An Adaptive Low-Overhead Mechanism for Dependable General-Purpose Many-Core Processors

Wentao Jia*, Rui Li, and Chunyan Zhang

School of Computer, National University of Defense Technology, China
{wtjia,lirui,cyzhang}@nudt.edu.cn

Abstract. Future many-core processors may contain more than 1000 cores on single die. However, continued scaling of silicon fabrication technology exposes chip orders of such magnitude to a higher vulnerability to errors. A low-overhead and adaptive fault-tolerance mechanism is desired for general-purpose many-core processors. We propose high-level adaptive redundancy (HLAR), which possesses several unique properties. First, the technique employs selective redundancy based application assistance and dynamically cores schedule. Second, the method requires minimal overhead when the mechanism is disabled. Third, it expands the local memory within the replication sphere, which heightens the replication level and simplifies the redundancy mechanism. Finally, it decreases bandwidth through various compression methods, thus effectively balancing reliability, performance, and power. Experimental results show a remarkably low overhead while covering 99.999% errors with only 0.25% more networks-on-chip traffic.

Keywords: Many-Core, Redundant Execution, Adaptive Dependable, Low-Overhead.

1 Introduction

Transistors continue to double in number every two years without significant frequency enhancements and extra power costs. These facts indicate a demand for new processors with more than 1000 cores and an increasing need to utilize such a large amount of resources [1]. As transistor size decreases, the probability of chip-level soft errors and physical flaws induced by voltage fluctuation, cosmic rays, thermal changes, or variability in manufacturing further increases [2], which causes unavoidable errors in many-core systems.

Redundant execution efficiently improve reliability, which can be applied in most implementations of multithreading such as simultaneous multithreading (SMT) or chip multi-core processors (CMP). Current redundant execution techniques such as SRT, CRT and RECVF [3] entail either high hardware costs such

* This work was Supported by the National Nature Science Foundation of China under NSFC No. 61033008, 60903041 and 61103080.

K. Mustofa et al. (Eds.): ICT-EurAsia 2013, LNCS 7804, pp. 337–342, 2013.

as load value queue (LVQ), buffer, comparer, and dedicated bus, or significant changes to existing highly optimized micro-architectures, which may be affordable for CMP but not for general-purpose many-core processors (GPMCP). Same core-level costs in many-core processors result in an overhead up to 10 or 100 times higher than that in CMP. High overhead may be reasonable for fixed high-reliable applications but not for all applications in GPMCP as the latter induce large overhead for applications that do not require reliability.

GPMCP present two implications. First, not all applications require high reliability. Second, the chip contains more than 1000 cores and some cores usually become idle. We proposed high-level adaptive redundancy (HLAR), an adaptive, low-overhead mechanism based application assistance and system resource usage for GPMCP. Our evaluation-based error injection shows that HLAR is capable of satisfying error coverage with minimal NoC traffic that covers 99.999% errors with 0.25% more NoC traffic.

2 Background and Related Work

Many-core architectures such as Tilera Tile64, Intel MIC, and NVIDIA Fermi, among others show good perspective. A challenge for many-core architecture is that hardware-managed Cache entail unacceptable costs. As alternative model, software-managed local memory (LM), exposes intermediate memories to the software and relies on it to orchestrate the memory operations. The IBM C64 is a clear example of a device that benefits from the use of LMs.

Applications in GPMCP are parallel and consist of threads with different reliability requirements. Some applications are critical. Wells [4] managed two types of applications: reliable applications and performance applications. Kruijf [5] argued that some emerging applications are error-tolerant and can discard computations in the event of an error.

The following have prompted this study: DCC [6] allows arbitrary CMP cores to verify execution of the other without a static core binding or a dedicated communication hardware. Relax [5] proposed a software recovery framework in which the code regions are marked for recovery. Fingerprinting [7] proposed compressing architectural state updates into a signature, which lowers comparison bandwidth by orders of magnitude.

