Zero-Performance-Overhead Online Fault Detection and Diagnosis in 3D Stacked Integrated Circuits

Saleh Safiruddin, Mihai Lefter, Demid Borodin, George Voicu, Sorin Dan Cotofana
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Delft, The Netherlands
{s.safiruddin, m.lefter, d.borodin, g.r.voicu, s.d.cotofana}@tudelft.nl

***Abstract*—In this paper we present a zero-performance-overhead online fault detection and diagnosis scheme that exploits the vertical proximity of hardware inherent in 3D stacked integrated circuits (3D-SIC). We consider a 3D stacked processor executing independent instruction streams from different threads, on each die. We propose the vertical clustering of functionally identical computational blocks in order to enable the utilization of the 3D specific low-latency interlayer communication infrastructure. The clustering facilitates the parallel re-execution of instructions on idle units located in the proximity of the units which initially computed them and in this way creates the means for fault diagnosis and detection. We detail the control, interconnection communication infrastructure, instruction distribution, and results processing policies required for our scheme. To determine the effectiveness of the approach, we evaluate its performance in terms of diagnosis latency and percentage of verified operations on 3 to 8 core processors implemented on 3 to 8 tier 3D-SICs, respectively, by means of simulations. Our experiments indicate that the diagnosis latency ranges from 9 to 5 cycles, for 3 to 8 cores, respectively. For transient fault detection our simulations indicate that 86% to 94% of all executed instructions are verified, for 3 to 8 cores, respectively. When only one of the layers is protected against transient faults the number of verified operations increases to 94% to 99%, for the same simulation conditions. This suggests that, if certain conditions are fulfilled at design time, our approach can completely protect one instruction stream identified as being critical for the application. Our simulations clearly indicate that the proposed scheme has the potential to improve the 3D stacked integrated circuits dependability with no performance overhead and at the expense of little area overhead.**

I. Introduction

Modern semiconductor industry faces severe scaling problems while attempting to keep pace with Moore's Law [1]. In particular, major challenges include the increasing communication delay [2] due to long interconnect wires, and maintaining the yield and lifetime of chips [3], [4]. Interference from external physical phenomena starts becoming an important failure factor, resulting in transient errors in logic circuits becoming a concern [5], [6], [7]. For future technologies, transient errors are expected to be very significant, and could even last multiple cycles [8], [9].

The emerging three-dimensional stacked integrated circuit (3D-SIC) technology [10] has the potential to alleviate some of these problems as it significantly contributes to overcoming the interconnect scaling barriers, by providing designers with the third dimension [11], [12]. The third dimension enables them to replace long intra-die interconnect wires with significantly shorter inter-die (vertical) communication channels implemented with, e.g., Through-Silicon Vias (TSVs). By making use of the z-dimension the 2D circuits can be folded at different granularities, such as stacking processors, functional units, or splitting a functional unit across different dies[1] [11]. Thus, a 3D stacked implementation of any wire-dominated circuit can significantly outperform the 2D counterparts in terms of performance [13] and energy consumption [14].

In addition to the straightforward benefits achievable by replacing long wires with shorter ones in existing systems, 3D integration opens new design opportunities. These opportunities can be leveraged to even further improve the system performance measured in terms of, e.g., latency, throughput, energy, and reliability. For example, dies produced with different fabrication technologies can be stacked together. This enables, for instance, energy effective and computationally efficient 3D stacked hybrid platforms [15].

As another example, the low-latency vertical communication channels between dies with processors and dies with caches allow for the placement of cache banks closer to the processors which make use of them, and thus by implication, for the reduction of the cache access time [16].

Furthermore, 3D-SIC technology extends the design space for integrated circuits [17], offering many new opportunities for dependable computing, to address the other major semiconductor industry problem: chip yield and dependable operation. 3D integration adds two new dimensions to the design space: (i) the z-dimension, as now the application can be mapped on parts of the circuit that are placed in different planes, and (ii) the R-dimension as different planes can be implemented with different reliabilities. Possible 3D-SIC dependability related research avenues, identified in [17], are 3D enhanced verification, 3D critical system part protection, 3D vulnerable resource protection, 3D check-pointing, and exploiting vertical proximity of hardware.

Following the 3D critical system part protection principle, in [18], the authors adapt the DIVA approach [19] for 3D systems. In [20] the vertical proximity of hardware is exploited and functional unit sharing is proposed that makes use of fast

[1]In this paper, to specify one die in the 3D stacked IC, we use the terms die, tier, and layer interchangeably.

vertical access between these resources in order to improve the system reliability and/or performance.

In this paper we introduce a novel dependability improvement approach, which builds upon the effective resource sharing between 3D stacked processors. We consider a 3D stacked processor executing an independent instruction stream on each die. The instruction streams can originate from different threads or tasks, for example. For improving system dependability, we wish to verify whether these instructions were executed correctly and if not to localize any permanent or transient faults where required. In 2D processors, this was achieved by introducing some sort of redundancy into the system, e.g., space or time redundancy, incurring significant area or performance overheads. In our 3D approach we do not rely on redundant resources as we propose to take advantage of the fact that in a 3D-SIC, identical resources, e.g., functional units, can be made to reside in vertical proximity to each other. Since the executed instruction streams are independent, they do not have the same resource utilization (occupancy) profile. This provides opportunities for the parallel re-execution of instructions on idle units located in the proximity of the unit which initially executed them. In this way the instruction results can be used to detect and locate faults.

