes is enabled by replicating clusters and interconnecting them with a scalable medium like a NoC.

The performance of most digital systems today is indeed interconnect-limited, and the design of a high performance on-chip interconnection network is crucial, as the performance impact due to the latency of the interconnection network in a many-core architecture can be as high as the cache miss [6]. In this work, we focus on a fully combinational Mesh-of-Tree (MoT) interconnection network proposed in [1] which is suitable for tightly-coupled processor clusters. This network provides single-cycle transfer from processor to memory and round-robin arbitration for a fair access to memory banks, as well as fine-grained address interleaving to reduce memory bank conflicts.

978-3-9810801-8-6/DATE12/©2012 EDAA

Scaling down of process technologies has increased process variations and transistor wearout. Because of this, delay variations increase and impact the performance of the design. The interconnect architecture in a chip multiprocessor becomes a single point of failure. A faulty processing element may be shut down entirely, but the interconnect architecture must be able to tolerate variations and aging in the circuit, providing graceful performance degradation [7], [8]. Conventional inflexible designs often handle delay variations through conservative guard-bands in the operating frequency and voltage to ensure error-free operation across a wide range of dynamic and static variations over circuit lifetime [9]. Consequently, these inflexible designs cannot exploit opportunities for higher performance by increasing frequency or lower energy by lowering V_{cc} under favorable operating conditions. Since most systems usually operate at nominal conditions where worst-case scenarios rarely occur, static worst-case design severely limits performance and energy efficiency.

For the single-cycle interconnection network where achieving a satisfying working frequency is the key for the performance, conservative guard-banding may ensure safe write/read operation on shared-memory modules but with a high performance and power cost (i.e. over-sized gates and buffers, fast but leaky cells, etc.). Thus, over-design leads to a very inefficient system. For this reason, a design which can detect the variation and tune itself is desired to reduce guard-banding while guaranteeing error-free operation. Some approaches in the literature are based on the error detection sequential (EDS) [10], [11], [12]. In these techniques a combination of a flipflop and a latch can detect the errors in the critical path timing and asserts an error signal. These techniques are suitable for the pipeline stages, and the recovery mechanism is performed by either instruction reply at a lower frequency or multiple-issue instruction reply at the original frequency. Since our timing path is fully combinational, we cannot use these techniques. Moreover, if there is no variation they impose area, power and delay overhead on the design. Post-silicon tuning at the circuit level such as adaptive body biasing (ABB) and dual-Vdd are other types of methods to face with the delay variation [13], [14]. However, they impose leakage and dynamic power overhead and special care should be taken during the design to reduce these overheads. Moreover, since these techniques are at the circuit level, it is very difficult to apply them only on the critical paths.

Contribution. The main contribution of this work is the development of a resilient single-cycle interconnection network suitable for shared-L1 CMP that can tolerate delay variations due to aging or static variations with a small overhead on read/write latency. Our contribution is two-fold:

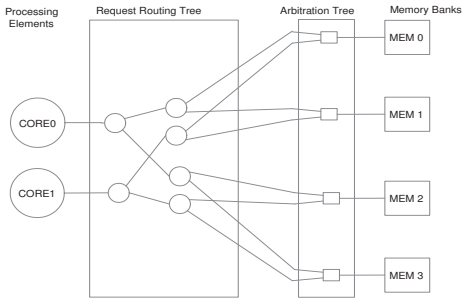

Fig. 1. Mesh of trees 2x4: empty circles represent routing switches and empty squares represent arbitration switches.

first we propose an offline (i.e. boot time) technique which can detect timing failures in the CMP. Our detection approach is based on Test Pattern Generation and Diagnosis performed in off-line mode and during start-up of the system. Then, we propose a reconfiguration mechanism to be triggered after the detection phase. In this phase, we reconfigure the design so that it can overcome the timing failure by injecting pipeline stage in the path and increasing the latency of the read/write transaction. Note that, if there is no variation and therefore no timing failure the network operates with single cycle latency.