3 HLAR Mechanism

Redundancy overhead. HLAR redundancy are not executed for the whole chip, thus, we define *redundancy entry* as cores that execute redundancy and *outside redundancy entry* as cores that do not execute redundancy. Considering processors without redundancy as the baseline, we classify redundancy overhead.

(i) Fixed overhead in all cores because of hardware cost or performance lost due to hardware modification (O_{FA}).

(ii) Special overhead in redundancy entry (O_{SRE}).

(iii) Temporal overhead in redundancy entry (O_{TRE}).

(iv) Overhead outside redundancy entry due to bandwidth and other shared resources (O_{ORE}). If redundancy cores utilize additional shared resources, other cores may cost more to obtain the resources. NoC bandwidth is a global resource and affects the performance of the die.

HLAR Architecture. HLAR is a low overhead GPMCP redundancy mechanism It supports redundancy between arbitrary cores and causes minimal modifications to the core. Fig.1 shows the HLAR architecture. HLAR employs a many-core processor interconnected with a mesh network. For each core, we added a redundancy control module called redundancy manager (RM). RM consists of small control circuits and two FIFO buffers. The RM area is small, compared with cores/networks. HLAR uses the existing chip networks in transferring trace data and supports input replication via a remote value queue (RVQ).

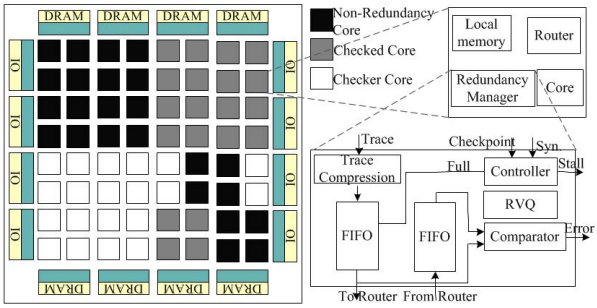

Fig. 1. HLAR architecture

HLAR cores can be a checked core, a checker core, or a non-redundancy core. Non-redundancy cores do not execute redundancy. In the checked core, the RM receives register updates from the core and writes these updates onto the sender FIFO. Typically, the register update information consists of address/value pairs (8bit/32bit). The RM interfaces with the NoC, compresses the trace data, and sends messages to the checker core. The RM of the checker core writes its compressed register updates into its sender FIFO. A comparator compares elements in the two FIFOs. When these two vary, the RM raises a redundancy error exception. The RM controller includes a simple state-machine that stalls the processor when the FIFOs become full.

Input Replication. Unlike the previous work, only the remote value but the load value requires replication in HLAR. HLAR supports input replication via RVQ. The checker core reads values from the RVQ rather than access the remote node. Programs and data in the checked core's local memory must be copied onto the checker core's, thus, address space replication is needed. A register named *reg_addr_remap* is used to replicate the address space.

Trace Compression. We employed CRC-1, CRC-5, and CRC-8 to obtain an adaptive compression rate, and found that these methods adequately satisfy

coverage. To obtain more effective compression, HLAR summarizes many updates for once CRC check. CRC-1/10 means one-bit CRC-1 check for every set of 10 trace values, hence, CRC-5/100, CRC-8/1000 and so on. NoC overhead is very small in HLAR(for example, CRC-8/1000 only costs 0.025% more bandwidth)

Recovery. Redundant execution is often combined with a checkpoint retry to gain recovery. However, the checkpoint overhead increases, especially if a short checkpoint interval is employed. HLAR indicates the *checkpoint_interval* for a configurable checkpoint mechanism. Aside from the checkpoint, a simple forward error recovery (FER) mechanism is employed, which discards incorrect results and continues to make progress.

Application Framework in HLAR. HLAR for applications can be as simple as system devices, in which only require configuring and enabling. The device views simple usage in applications and management in system. The application first configures HLAR through HLAR_config() and the control registers in RM are then set. When HLAR_enable() is prompted, the Hypervisor selects the appropriate core, copies the state from the checked core to initialize the checker core, and then begins the redundant execution. The hypervisor completes the redundancy and disables the RM until HLAR_disable() is called.