We propose a system level diagnosis scheme which applies the above concept. The method does not require the presence of redundant hardware, and furthermore, no performance penalty is incurred, since only idle resources are utilized. The scheme runs online, i.e, it functions in the background during normal application execution. The 3D-SIC has to be organized so that we can make effective use of the available resources. This includes 3D vertical clustering; a definition of a global or local controller, and the necessary physical TSV-based interconnection infrastrucure that should exist; policies for instruction distribution; and policies for results selection and collection.

To evaluate the implications of the proposed technique we simulate our diagnosis scheme in the context of a 3D embodiment of a multicore system, including all the infrastructure and mechanisms required by our proposal. Our simulation results indicate that the diagnosis latency ranges from 9 to 5 cycles, for 3 to 8 cores, respectively. For transient fault detection our simulations indicate that 86% to 94% of all executed instructions are verified, for 3 to 8 cores, respectively. When only one of the layers is protected against transient faults the number of verified operations increases to 94% to 99%, for the same simulation conditions. This suggests that, if certain conditions are fulfilled at design time, our approach can completely protect one instruction stream identified as being critical for the application.

The paper is organized as follows. In Section II we present the main concepts behind the proposed diagnosis scheme, and the organization required for our scheme to function. In Section III we discuss fault detection time theoretical bounds of the scheme and in Section IV we evaluate the proposal for a number of benchmarks, by applying our scheme for a case study. Section V concludes the paper.

II. Proposed Online Diagnosis and Fault Detection Scheme for 3D-SICs

In this section we first describe the principle behind the proposed online diagnosis and fault detection scheme. We then detail the structure and requirements of the scheme.

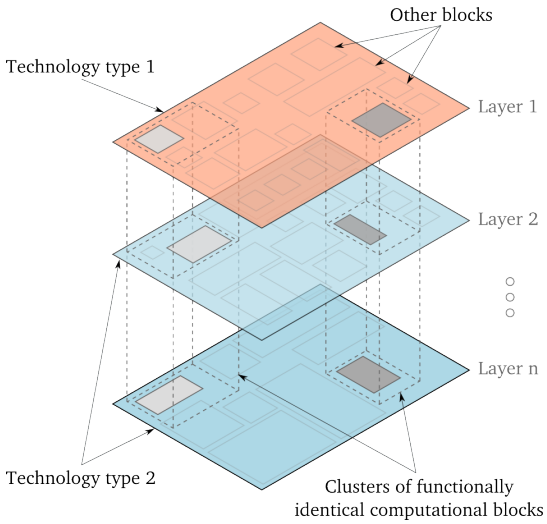

Fig. 1. Composing clusters of functionally identical computing blocks to which the diagnosis scheme can be applied. Interconnects are present but not drawn.

A. Main Concept

Let us consider a 3D stacked processor executing an application. The application can consist of multiple threads, with each thread supplying an independent instruction stream for each die. For improving system dependability, we wish to verify whether these instructions were executed correctly. The independent resource utilization of each thread provides opportunities for the parallel re-execution of instructions on idle units located in the proximity of the unit which initially executed them. In this way the instruction results can be used to detect and locate faults.

Our proposed diagnosis scheme utilizes the above concept and runs in the background simultaneously with the system applications, being completely transparent from the software perspective. Once the scheme raises a red flag, i.e, faulty instruction execution is discovered, the system can take appropriate action. For transient faults, the application may require execution verification of all the threads, or only for a specific, critical one. It may only need fail-safe operation or guarantee execution, by check-pointing for example. Furthermore, it may be required to locate the thread in which the fault occurred. For permanent faults, another execution on the faulty hardware may have to be prevented. For example, the faulty hardware can either be disabled, or self-repair can be applied, with a built-in spare being activated.

Depending on what the application diagnosis requirements are, the 3D-SIC has to be organized in a certain way so that the scheme can make effective use of the available resources. First, the identically functioning groups of hardware should

be placed in vertical proximity across a 3D stack. A physical interconnect communication infrastructure is required through which to distribute instructions and to collect results. 3D-SIC offers us a low latency communication channel through TSVs. Furthermore, each die requires additional hardware to interact with the communication infrastructure and apply the diagnosis scheme. The controller for each block can be global or local. Protocols are defined for instruction distribution as well as for results selection and collection.

B. 3D Vertical Clustering

Let us assume a 3D stacked organization composed of multiple processing systems, e.g., System-on-Chip, possibly with multicore processor, or a systolic array of processing elements. We propose that during the floor-planning phase, the organization is carefully partitioned across different layers in such a way that computational blocks performing identical operations are placed directly on top of each other. Multiple clusters are formed in this manner, composed of blocks that are spread over different layers, as it is presented in Figure 1.

It is important to mention that computational blocks are grouped inside a cluster based only on their identical functionality and location. Their hardware implementation can be totally different. Moreover, these computational blocks can be part of systems that perform different tasks, being implemented using the same, or different type of technology.

C. Global/Local Diagnosis Control

The proposed clustering organization exploits the vertical proximity advantage that is inherent in the 3D arrangement and facilitates the application of different reliability techniques. The generic structure of one vertical cluster based reliability system, consisting of n layers, is depicted in Figure 2. The main general blocks are highlighted.

