Variation-Aware Scheduling for Chip Multiprocessors with Thread Level Redundancy

Jianbo Dong*†, Lei Zhang*, Yinhe Han*†, Guihai Yan*† and Xiaowei Li*†

*Key Laboratory of Computer System and Architecture
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
†Graduate University of Chinese Academy of Sciences, Beijing, P.R. China
Email: {dongjianbo, zlei, yinhes, yan_guihai, lxw}@ict.ac.cn

***Abstract*—Thread-Level Redundancy in Chip Multiprocessors (TLR-CMP) is efficient for soft error tolerance. Process variation causes core-to-core (C2C) performance asymmetry across a chip, which should be taken into consideration for application scheduling. In this paper, two types of variations beyond C2C are introduced, i.e., inter-pair and intra-pair variation in TLR-CMP. Intra-pair performance asymmetry can affect the performance of applications differently. Based on the above observation, we firstly formalize the variation-aware scheduling in TLR-CMP as a 0-1 programming problem, to maximize the system weighted throughput. An efficient scheduling algorithm, named *IntraVarF&AppSen*, is then proposed to tackle this problem, which can be proved to be optimal when the number of applications to be scheduled is equal to the number of core pairs. Simulation on a 64-core CMP shows 2.8%-4% improvement in weighted throughput when compared to prior *VarF&AppIPC* algorithm.**

***Keywords*-process variation; thread-level redundancy; chip multiprocessor; scheduling;**

I. Introduction

Transient faults, also called soft errors, represent a critical reliability concern for current and future Integrated Circuits (ICs). Soft errors occur when energetic particles strike and then invert the state of a device, such as a storage cell or a gate. The invert may further propagate to cause an execution error in users' programs. Advancing of manufacturing technology slightly reduces the error rate of a single transistor. However, exponential growth of the number and density of transistors on a single die, results in, once again, considerably high error rate of an entire chip [13]. As a result, it is a necessity to employ fault tolerance techniques to protect circuits and systems from soft errors.

Chip Multiprocessor(CMP) is generally regarded as the most promising architecture for future high performance computing, which has the benefits of power-efficiency and short time-to-market. As the number of on-chip cores increases, the inherent hardware redundancies provide opportunities to be explored for reliability purpose. Thread-Level Redundancy (TLR), which is widely adopted in multiprocessor systems [8], [9], prevails as one of the most efficient soft error tolerance approaches in CMPs. Operating

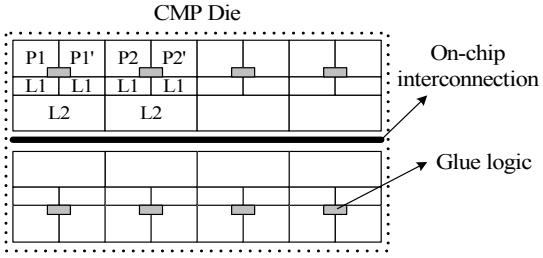

Figure 1. TLR-CMP Architecture

Systems duplicate the execution of threads on separate cores to detect and recover from soft errors. There is always a slack of instructions between the two replicated threads, through which the leading thread can forward load values and branch targets to the trailing thread, thereby improving the latter's performance. Since the comparison of execution requires frequent communication between a pair of cores, TLR typically couples two adjacent cores statically with glue logics in between, such as communication channels, buffer queues etc., as depicted in Figure 1. We call it TLR-CMP architecture in this paper, which is much similar to Paceline in [4].

Process variation [2][17][18] is another important issue that can not be ignored during system and architecture design phases, as precise control of manufacturing process becomes almost impossible. For CMPs, within die variation causes core performance to differ significantly, even though they are homogeneous in architectures. The maximum difference in core frequencies could be approximately 20%[2]. Regarding core-to-core variations in CMPs, prior research work mainly focused on the scheduling problem of applications to different cores to maximize performance or throughput [6] or to reduce power consumption [5].

In TLR-CMP architecture as shown in Figure 1, process variation will result in performance gap in between a core pair. Obviously, the leading thread should be dispatched to a higher performance core, which is called leading core, while the trailing thread to the weak trailing core. The performance

asymmetry between leading and trailing cores within a pair is named intra-pair variation, while the asymmetry among leading cores in different core pairs is named inter-pair variation in this paper. Prior research work on variation-aware scheduling in CMPs only considered the inter-pair case. However, for TLR-CMPs, the scheduling problem is quite different. Thread execution will be assigned to a core pair and the performance gap between leading and trailing cores, i.e., intra-pair variation, will also affect the behavior of the executed thread. Thus, the scheduling problem in TLR-CMP should take both inter- and intra-pair variations into account.