4 Evaluation

4.1 Methodology

HLAR is implemented based on the OpenRISC 1200 (OR1200) [8] core. The OR1200 configuration is used as the default parameter, except for the following: 8 KB local instructions Cache, 8 KB local data Cache, and 32 KB local memory.

Our experimental results are based on OR1200 cores in Altera Stratix IV FPGA prototyping boards. The primary advantage of using prototyping boards instead of simulation is speed. We evaluated the error coverage and latency by error injection and evaluated the overhead based hardware counters and EDA tools(Quartus II 11.0).

Workload and Fault Model. The applications evaluated include the following: MatrixMult (4M) and BubbleSort (5M). The fault model consists of single bit-flip (95%) and double bit-flip (5%). The number of experiments is 20,000 + 40,000 + 100,000 (fault sites for CRC-1, CRC-5, and CRC-8) * 6 (summarized interval) * 2 (applications) = 1,920,000.

Error Injection. One register from the 32 GPRs and 6 control registers (PC, SR, EA, PICMR, PICPR, and PICSR) were randomly chosen. Likewise, 1- or 2-bit out of 32-bit locations were randomly selected and flipped.

4.2 Experimental Results

Temporal Overhead in Redundancy Entry. O_{TRE} is usually shown as performance degradation. Performance degradation shown in Fig.2(a) is 0.71% for

MM and 0.68% for BS at 100,000 instructions of the checkpoint. These rates increase to 12.2% and 9.8% at 1000 instructions. When a discard recovery mechanism is employed, the degradation is negligible at 0.42% and 0.23%, as shown in Fig.2(b).

Fixed Overhead. The only fixed overhead in HLAR is RM. Logic utilization is shown in Fig.2(c). RM utilizes 359 combinational ALUTs and a 160 byte memory, which only use 2.46% and 0.25% of the total (OR1200 core and router). The fixed overhead in HLAR is much less compared with Reunion or RECVF.

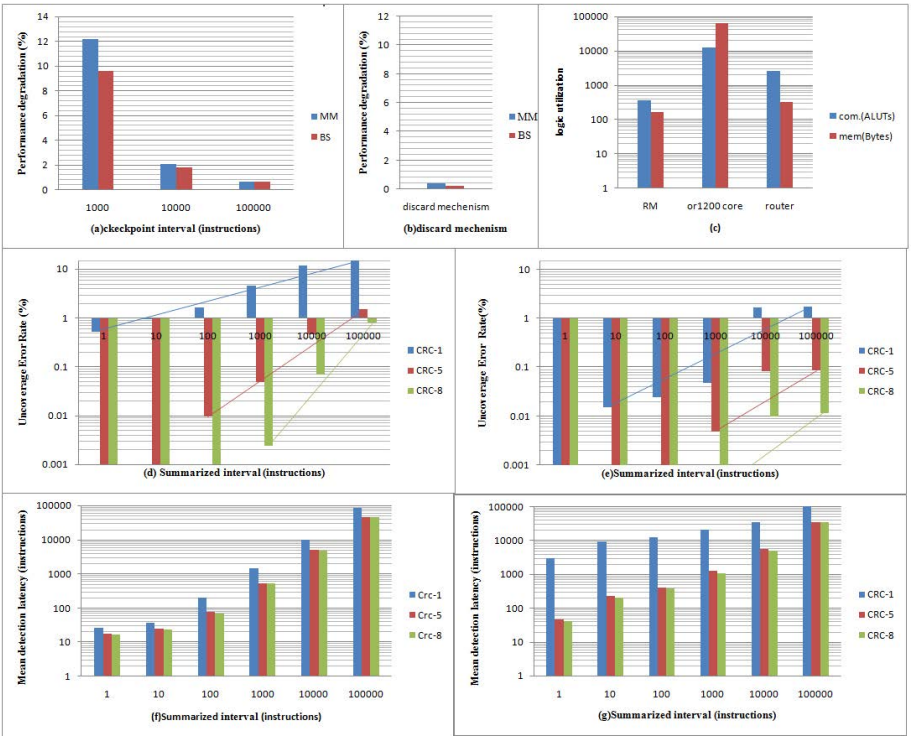

Fig. 2. Result: (a)Temporal overhead for checkpoint and for (b)discard mechanism; (c)Fixed overhead; (d)Error coverage rate for MM and (e)BS; (f)Mean detection latency for MM and (g)BS.