The control of the applied reliability techniques can take place at the global level, at the local level, or at both local and global levels. The figure presents the general functional description of the system. It can be noticed that both the locals and the global controllers are present. However, in the case of a system in which the control takes place only at the local level, the global controller can be disregarded from the picture. The same holds true for the local controllers in case the control takes place only at the global level. Nevertheless, some special blocks are required in all the cases.

The related reliability data and information, such as operations and results, are transmitted through a common inter-layer communication infrastructure. In each layer, a Dispatcher is attached to this inter-layer communication infrastructure. Its main role is to gather the originally executed operation and to make sure that it is executed when the computational block is idle. This involves some switching mechanism of the usual inputs. Once the computational block has completed the operation the relevant information is put on the communication infrastructure by the Transmitter block.

In a system with global control unit, as the name suggests, the entire reliability mechanism is controlled at the global

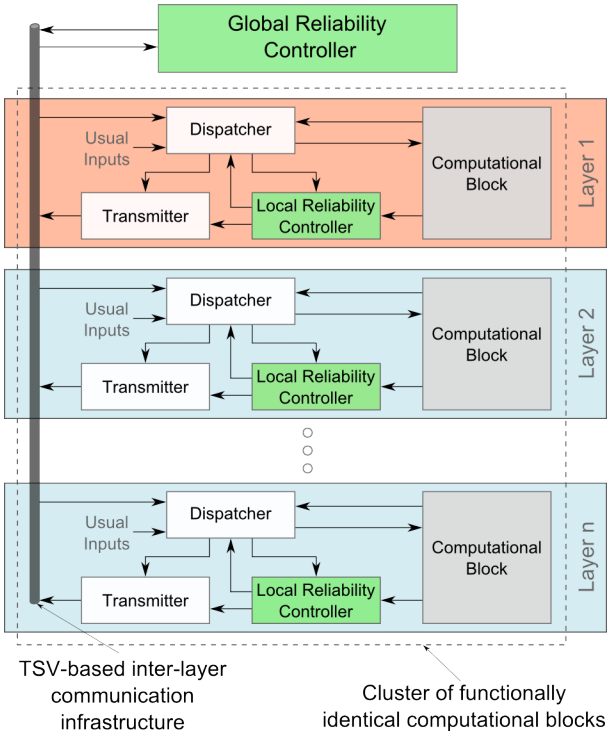

Fig. 2. Generic reliability framework required for diagnosis process.

level. The global controller determines what kind of techniques are applied, and at which time. Furthermore, it can either perform a reliability analysis of all the computational blocks in the cluster, or the ones in specific dies. It distributes the operations throughout the blocks in a vertical cluster, for execution when the respective block is idle. Based on the received results corrective actions can be taken.

In contrast to the global control, in the local control each computational block has a reliability controller which takes care only of its attached computational block.

D. Interconnect Infrastructure Requirements and Options

Communication inside clusters requires a TSV based infrastructure in order to assure a low latency interconnection for our proposal. Two aspects need to be considered in terms of infrastructure access: (i) how are the clusters connected to this infrastructure, and, (ii) how is the sharing of the infrastructure among the clusters implemented. For (i), the optimum wire length delay penalty should be determined at the layout level. For (ii), the interconnection can be seen as a bus or as a NoC segment. Figure 3 presents the possible options.

In the first option, Figure 3(a), we consider that the existing interlayer communication channels are utilized. Blocks from each cluster are connected to the already present blocks that perform data transfers between layers. The communication scheduling mechanism needs to be modified and adapted to the fact that a new data stream dedicated to the diagnosis has to be accommodated. This approach would adversely affect the performance of the fault detection and diagnosis methods but would entail no TSV overhead.

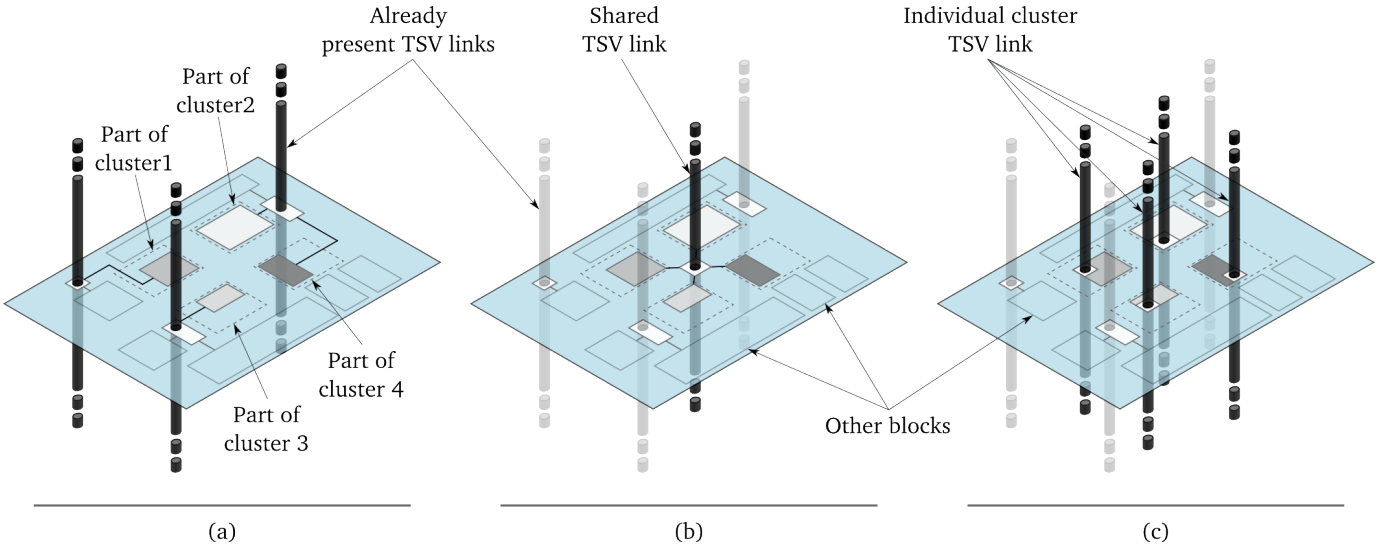

Fig. 3. Different types of interconnection: (a) using the existing infrastructure; (b) new infrastructure shared by multiple clusters; (c) each cluster with its own communication infrastructure.