In this paper, we first evaluate the impact of intra-pair performance asymmetry on SPEC2000 benchmarks. We observe that some applications such as *crafty* are greatly affected by intra-pair variation, while some others like *swim* are not sensitive to this kind of variation. We adopt a weighted throughput metric to evaluate the variation-aware application scheduling in TLR-CMP, and then formalize it as a 0-1 programming problem. An efficient scheduling algorithm, called *IntraVaF&AppSen*, is then proposed to tackle this problem. We prove that when the number of threads to be scheduled is equal to the number of core pairs, *IntraVaF&AppSen* gives optimal solutions. Simulation results on a 64-core TLR-CMP (32 pairs) show that the weighted throughput is improved by 2.8%-4% when compared to VarF&AppIPC algorithm [5], which considers leading core frequencies only.

The rest of this paper is organized as follows. Section 2 reviews prior related work. In Section 3, we analyze the impact of intra-pair variation on redundant execution of SPEC 2000 benchmarks. The variation-aware scheduling problem for TLR-CMP is then formalized in Section 4. Section 4 also describes the proposed scheduling algorithm. Simulation results are shown in Section 5. Section 6 concludes the paper.

II. Related Work

A. Thread-Level Redundancy in CMPs

Traditional multiprocessor systems, i.e. IBM Z900 [9] and Compaq NonStop Himalaya [8] both employ thread-level redundancy (TLR)[3][4][14][15][16]for high reliability and availability, in which the execution of the same instruction is checked in a clock-by-clock basis, i.e., lockstep. For the first time, Mukherjee et al. [3] proposed the Chip-level Redundant Threading (CRT) for transient faults tolerance in CMP. Instead of lockstep, CRT places an execution slack between two threads. Thus the leading thread can forward load values and branch targets to the trailing thread. When a store instruction in the leading thread is to be committed, it has to wait for the same instruction in the trailing to check for correctness. Gomaa et al. [11] proposed efficient checkpoint and rollback recovery mechanisms based on CRT. LaFrieda et al. [10] presented a Dynamic Core Coupling (DCC) architecture, which allows arbitrary CMP cores to verify each other's execution and requires no static core binding and dedicated communication hardware.

System error rate due to transient faults continues to increase as technology advances into nanometer scale. All kinds of applications, not only high reliable ones require redundancy to ensure correctness. TLR-CMP, for its efficiency and flexibility, will be widely adopted, and more research efforts should be devoted in this domain.

B. Scheduling Considering Process Variation

Static process variations due to imperfect manufacturing manifest themselves as die-to-die (D2D), within-die (WID), and wafer-to-wafer (W2W) variations. For CMPs, WID variation across the chip causes individual cores to differ significantly in frequency, standby leakage current, etc., which is called core-to-core variation (C2C). The scheduling of applications in CMPs will be suboptimal if C2C variation effect is not taken into consideration.

Teodorescu and Torrellas [5] compared various variation-aware algorithms for application scheduling and power management. To maximize system throughput at a given power budget, linear programming is used in [12] to find the best voltage and frequency of each core in a CMP. Lakshminarayana et al. [7] evaluated and compared a task size aware and a critical section length aware scheduling algorithms in C2C variation environment.

For TLR-CMP architecture, a single thread will be dispatched on a pair of cores, thus C2C variation can be further divided into inter-pair and intra-pair variation. The former is the performance asymmetry among leading cores, while the latter is the variation between leading and trailing cores within a pair, as stated before. Prior work on this issue mainly focused on scheduling algorithms considering inter-pair variation. However, in TLR-CMP, the performance of a thread can also be affected by the performance gap within a pair, i.e., intra-pair variation. As a result, scheduling in TLR-CMP architecture should take both inter- and intra-pair variations into account. Before introducing our proposed algorithm, we firstly evaluate and analyze the impact of performance asymmetry within a core pair on SPEC2000 benchmark programs.

III. Intra-pair Variation Impact on TLR Execution

Figure 2 illustrates the adopted microarchitecture of TLR in this paper, which is much similar to CRT in [11]. The memory access operations, i.e., load and store instructions are all performed by leading cores. The load values of leading core from memory hierarchy are forwarded and stored in LVQ (load value queue). When trailing core needs the same data, it will fetch it from the head of LVQ. The performance gap and execution slack also enable the correct branch targets generated by leading core to be

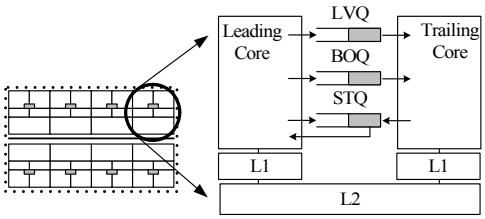

Figure 2. Thread-Level Redundancy Microarchitecture

Table I
CONFIGURATION IN A CORE PAIR

Core	4 issue, 5 retire, 176 ROB
L1 I-cache	32KB, 2way, 2cyc
L1 D-cache	32KB, 4way, 2cyc, WT
L2 cache	512KB, 8way, 9cyc, WB
LVQ/BOQ/STQ	32entries/32entries/64 entries, 2cyc