Error Coverage. HLAR compresses traces with CRC-1, CRC-5, CRC-8, and summarizes CRC to balance reliability and NoC performance. The uncoverage error rate is shown in Fig.2(d) and (e). Comparing traces without compression can obtain a 100% coverage. CRC-1/1 reduces bandwidth by up to 40 times without losing coverage, 0.53% for MM, and below 0.001% for BS. When the summarized interval increases by 10 times, uncoverage increases (denoted by a line) on a small scale. Uncoverages in CRC-5 and CRC-8 are low even at intervals of 10,0000 instructions; uncoverage rates are 1.5% and 0.8%, respectively,

for MM and are 0.086% and 0.012% for BS. As the interval decreases, uncoverage decreases significantly. An uncoverage rate below 0.001% in Fig.2(d) and (e) indicates that no SDC occurs after error injection. Minimal uncoverage (below 0.001%) occurs at 10 and 100 instructions in CRC-5 and at 100 and 1000 instructions in CRC-8 in MM and BS, respectively, which means only 0.25% or 0.025% more NoC traffic.

Detection Latency. NoC communicating with a distant core may incur greater latency than communicating with an adjacent core. The summarized CRC may also lead to larger latency. However, the results in Fig.2(f) and (g) show that the detected latency in HLAR is bounded. Mean error detection latency (MEDL) for MM is consistent with the summarized interval, increasing from 27 in CRC-1/1 to 86,490 in CRC-1/100000. CRC-5 and CRC-8 show lower MEDL than CRC-1. For instance, at the interval of 1000 instructions, MEDL is 1424 in CRC-1, 539 in CRC-5, and 515 in CRC-8.

5 Conclusion

We analysed the redundant execution overhead and proposed HLAR, an adaptive low-overhead redundancy mechanism for GPMCP. Unlike prior mechanisms, HLAR can sense application requirements and system resource usage to reconfigure redundancy. Thus, HLAR decreases the overhead by executing only the necessary redundancy and using the idle core for this redundancy. HLAR expands the local memory within the replication sphere, which provides relaxed input replication, distributes the memory access, and allows the core pairs to progress simultaneously. HLAR is capable of perfect error coverage with a minimal overhead, covering 99.999% of errors with less than 0.25% more commutation.

References

1. Borkar, S.: Thousand core chips: a technology perspective. In: Proceedings of the 44th Annual Design Automation Conference (June 2007)
2. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: The impact of technology scaling on lifetime reliability. In: Intl. Conf. on DSN (June 2004)
3. Subramanyan, P., Singh, V., Saluja, K.K., Larsson, E.: Energy-Efficient Fault Tolerance in Chip Multiprocessors Using Critical Value Forwarding. In: Intl. Conf. on Dependable Systems and Networks (June 2010)
4. Wells, P.M., Chakraborty, K., Sohi, G.S.: Mixed-mode multicore reliability. In: Intl. Conf. on ASPLOS (March 2009)
5. de Kruijf, M., Nomura, S., Sankaralingam, K.: Relax: An architectural framework for software recovery of hardware faults. In: ISCA (2010)
6. LaFrieda, C., Ipek, E., Martinez, J.F., Manohar, R.: Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In: Intl. Conf. on DSN (2007)
7. Smolens, J.C., Gold, B.T., Kim, J., Falsafi, B., Hoe, J.C., Nowatzyk, A.G.: Fingerprinting: bounding soft-error detection latency and bandwidth. In: Intl. Conf. on ASPLOS (October 2004)
8. Lampret, D.: OpenRISC 1200 IP Core Specification (September 2001), `http://www.opencores.org`