The second option, Figure 3(b), assumes a dedicated TSV bus that is to be shared by different clusters. As in the previous option, an extra delay appears due to the fact that the interconnection infrastructure is shared. However, this type of sharing allows more access to the link than the previous option.

In the later option, Figure 3(c), each cluster has a dedicated TSV-based bus through which to pass data. This obviously requires a large overhead in TSVs. Nevertheless, it may be the preferred choice in terms of performance.

Finally, there is also the possibility of combining the above presented intercommunication strategies.

E. Instruction and Data Distribution and Collection

The data that are transferred onto the interconnection infrastructure consist of: (i) the data required by the clustered blocks in order to be able to execute the diagnosis operation, and (ii) the data that hold the diagnosis related information. Therefore, two types of propagation can be distinguished, namely "instruction data propagation", that propagates the former sort of data, and "'diagnosis information propagation" that propagates the latter. The instruction data propagation is a one-to-many distribution, while the diagnosis information propagation is usually a many-to-one distribution.

We introduce the concept of "initiator" block/die/layer, whose purpose is to select a diagnosis instruction and to acquire the data required by the instruction data propagation. The selected diagnosis operation can be either an already executed one or an instruction that is just to commence executing. In Section II-F there is a more detailed discussion about the implications of these types of instruction selecting methods. The "initiator" die can be static, where it is always the same die, or dynamic, where periodically the "initiator die" is cycled among all the dies.

We propose in Figure 4 two methods by which the data can be distributed.

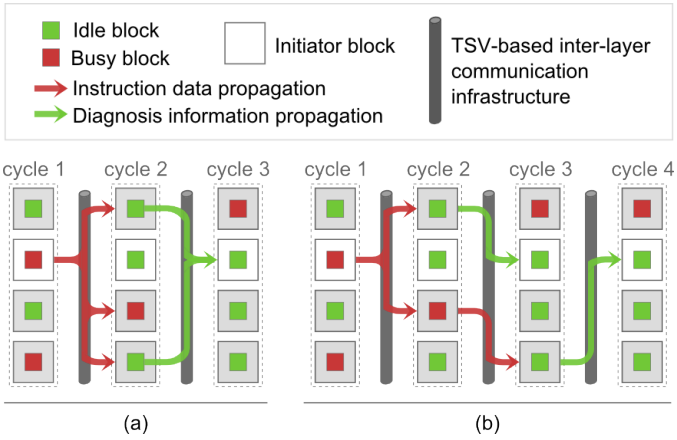

Fig. 4. Instruction data sending by (a) broadcasting to the entire vertical block; instruction data accepted only by idle functional units, and (b) cascading outwards.

- Broadcast - Consists in sending in the same moment the data to all the involved computational blocks in the cluster. Both the instruction data propagation and the diagnosis information propagation are finished each in one shot. The broadcast frequency period depends on the diagnosis requirements. In terms of communication infrastructure, a dedicated TSV-bus as in Figure 3 suits the method the best. The broadcast method is depicted schematically in Figure 4(a).
- Cascade - Consists in propagating the data to computational blocks in the vertical cluster on adjacent dies. If the hardware is available at the time the data are received, then the operation is performed, otherwise the data are propagated to the next block in the cluster. In terms of communication infrastructure, cascading does not have such great demands as broadcasting. A shared channel

such as in Figure 3(a,b) can usually accommodate the requirements. The cascading method is depicted schematically in Figure 4(b).

The diagnosis information propagation from Figure 4 is directed towards the initiator, but this may not always be the case. The reason is that the initiator is not always the die that also processes the diagnosis information.

F. Results Processings Policy

Our approach consists in making idle computational blocks execute identical instructions. Later on, the produced results need to be compared in order to be able to assess the faulty behavior of the involved blocks. Regarding the comparison operation, we can distinguish two possibilities: either the comparison operation is performed locally and its result in the form of an agree bit is propagated to the initiator block (therm introduced in Section II-E), or the result of the checked operation is propagated to the initiator, which needs to perform a global comparison itself.

The global comparison scheme proves to be more expensive in terms of interconnection requirements. The number of results that need to be propagated increases with the number of layers in the cluster, counterbalancing the fact that only the operands are sent in the instruction data propagation phase. Another drawback of the global comparison scheme arises from the computational complexity pressure that is put on the initiator to perform all the comparisons when the broadcast distribution mechanism is utilized. However, with cascading distribution the global comparisons can be serialized and the hardware requirements are thus reduced. According to Figure 4(b) at most two results are coming each cycle.