Experimental results show that our speed adaptation approach can compensate up-to 90% degradation in the request path and 55% in the response path. We also show that the total area overhead of our approach is less than 1% for a shared-L1 CMP with 16 processors and 32 memory banks. The results show that if we use guard-banding technique, the critical path and the leakage power increase by 19% and 21%, respectively. In the worst case, if an extra stage is inserted on all processor to memory paths, our approach imposes less than 1% runtime overhead on ALU-dominated applications and 50% on memory-dominated programs. However, performance degradation is graceful: less than 10% application slowdown is measured for memory-dominated programs when up to four processors have degraded speed.

II. Interconnection Architecture

A fully-synthesizable Mesh-of-Trees (MoT) interconnection network suitable for shared-L1 processor clusters has been proposed in [1], featuring single-cycle transfer from processor to memory and vice versa. We use the architecture of this logarithmic interconnection. Topology of MoT interconnection which connects 2 processors to 4 memory modules (2x4) is shown in Figure 1. The original interconnection [1], supports non-blocking communication between the processing clusters (PCs) and memories modules (MMs), within a single clock.

As shown in Figure 1, the MoT network connects N=2n PCs and M=2m MMs. It contains Log2M levels of routing primitives and Log2N levels of arbitration primitives. Each memory request issued by PCs must pass through Log2M levels of routing primitives to reach at one of MxN leaf nodes in the arbitration switches; and arbitrates among Log2N levels of arbitration primitives to reach at MM side. In reverse order, memory responses propagate through arbitration and routing levels to reach at PC side.

A. Network Operation

During a read/write operation, data and control signals, asserted by the PCs, are routed through combinational routing switches, until they reach one of NxM ports of routing tree. In order to reach the memory module the request must be arbitrated among the other simultaneous requests for the same memory module. After passing through all levels of arbitration switches, the request reaches the memory module, and the read/write operation can be performed. Once the request reaches the last level of the arbitration tree and gets the grant, a valid acknowledgement is asserted and propagated back to the related PC through the routing switches (backward). By receiving the acknowledgment signal, PC is able to issue the next read/write operation at the next clock cycle, otherwise it waits until it is received.

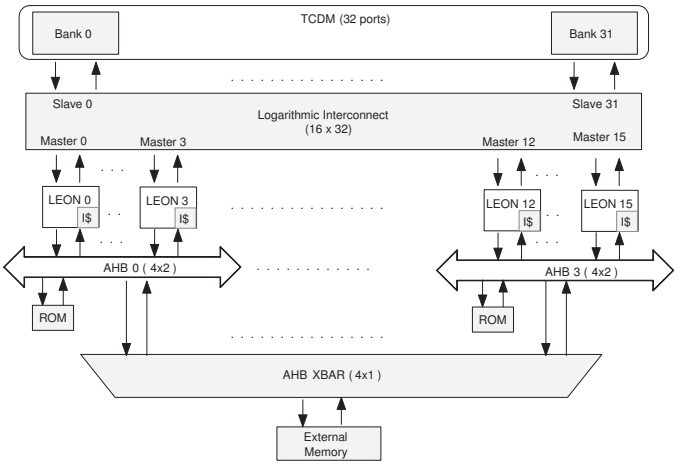

Fig. 2. Mega-leon architecture.

III. Mega-Leon Architecture

A complete multi-core system named Mega-Leon is designed and implemented as a case study for applying our approach. The Mega-Leon architecture is an evolution of the Multi-core Leon developed by Gaisler [15]. It contains sixteen SPARC-V8 processor cores which are connected through the high-bandwidth logarithmic network described in the previous section to a fast multi-banked, multi-ported Tightly-Coupled Data Memory (TCDM). The number of memory ports in the TCDM is equal to the number of banks to allow concurrent accesses to different banks. Processors can synchronize by means of standard read/write operations at logarithmic network providing test-and-set semantics (hardware semaphores). The cores are also connected to an AHB shared bus system to access the external modules (external memory, ROM, debug support unit and etc.). Figure 2 shows the diagram of this architecture. The SPARC-V8 used in this platform has been customized. Its memory controller has been modified to make the interface with TCDM. The embedded data cache has been removed and replaced with the TCDM which is shared between 16 cores.