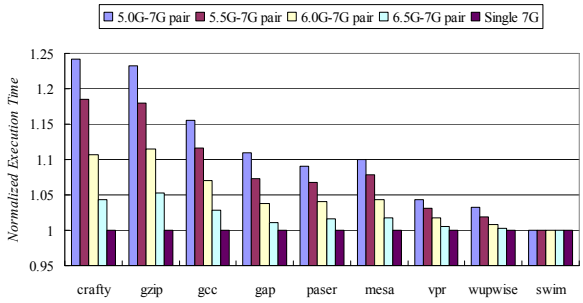

Figure 3. Normalized Execution Time of SPEC2000 Benchmarks

forwarded to trailing core via BOQ (branch outcome queue), thus accelerating the redundant thread execution. When the leading core encounters a store, it will forward the data and address to be stored in STQ (store queue) and stall to wait for trailing core. When trailing core catches up, it will compare the corresponding store instruction. If no errors occur, leading core will be triggered to continue execution. We don't consider the recovery mechanism in this paper, since we focus on the impact of performance variation on redundant execution.

A. Simulation Results and Analysis

We extend the cycle-accurate simulator SESC [1] according to the microarchitecture described above. The configuration of a core pair is listed in Table I. We keep the leading core working at 7GHz, and vary the frequency of trailing core. Simulation results on 9 SPEC2000 benchmark programs are shown in Figure 3.

As shown in Figure 3, the execution time of different intra-pair variations is normalized to that on a single 7GHz core. It is clear that different applications manifest different performance when running on core-pairs with different variations. For some applications, such as *crafty, gip, gcc*, a weak trailing core will drag the fast leading core dramatically, as shown in the figure. While for some others especially *swim*, no matter how the trailing cores are slow, the performance almost remains the same. Intra-pair variation has the least impact on such kind of programs. For *crafty, gzip, gcc*, etc, they are all computing intensive applications. Prefetching and forwarding of load values from leading core have little effect to speedup the execution in trailing core. As for *swim*, which is a memory intensive application, it can benefit greatly from prefetching, since memory access latency is the major obstacle for performance improvement.

For a fixed leading core performance, e.g., 7GHz in the above experiment, *crafty* should be scheduled to a less divergent core pair, because intra-pair variation degrades its performance greatly, while the highly divergent one should be left to *swim*, which is not sensitive to variation. If we schedule them oppositely, system throughput will be reduced. For applications that are sensitive to intra-pair variation, the faster leading core is slowed down because it always has to stall and wait for the trailing core. Stalls in leading cores waste computing resources and thus reduce system throughput.

Based on the above analysis, for the scheduling problem in TLR-CMP architecture, we should not only consider the performance asymmetry among leading cores but also the asymmetry exists within a core pair. Prior work [5] only considering inter-pair variation will result in suboptimal solutions in TLR-CMP. In the following sections, the scheduling problem for TLR-CMP is formalized, and an efficient intra-pair variation aware scheduling algorithm is proposed.

IV. Variation-Aware Application Scheduling for TLR-CMP

A. Evaluation Metric and Problem Formalization

Threads running on two redundant cores will have performance degradation when compared with single execution on the leading core. To take both inter- and intra-pair variation into consideration, we use throughput measured in millions of instructions per second (or MIPS) to evaluate such degradation. The $realMIPS_{ij}$ means the MIPS of application *i* running on core-pair *j*. Though the instruction count is doubled in TLR, we only consider the instructions executed in the leading core, since the trailing thread is functionally transparent from the view point of users.

On the other hand, for the same hardware configuration, different applications present different intrinsic MIPS. As a result, to avoid the impact of intrinsic difference of applications, we define the weighted throughput of dispatching application *i* on core-pair *j* (WT_{ij}) as the $realMIPS_{ij}$ normalized to the $refMIPS_i$ of the same application running on a 5GHz core with the same architecture:

$$WT_{ij} = realMIPS_{ij}/refMIPS_i. \quad (1)$$

Higher WT_{ij} means relatively larger instruction throughput when combining application *i* and core-pair *j*. The variation-aware scheduling problem is then to dispatch *m* threads to *n* core-pairs to maximize the system Weighted Throughput (*WT*):

$$WT = \sum WT_{ij}, \tag{2}$$

in which, $i \in \{0, 1, \cdots, m\}, j \in \{0, 1, \cdots, n\}$.