To be able to detect the faulty block in the case in which the initiator was not the targeted one, some identification needs to be attached to the propagated reliability information. This can be achieved with the global comparison scheme and it adds even more to the communication infrastructure. Different protocols that employ encoding schemes can be derived in order to minimize the communication requirements.

Once the results processing is complete, the system can take steps to remedy the problem. In diagnosis mode, if a permanent error is detected and located, some action has to be taken to prevent another execution on the faulty hardware. A number of actions can be undertaken, for example, the faulty hardware can either be disabled, or a spare which was chosen to be built in, can be activated. In detection mode, if a transient error is detected, some sort of recovery will have to take place. This could be in the form of Forward Error Recovery, e.g., some sort of redundancy, or in the form of Backward Error Recovery, e.g., utilizing check-pointing, or program re-execution.

G. Example: 3D Multicore Processor Functional Unit Clustering

Let us consider a cluster composed of n-bit integer Arithmetic Logic Units (ALU). In order for those to function in the 3D diagnosis context they have to be augmented as indicated in Figure 5 for the i die. We assume that local comparison is applied, as presented in Section II-F. Thus, the ALU instruction, the operands, and the computed result have to be send over the ALU cluster.

From the figure it can be noticed that the usual ALU inputs coming from the register file are multiplexed with the received operands. At the same time, the usual instruction opcode coming from the instruction decoder is also multiplexed with the received opcode. The muxes are controlled by the newly introduced control logic which, in addition, stops the pollution of the register file by selecting the right output of the ALU. Typically, an ALU already contains a built-in comparator which can be used to perform the comparison required by our proposed mechanism. The agree bit is thus generated with no comparison hardware overhead. In the case of cascading data propagation, the required receiver outputs are forwarded to the transmitter. Furthermore, the block in the figure can also act as an initiator.

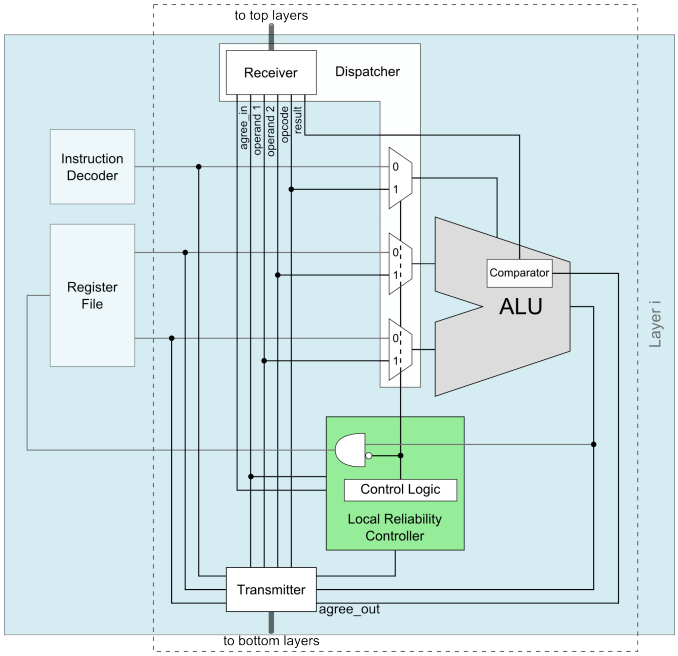

Fig. 5. Appended hardware for n-bit integer ALU clustering

Essentially, TSV lines need to accommodate sending three n-bit data (two operands and the result), as well as the opcode and finally the outputs from the comparison of the two results. The amount of TSV links required between 2 dies is equal to $3n + m + 1$, where n is the number of bits for the operands and result, m is the number of bits for the opcode.

III. Theoretical Analysis

In this section we model our scheme to analyze the performance upper bounds given varied hardware occupancy.

A. Utilizing 3D Vertical Clustering for System Diagnosis

In this subsection, we present a theoretical evaluation for the utilization of our scheme to detect and locate permanent faults.

In order to guard against a misdiagnosis of a transient fault as a permanent fault, each computational block is assigned a so-called error counter. As the diagnosis runs its course, each time a computational block is found to have executed an operation incorrectly, the error counter is incremented. The counter can be implemented as hardware, or as a variable in the operating system kernel. After a certain threshold the computational block is flagged to be faulty. This technique ensures that the probability of a misdiagnosed transient error is negligible. Thus, in the event of a permanent fault finally being diagnosed, the time required to diagnose the fault will depend on the chosen threshold value.

We model our scheme, with dedicated infrastructure for each vertical cluster and the broadcast method to distribute instructions and collect results. We model the best case scenario, and thus we get upper bound results. The hardware resource occupancy is varied from 1% to 99% with the occupancy being randomly distributed. The proposed mechanism is then applied and the amount of cycles required to diagnose the fault is measured. Figure 6 depicts the relation between the occupancy of resources and the time required to diagnose a fault. In the graph we observe that the more layers there are in the 3D stack, the faster the diagnosis of the fault. Furthermore, at 90% occupancy the diagnosis time is 13 cycles for 4 layers and 5 cycles for 64 layers, respectively, and substantially increases for occupancies larger than 99%.