As mentioned before, the request and response paths from processor to TCDM and vice-versa are single-cycle and fully combinational. These paths are shown in Figure 3. It is shown in Figure 3 that the request and response paths include not only the interconnection network but also some logics in the processor's pipeline as well as the memory controller which decodes the address and communicates with the network. To better understand the behavior of the request and response

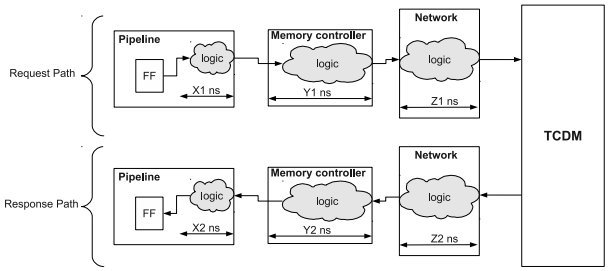

Fig. 3. Request and response paths from processor to TCDM.

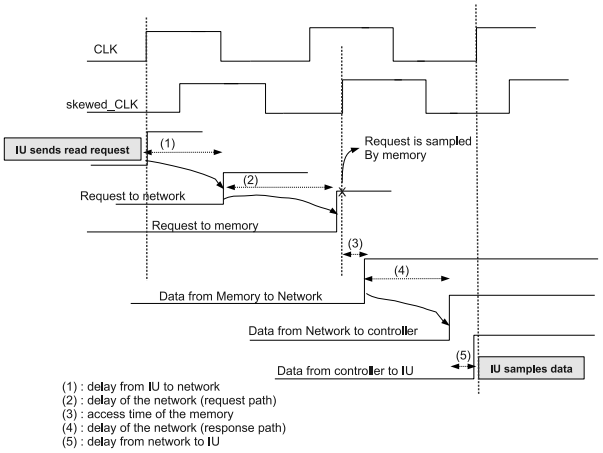

Fig. 4. Detail timing of the read access from processor to TCDM.

paths, the timing diagram of a read access is shown in Figure 4. In order to provide a single-cycle communication, each read/write request must be concluded within the clock period. To achieve this goal, we assume that PCs and network (only for round robin priority switching) are clocked with the main clock CLK, whereas MMs are clocked with a skewed_CLK (same frequency and typically 90° phase shift) [1]. These two signals are available at the input ports of the network, and are generated through an external block (PLL). It is clear that, for a given network configuration (NxM) and technology, the forward and backward latencies are lower bounded for maximum performances. By tuning the clock frequency, phase shift and memory access time, it is possible to meet the target frequency, thus avoiding timing violations.

However, in order to have a design working after fabrication we need to consider static variations and transistor wearout in the circuit which cause delay variations. Delay variations which happen in the combinational paths from PC to MM (and vice-versa) as well as the access time of the memory may lead to the complete failure of the PC-MM/MM-PC communication. In the next section we describe how we modify the architecture so that it can detect the timing violation and reconfigures itself to avoid communication failures.

IV. Reliable Architecture

The paths from processors to TCDM and vice-versa are fully combinational and have very tight timing constraints. Due to this fact, a little variation on the delay can lead to the complete failure of the processor-memory communication. In the following we discuss on some conventional approaches to face with this problem.

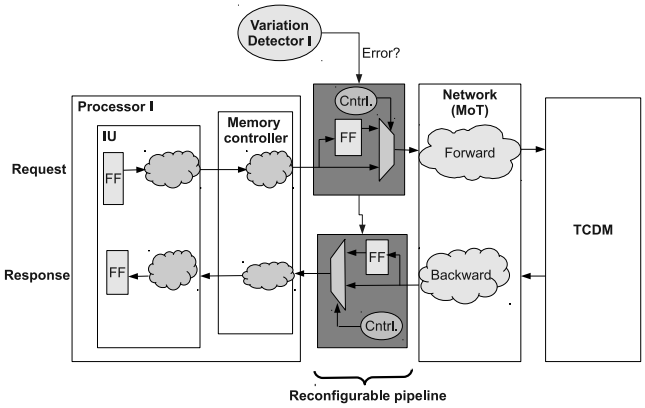

Fig. 5. Request and response paths with variability-compensation modules.

(i) The first solution is adding conservative and sufficient margin for the delay variation. This can be achieved by designing at the worst-case corner. However, this technique limits the performance and energy efficiency of the system. Synthesizing the design at the worst case corner increases both area and power of the design due to using large and low-Vth cells and still has lower MAX-frequency with respect to the nominal corner.

(ii) Another solution is using methods which are based on double-sampling with time-borrowing (DSTB) error-detection sequential (EDS) [10], [11], [12]. These techniques require at-least one stage of the pipeline on the path; while our path is fully combinational and if we add a pipeline for that, it increases the latency of the processor-memory communication which limits the architecture performance in the favorable mode when there is no delay variation.

(iii) Another technique is simply breaking the path and putting one stage of the pipeline on it. As mentioned before, this approach violates the main property of the architecture which is having single-cycle latency and limits the performance when there is no delay variation.

All the above techniques have performance penalty even if there is no variation; but, we need a solution that imposes speed overhead only if a failure happens in the timing of the processor-memory communication. We propose a new approach which is based on the reconfigurable pipeline.

We add a reconfigurable module on the path so that it can compensate the effect of the delay variation by inserting one cycle of the latency on the request or response transaction relaxing the timing constraints. This module does not increase the latency in the normal mode (without delay variation) and the read/write operation is completed in one cycle which is the main property of the architecture. Our reliable architecture for one processor is shown in Figure 5.

The reconfigurable pipeline is inserted in the middle of the combinational path i.e. between memory controller and the network. Every Processor has one delay fault tester which can detect failures in the read/write operation. Detection is performed off-line as described later. If the tester does not find any timing error, it sends a no-error signal to the reconfigurable pipeline. This module reconfigures itself in such away that the processor-memory path becomes fully combinational by selecting the second input of the Multiplexer (Figure 5); in this

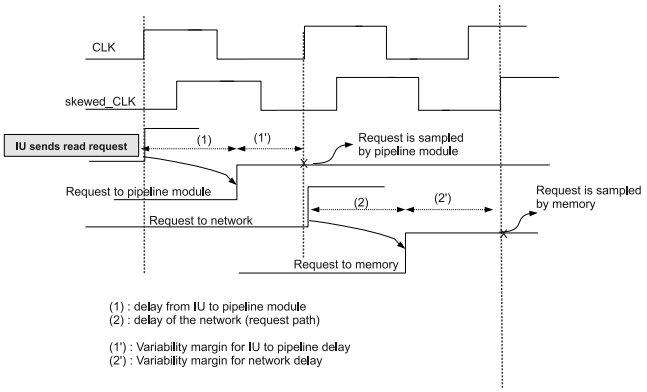

Fig. 6. Detail timing of request path from processor to TCDM when there is variation (2 cycles latency).

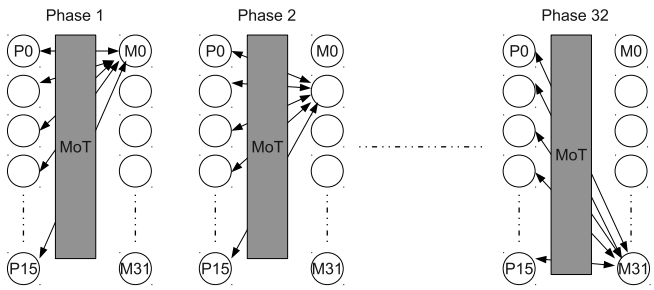

Fig. 7. Test phases for 16 processors and 32 memories.

mode the Flip-flops are out of the path. If variation happens and the tester finds an error, it asserts the related signal and the controllable pipeline switches to use Flipflop in the path by appropriate signaling with both memory controller and the network. Therefore, an extra cycle of latency is added to the read/write transaction. Figure 6 shows the timing diagram of the request path in the presence of delay variations (Flip-flops are in the path).