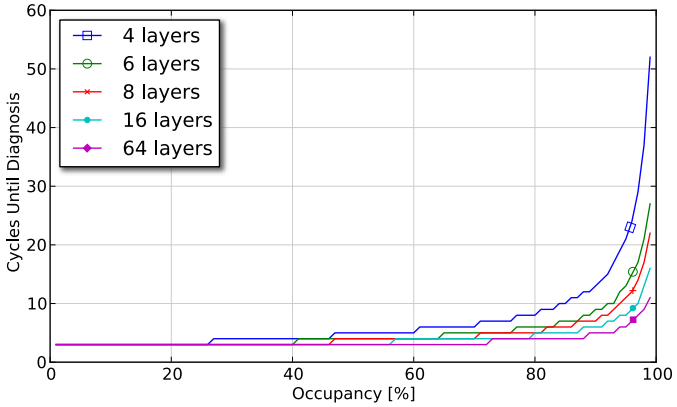

Fig. 6. Permanent fault diagnosis latency

B. Utilizing 3D Vertical Clustering for Fault Detection

In this subsection, we present a theoretical evaluation for the utilization of our scheme to detect transient faults. We model our scheme, with dedicated infrastructure for each vertical cluster and the broadcast method to distribute instructions and collect results. The best case scenario is modelled, and thus we get upper bound results. The hardware resource occupancy is varied from 1% to 99% with the occupancy being randomly distributed. The proposed mechanism is then applied and the percentage of operations that are verified by re-execution is measured. First. we model for the case where all threads are protected. In Figure 7 the relation between occupancy of resources and the percentage of protected operation is presented. In the graph we observe that for an occupancy of 20% almost all instructions can be verified, irrespective of the amount of layers in the stack. For 60% occupancy, the percentage ranges from 54% to 66%, for 3 to 64 layers, respectively. For occupancies greater than 70% the amount of layers does not affect the percentage of protected executions.

Next, we model for the case where one layer, designated as critical, is selected for protection by other layers. In figure 8 we observe that for 60% hardware occupancy, 63% to 98% of operations are verified, in a stack with 3 to 16 layers, respectively. In both cases we observe that the more layers there are in the 3D stack, the higher the percentage of operations that can be verified.

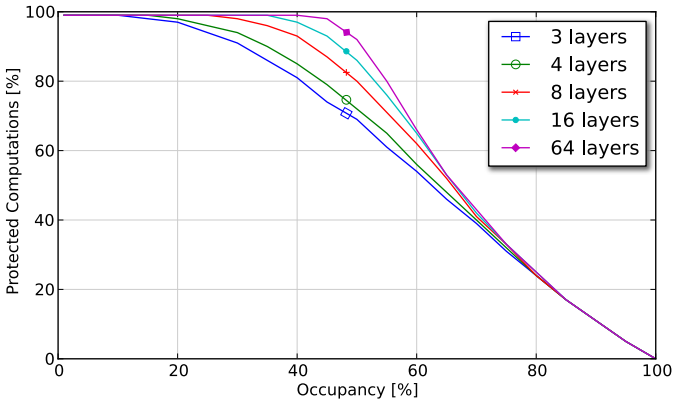

Fig. 7. Percentage of computations protected against transient errors

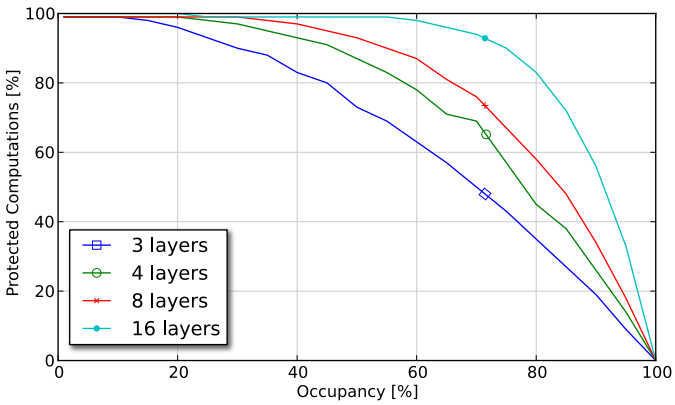

Fig. 8. Percentage of computations protected against transient errors for dedicated layer

IV. Case Study: Functional Unit Diagnosis and Fault Detection in a 3D Multicore Processor

In this section we evaluate the potential practical implications of our proposal by means of a simple case study. We consider a multicore processor composed of identical cores. The multicore is split across several dies, such that one core is placed on each die, and the identical functional units are in vertical proximity. Each vertical stack of functional units constitutes a group and each stacked group functions

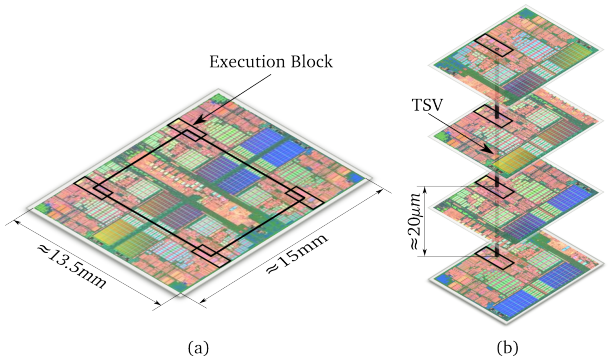

Fig. 9. Wiring difference between (a) 2D multicore arrangement and (b) 3D vertically placed multicore arrangement

TABLE I
BASE PROCESSOR CONFIGURATION

Issue	out-of-order
Fetch/Dec./Issue Width	4
# of Int. ALUs	4
# of Int. Mult./Div.	1
# of FP ALUs	1
# of FP Mult./Div.	1
RUU Size	64
LSQ Size	32
Memory Latency	112 cycles (first chunk), 2 cycles (subsequent chunks)
L1 Instruction Cache	32 KB, 2-way set associative
L1 Data Cache	32 KB, 2-way set associative
L2 Unified Cache	512 KB, 4-way set associative

independently from other stacked groups, for our purpose. Furthermore, each stacked group utilizes the results internally for determining which units are faulty. In Figure 9 we can see an example of a quad-core processor, with two arrangements. The layout is based on the micrograph die photo of the 65nm AMD Opteron processor [21]. In the figure, the blocks that include the functional units are highlighted with black squares.