We should note: because both detection and reconfiguration steps are offline, our approach is not suitable for dynamic delay variations due to the temperature changes or V_{cc} droops where the delay of the circuit changes dynamically when the system is working. Techniques which are based on Error Detection Sequential are very useful to mitigate these variations. Our approach is best suited for static delay variations due to aging, random variations such as doping related variations, variation in transistor threshold voltage caused by density variations of impurities in the transistor material, and systematic variations such as exposure pattern variation in lithography process or silicon surface flatness variations in the CMP (Chemical Mechanical Planarization).

A. Detection Mechanism

To detect the timing failure in the read/write operation we propose an offline technique which is based on functional test pattern generation and diagnosis. Similar techniques have been proposed in the literature to detect manufacture failures in the chip after fabrication [16], [17].

We use a tester module which is responsible to send the test patterns and to verify them. This module implements two similar state machines which generate data and address for the write and read transactions and verify the incoming data from memories. We have only one tester which is replicated for all processors. It gets the ID of the processor as an input.

The complete test is performed during M phases where M is the number of memory banks. During each phase all testers write the test pattern to the same bank but at different locations and then read and verify them. Testers get the phase number as an input and based on that generate the write/read address. The related phases for 16 processors and 32 memory banks are shown in Figure 7. The test pattern (data and address) is chosen so that it can detect the timing failure in the read/write operation. The test patterns contain two consecutive writes and two consecutive reads. The first data that is written is All-zero and the second data is 01010....01. The data are written at addresses ID and $ID + 1$ in the Memory bank I where ID is the identifier of the processor and I is the number of the testing phase. Since at each phase of the test all processors write/read to the same bank, the maximum contention occurs in the interconnection network and therefore all the arbitration paths as well as routing paths from all processors to Bank I (in Phase I) are tested. Tester writes data to the specified locations of each bank and then reads the same locations. If it reads the wrong data, it means one of the write or read operations or both do not work correctly and tester generates an error signal and reconfigurable module is switched to the pipeline mode. After that we perform one extra test to see if after using the pipeline in the path the test is passed or not. If the test is not passed for the second time, we consider the related processor as faulty.

V. Experimental Results

In this section, we discuss the experimental results for the resilient Mega-Leon architecture in terms of delay, power, area and the latency. We quantify the cost of adding variation-tolerant modules into the original design. To get these results, we synthesized the whole architecture including 16 processors, and the interconnection network on a general purpose 65nm commercial technology library [19]. 32 memory banks of 8 KBytes (256 KB in total) are also obtained from the same technology library. Using RTL simulation and the compiler tool-chain of the SPARC-V8, we ran different applications (C program) on our multi-core architecture to analyze the effect of the compensation approach on the performance of the applications.

A. Design Flow of Resilient Architecture

To obtain the maximum frequency at which our Multi-core architecture can operate we developed an advanced design and synthesis flow. The synthesis stage is performed in two passes: the first is a pure logic synthesis, and the network and cores are synthesized without physical constraints. Synthesis is performed at the nominal corner with $1.2V$ and $25°C$. In order to reduce the power, we exploited multi-Vth design where low-Vth cells are used for timing critical paths and high-Vth cells for other parts of the design. The preliminary output netlist is used in the back-end tool to perform the power planning and floorplanning, and at the end of this step, the physical information are exported in a "DEF" file and back-annotated in the synthesis tool, with topographical features enabled. The second synthesis run performs both remapping and coarse placement plus physical optimization taking into account physical geometries based on the back-annotated

floorplan. Using this methodology, good convergence between post-synthesis and post-layout results is achieved early in the flow. For routing global wires we used the technique proposed in [18] during place&route. Using this design flow we could achieve a frequency of 250 MHZ (critical path=$4ns$) for our architecture which includes 16 processors, logarithmic network and 32 memory banks (each 8KB).