To be able to determine the applicability of our aproach in the case of functional unit clustering, we have initially performed an evaluation of their occupancy behaviour. This evaluation consisted of determining the total numer of cycles when the functional units are busy/idle, as well as determining the distribution of the consecutive busy/idle cycles within the entire execution time.

The evaluation was performed on a modified version of the SimpleScalar tool set [22], with the base processor configuration being described in Table I. Several benchmarks were run from both the Mediabench [23] and SPEC2000 [24] suites: JPEG compression, MPEG2 video compression, GSM sound (voice) compression, GCC compiler, BZIP compression, ART image recognition, MCF combinatorial optimization, MESA 3-D graphics, and VORTEX database. In the chosen out-of-order processor configuration four integer Arithmetic Logic Units (ALUs) are present. Computations are being allocated to each ALU in a round-robin manner.

The obtained results show an average occupancy of 42%, for all the considered benchmarks, with a standard deviation of 18.2%. The maximum occupancy was 64.4%, while the minimum was 13.4%. This indicates that there is a sufficient number of underutilized integer ALUs to assure redundant execution of operations received from remote CPUs. By corroborating these results with the ones from Figure 7, we observe that on the average more than 80% of the instructions can be protected against transient errors by utilizing our proposed mechanism.

In Figure 10 the distribution of different consecutive busy/idle cycles is presented. This part of the figure is quite revealing in several ways. First, it can be noticed that functional units are busy between 1 and 4 consecutive cycles for the majority of the time (85% on average). Therefore, it is apparent that there is enough room for redundant operations to be executed in a relatively short time interval. Secondly, a functional unit is for the majority of the time idle for more than two consecutive cycles (65% on average). This leaves space for redundant operations from different layers to be executed in those consecutive cycles.

In a real multicore system, cores can run different applications concurrently, and the occupancy rates in each core

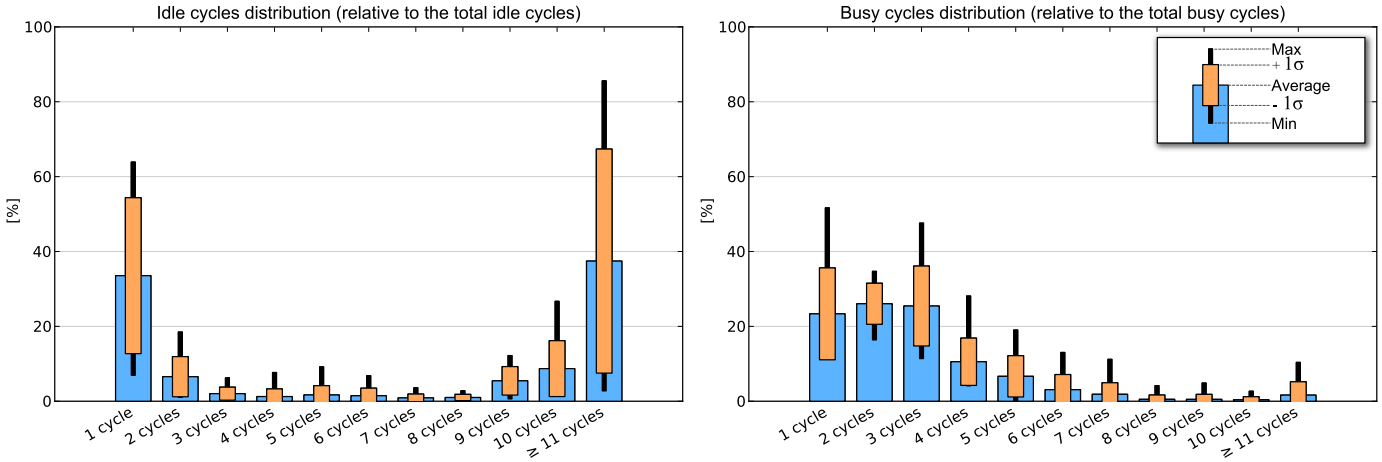

Fig. 10. Distribution of consecutive idle/busy cycles

depends heavily on which application is running. We simulate a multicore running different applications and run above mentioned benchmarks on the cores. In Figure 11 we can see the results for the broadcast mechanism. For the benchmarks executed on 3 to 8 cores, the range of the hardware occupancy was between 34% and 42%. We can see in Figure 11(a) that our scheme detects and locates the fault within 9 cycles on average, for 3 cores, and approaches 4 cycles for increasing amount of cores. In Figure 11(b) the percentage of protected instructions reaches 94% for 8 cores , where the entire system is being protected; and in Figure 11(c) this percentage rises to 99% for 8 cores, where one critical die is being protected.

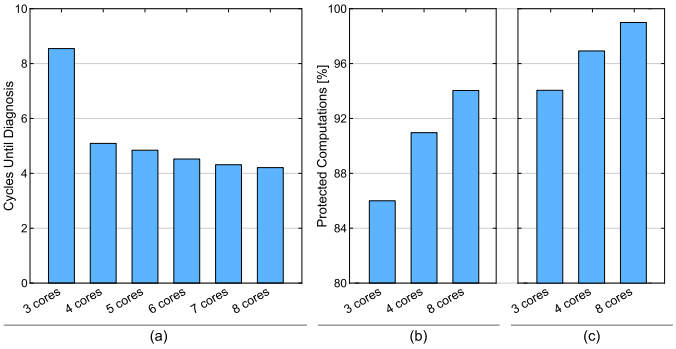

Fig. 11. Performance of our diagnosis scheme for (a) permanent fault diagnosis, (b) protection of all threads, and (c) dedicated thread protection

V. Conclusion

In this paper we proposed a 3D system dependability improvement approach which leverages the vertical proximity of identical resources. Idle resources in vertical clusters are utilized to verify the performed operations. Based on the results from redundant execution the online error detection and location is enabled. No performance overhead is introduced, because only idle resources execute redundant operations.

We evaluated by theoretical analysis and simulation the permanent error detection latency, as well as the transient fault coverage, achieved by the proposed approach. With the resource occupancy indicated by the used benchmarks, the proposed approach is feasible both for detection and location of permanent faults (enabling online fault diagnosis and graceful system degradation) and for transient fault detection (enabling a fail-safe operation).

References

[1] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, April 1965.

[2] R. Ronen, S. Member, A. Mendelson, K. Lai, S. lien Lu, F. Pollack, John, and J. P. Shen, "Coming Challenges in Microarchitecture and Architecture," in *IEEE Proc.*, 2001, pp. 325–340.

[3] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *International Conference on Dependable Systems and Networks*, 2004, pp. 177–186.

[4] ——, "Lifetime reliability: Toward an architectural solution," *IEEE Micro*, vol. 25, no. 3, pp. 70–80, 2005.

[5] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, and T. Turflinger, "Digital single event transient trends with technology node scaling," *IEEE Transactions on Nuclear Science*, vol. 53, no. 6, pp. 3462–3465, Dec. 2006.

[6] I. Polian, J. Hayes, S. Reddy, and B. Becker, "Modeling and mitigating transient errors in logic circuits," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 4, pp. 537–547, 2011.

[7] P. Dodd, M. Shaneyfelt, J. Felix, and J. Schwank, "Production and propagation of single-event transients in high-speed digital logic ICs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3278–3284, Dec. 2004.

[8] H. Yu, X. Fan, and M. Nicolaidis, "Design trends and challenges of logic soft errors in future nanotechnologies circuits reliability," in *9th International Conference on Solid-State and Integrated-Circuit Technology*, 2008, pp. 651–654.

[9] C. Lisboa, M. Erigson, and L. Carro, "System level approaches for mitigation of long duration transient faults in future technologies," in *12th IEEE European Test Symposium*, 2007, pp. 165–172.

[10] G. Katti, A. Mercha, J. Van Olmen, C. Huyghebaert, A. Jourdain, M. Stucchi, M. Rakowski, I. Debusschere, P. Soussan, W. Dehaene *et al.*, "3D stacked ICs using cu TSVs and die to wafer hybrid collective bonding," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, 2010, pp. 1–4.

[11] Y. Xie, G. H. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 2, pp. 65–103, 2006.

[12] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *MICRO-39: Proc. of the 39th Annual IEEE/ACM Int. Symp. on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 469–479.

[13] A. Rahman and R. Reif, "System-Level Performance Evaluation of Three-Dimensional Integrated Circuits," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 8, no. 6, pp. 671–678, 2000.

[14] J. W. Joyner and J. D. Meindl, "Opportunities for Reduced Power Dissipation Using Three-Dimensional Integration," in *Proc. of the IEEE 2002 Int. Interconnect Technology Conf.*, Burlingame, CA, USA, Jun 2002, pp. 148–150.

[15] M. Enachescu, G. Voicu, and S. D. Cotofana, "Is the Road Towards Zero-Energy Paved with NEMFET-based Power Management?" in *(To appear in) IEEE International Symposium on in Circuits and Systems, 2012.*, May 2012.

[16] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 130–141, 2006.

[17] S. Safiruddin, M. Lefter, D. Borodin, G. Voicu, and S. D. Cotofana, " Is 3D Integration the Way to Future Dependable Computing Platforms?" in *(To be published in) 13th International Conference on Optimization of Electrical and Electronic Equipment, 2012.*, May 2012.

[18] N. Madan and R. Balasubramonian, "Leveraging 3D Technology for Improved Reliability," in *MICRO-07: Proc. of the 40th Annual IEEE/ACM Int. Symp. on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 223–235.

[19] T. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," in *MICRO-32: Proc. 32nd Annual ACM/IEEE Int. Symp. on Microarchitecture*, Washington, DC, USA, Jun 1999, pp. 196–207.

[20] D. Borodin, W. Siauw, and S. D. Cotofana, "Functional Unit Sharing Between Stacked Processors in 3D Integrated Systems," in *IC-SAMOS XI: Proc. Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation*, July 2011, pp. 311–317.

[21] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, "An integrated Quad-Core opteron processor," in *IEEE International Solid-State Circuits Conference*, 2007, pp. 102–103.

[22] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[23] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, 1997, pp. 330–335.

[24] J. Henning, "SPEC CPU2000: measuring CPU performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.