To compare our approach with the guard-banding technique where the design is synthesized with a very conservative margin, we performed the synthesis on the same library but at the worst-case corner ($1.1V$ and $125°C$) and with the same timing constraints. Table I shows the related results.

TABLE I
SYNTHESIS RESULTS OF NOMINAL AND WORST-CASE CORNERS

	Multi Vth	*Critical Path* ns	*Area* mm^2	*Leakage* mW
Nominal	✓	4	3.993	2.85
Worst-case	✓	4.75	4.078	3.44
Worst Case overhead	**-**	**19%**	**3%**	**21%**

As seen in Table I, by synthesizing at the worst-case corner, which contains slower cells, the critical path increases by 19%. This critical path was the best timing that we could get using the worst-case corner. The leakage power increases by 21% because the synthesis tool uses more low-Vth cells (which are more leaky) to meet the timing. The area increases by 3% because of using larger cells to meet the timing constraints.

After carefully reviewing the timing paths of the design, as it was expected, we found that the most critical paths are between processors and the TCDM. They start form the Integer Unit (IU) of the processors, traversing the memory controllers and interconnection network and ending at the memory banks. We also found that the delay of the paths whose sources and targets are inside the processors is almost $2/3$ of that of the critical path (PC to TCDM). Therefore, if any variation occurs, those paths that are inside the processors have reasonable margins to tolerate it. Thus, we have to take care of the paths between processors and memories.

To do so and to apply our variation-tolerant approach, we had to find a suitable location on the critical path to insert the reconfigurable pipeline. By carefully investigating all critical paths (processors to memories) of the whole architecture, we figured out that the best position is exactly before the interconnection network and after the memory controller. That location was about $2.1ns$ away from the beginning of the request path and $1.5ns$ from the end of it. It was $2.6ns$ away from the beginning of the response path and $1.1ns$ from the end of it. We put our reconfigurable pipeline exactly before the network. Putting the pipeline at this location creates a large margin for the delay variation and enables us to compensate a degradation of 90% $((4-2.1)/2.1)$ on the request path and 55% $((4-2.6)/2.6)$ on the response path. In other words, if the variation causes that the delays of the request and response paths increase by 90% and 55%, respectively, we are still able to compensate it by switching to the pipeline mode.

To realize the range of the variation that can occur on the design, we performed worst-case timing analysis on the design synthesized at nominal corner. We calculated the delay of the request and response paths in both corners. Table II shows the results. Note that, for these results the design is synthesized at the nominal corner but the timing analysis is performed at both nominal and worst-case corners. It is shown that the maximum variations on the request and response paths are 66% and 30% respectively. Therefore, our approach can compensate the delay variations of both request and response paths if the design is in the worst-case mode.

Based on the results of Tables I and II, if we synthesize our design at the nominal corner we would get a reasonable speed without a huge synthesis effort and power and area overhead. Then at the run-time we can re-adjust the design by adding the pipeline stage if we are in the worst case condition. Since the worst case is very rare, this will impact only very few chips. Moreover, although using the pipeline mode in the presence of the variation increases the latency of read and write transactions, it does not change the working frequency of the whole architecture and processors are still running at the full speed.

TABLE II
TIMING ANALYSIS OF DIFFERENT PATHS ON DESIGN SYNTHESIZED AT NOMINAL CORNER

Path	*Nominal corner delay (ns)*	*Worst-case corner delay (ns)*	***Variation*** %
PC-Network	2.10	3.5	**66%**
Network-TCDM	1.57	2.4	**52%**
TCDM-Network	2.6	3.38	**30%**
Network-PC	1.1	1.3	**18%**

B. Hardware and timing overhead

Our approach requires a few hardware modules including the tester and the reconfigurable pipeline to be added to the original design. These additional blocks are per processor. Therefore, we have 16 tester and pipeline modules for the whole system. We also need one global controller which controls the flow of the testing and synchronizes different test phases. Table III shows the hardware overhead of these modules.

TABLE III
AREA OVERHEAD OF OUR RESILIENT ARCHITECTURE

Module	*Area* μm^2	*Overhead on one processor*	*Overhead on Mega-leon*
Tester	2250	2%	0.8%
Reconfigurable pipeline	1200	1%	0.5%
Global controller	957	-	0.01%
All	4407	-	1.3%

As can be seen in this table, the total area overhead of our approach on the whole Mega-Leon architecture is less than 2% which is very small for our resilient architecture.

The reconfigurable pipeline adds one Multiplexer on the critical path and it may increase the delay even if there is no variation. We calculated this additional delay and it was around $20ps$ (0.5% of the critical path ($4ns$)). This small additional delay had no effect on the target frequency since the synthesis tool was able to optimize it.

C. Testing time

As mentioned before, each test sequence contains two writes and two reads. All processors perform at-least one test sequence during each phase. If the first test sequence fails then the related reconfigurable pipeline is switched to use the

extra stage and another test sequence is initiated. Therefore, each processor performs a maximum of two test sequences in each phase. During each phase, the first test sequence takes 4 clock cycles in the tester, and 4 cycles in the processor to memory communication (8 cycles in total). The second test sequence (if the first one fails) takes 4 more cycles (12 cycles) due to having pipeline in the path. Therefore, each processor needs a maximum of 20 clock cycles for each test phase. Since during each phase all processors communicate with the same bank of the memory, maximum congestion happens in the interconnection network. In the worst case processors can get grant sequentially. Therefore, each phase of the test takes 320 clock cycles ($16 * 20$). Since we have 32 different phases the testing procedure requires $32 * 320 = 10240$ clock cycles. We consider some other cycles for synchronization between different phases. At the worst-case we need 11000 clock cycles for the testing procedure to be finished. With a clock period of $4ns$, the complete detection process takes $44\mu s$. We should again note that, the testing procedure is performed offline and during start-up of the system.

D. RTL simulation and performance analysis

As described, our speed adaptation technique increases the latency of read and write operations. This may slow-down the applications running on the processors. The amount of the slow-down depends on the type of the application. If the application is ALU-dominated and has a few transactions with the memory then the slow-down is very small. However, if the program is memory-dominated and has a lot of load and store instructions, the slow-down is higher.

To evaluate the effect of our approach on the application's run-time, we created some benchmark programs with the help of SPARC-V8 tool-chain. These programs are written so that they can cover a range of applications from very ALU-dominated to very memory-dominated. For accurate evaluation, we did not use any high level simulation, but we used the real RTL model with a commercial RTL simulator.

Figure 8 shows the results of our speed adaptation approach on four applications. The curves in the figure show the completion time of each program. X axis is the number of processors for which our speed adaptation technique is applied.

As can be seen, for an application like Fibonacci that is ALU-dominated the overhead of our approach on the run-time is almost negligible. For memory dominated applications like Matrix Multiplication and Transpose, if variation occurs on up-to four processor-memory paths, the overhead is very small (less than 10%). If the variation occurs on more than 8 paths, the overhead is from 20% to 50% for memory dominated programs. Even this overhead is reasonable since the working frequency of the design does not change and we are able to run the whole system under the original frequency even in the presence of the variation.

VI. Conclusions

We proposed a resilient architecture for shared-L1 processor clusters. The architecture uses a single-cycle interconnection network between processors and a unified Tightly Coupled Data Memory (TCDM). By adding one stage of the controllable pipeline on the processor to memory paths we are able to mitigate the static delay variations. If there is no variation, the processor to memory path is fully combination. If the variation occurs, the controllable pipeline is switched to the pipeline mode and by increasing the latency of read/write operation we mitigate the effect of the variations.

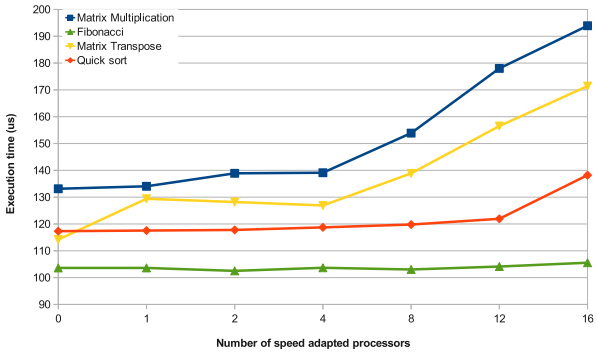

Fig. 8. Performance overhead due to the speed adaptation on 4 benchmark programs.

Acknowledgment

This work is supported by JTI (ENIAC) Modern project (GA n. 120003).

References

[1] A. Rahimi, I. Loi, M.R. Kakoee, L. Benini "A Fully-Synthesizable Single-Cycle Interconnection Network for Shared-L1 Processor Clusters ," in Proc. of the ACM/IEEE DATE, 2011.

[2] Plurality Ltd. The HyperCore Processor. www.plurality.com/hypercore.html

[3] ST Microelectronics and CEA. Platform 2012: A Many-core programmable accelerator for ultra efficient embedded computing in nanometer technology. 2010.

[4] NVIDIA. Next Generation CUDA Compute Architecture: Fermi - WhitePaper. www.nvidia.com/content/PDF/ fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2010.

[5] Tilera Corp. Product Brief. Tilepro64 processor. 2008.

[6] S. Akram, R. Kumar, D. Chen, "Workload adaptive shared memory multicore processors with reconfigurable interconnects," in Proc. of the 7th IEEE Symposium on Application-Specific Processors, SASP, July 2009, pp. 7-14.

[7] S. Mitra, K. Brelsford, Kim Young Moon, et al., "Robust System Design to Overcome CMOS Reliability Challenges," Emerging and Selected Topics in Circuits and Systems, IEEE Journal on , vol.1, no.1, pp. 30-41, March 2011

[8] S. Borkar,"Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," IEEE Micro, Vol. 25, No.6, pp. 10-16, 2005.

[9] K. Bowman, et al., "Circuit techniques for dynamic variation tolerance," in ACM/IEEE DAC, pp. 4-7, 2009.

[10] K.A. Bowman, J.W. Tschanz, et.al, "A 45 nm Resilient Microprocessor Core for Dynamic Variation Tolerance," Solid-State Circuits, IEEE Journal of , vol.46, no.1, pp.194-208, Jan. 2011

[11] D. Bull, S. Das, et al., "A Power-Efficient 32 bit ARM Processor Using Timing-Error Detection and Correction for Transient-Error Tolerance and Adaptation to PVT Variation," Solid-State Circuits, IEEE Journal of , vol.46, no.1, pp.18-31, Jan. 2011.

[12] D. Ernst, et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in Micro-36, pp. 7-18, 2003.

[13] M. R. Kakoee, L. Benini, "Fine-Grained Power and Body-Bias Control for Near-Threshold Deep Sub-Micron CMOS Circuits," Emerging and Selected Topics in Circuits and Systems, IEEE Journal on , vol.1, no.2, pp.131-140, June 2011.

[14] A. Ghosh et al., "A centralized supply voltage and local body biasbased compensation approach to mitigate within-die process variation," in ACM/IEEE ISLPED, 2009, pp. 45-50.

[15] http://www.gaisler.com

[16] M.R. Kakoee, V. Bertacco, L. Benini, "A distributed and topology-agnostic approach for on-line NoC testing," Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on , pp. 113-120, 1-4 May 2011.

[17] E. Cota, F.L. Kastensmidt, et al., "A High-Fault-Coverage Approach for the Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip," IEEE Trans. on Computers, Vol. 57, No. 9, pp. 1202-1215, 2008.

[18] M.R. Kakoee, Igor Loi, L. Benini, "A new physical routing approach for robust bundled signaling on NoC links," Proceedings of the 20th symposium on Great lakes symposium on VLSI, pp. 3-8, 2010.

[19] http://www.st.com