The above scheduling problem can be formalized as follows. *T* is an $m \times n$ matrix recording the weighted throughputs of all *m* threads running on all *n* core-pairs separately.

$$T = \begin{bmatrix} WT_{11} & WT_{12} & \cdots & WT_{1n} \\ WT_{21} & WT_{22} & \cdots & WT_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ WT_{m1} & WT_{m2} & \cdots & WT_{mn} \end{bmatrix}$$

S represents a certain scheduling solution, where ALO_{ij} is 0 or 1, indicating whether application *i* is allocated to core-pair *j*.

$$S = \begin{bmatrix} ALO_{11} & ALO_{12} & \cdots & ALO_{1n} \\ ALO_{21} & ALO_{22} & \cdots & ALO_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ALO_{m1} & ALO_{m2} & \cdots & ALO_{mn} \end{bmatrix}$$

The variation-aware application scheduling problem for TLR-CMP can then be formalized as:

$$Max: WT = \sum_{i=1}^{m} \sum_{j=1}^{n} WT_{ij} \times ALO_{ij};$$

$$SAT: \sum_{j=1}^{n} ALO_{ij} = 1, i \in \{0, 1, ..., m\}, ALO_{ij} \in 0, 1.$$

It is clear that the scheduling for TLR-CMP considering variation is a 0-1 programming problem, which has been proved to be NP-hard. Therefore, we don't hold much hope for finding an exact polynomial time algorithm for its solution. An efficient heuristic is then proposed in the following subsection.

B. Variation-aware Scheduling Algorithm for TLR-CMP

In this paper, we consider the scheduling in TLR-CMP when the number of threads *m* is equal to the number of core-pairs *n*. This assumption will simplify the problem as described in the following. On dealing with the above scheduling problem, we should consider three factors, i.e., intra-pair variation, inter-pair variation and application's sensitivity to variation. Before introducing our algorithm, we analyze the relationship between Weighted Throughput and the above three factors.

Running an application *i* on a core-pair *j* will have a reduced Instructions-Per-Cycle (i.e., $realIPC_{ij}$), when compared to running on the leading core of pair *j* (IPC_i). The *IPC loss* is defined as: $IPC\ loss = 1 - realIPC_{ij}/IPC_i$. For the same intra-pair variation ($IntraVar$), different applications may have different *IPC loss*. We use application's sensitivity to intra-pair variation (*AS2IV*) to indicate the intrinsic characteristics of different applications, as shown in Figure 3. $AS2IV_i$ is the average *IPC loss* for application *i* when performance gap within a core-pair is increased by one unit (e.g., 0.5GHz) for a certain architecture. As a result, the *realized IPC* of an application *i* running on a core-pair *j* can be repressed as:

$$realIPC_{ij} = IPC_{lead} \times (1 - AS2IV_i \times IntraVar_j). \tag{3}$$

It is known that MIPS can be expressed as $IPC \times F \times 10^{-6}$. Accordingly, we have:

$$refMIPS_i = IPC_{ref} \times F_{ref} \times 10^{-6},$$

and

$$realMIPS_{ij} = realIPC_{ij} \times F_{lead} \times 10^{-6},$$

where, F_{lead} is leading core frequency and F_{ref} is reference core frequency. According to Equation (1) and (3):

$$\begin{aligned} WT_{ij} &= realMIPS_{ij}/refMIPS_i \\ &= (realIPC_{ij} \times F_{lead})/(IPC_{ref} \times F_{ref}) \\ &= \frac{(1 - AS2IV_i \times IntraVar_j) \times F_{lead} \times IPC_{lead}}{IPC_{ref} \times F_{ref}}. \end{aligned}$$

Note that for the same application and the same architecture, IPC_{ref} is equal to IPC_{lead}. Then,

$$WT_{ij} = (1 - AS2IV_i \times IntraVar_j) \times F_{lead}/F_{ref}.$$

According to Equation (2),

$$WT = \frac{\sum F_{lead} - \sum IntraVar_j \times F_{lead} \times AS2IV_i}{F_{ref}}. \tag{4}$$

Equation (4) gives an important conclusion. If the number of threads is equal to the number of core pairs, that means all pairs will be used, and $\sum F_{lead}$ is the same for all scheduling. Then we need to minimize $\sum IntraVar_j \times F_{lead} \times AS2IV_i$, in which the three terms represent intra-pair variation, inter-pair variation and application's sensitivity to variation respectively.

Based on the above analysis, we introduce a simple and efficient algorithm, called *IntraVarF&AppSen* to tackle the problem. Suppose there are *n* applications to be scheduled to *n* pairs, *IntraVarF&AppSen* dispatches an application with lowest *AS2IV* to a core-pair with highest product of intra-pair variation and leading core frequency.

Suppose two applications *S* and *T* are scheduled to core-pair *i* and *j* respectively according to *IntraVarF&AppSen*.

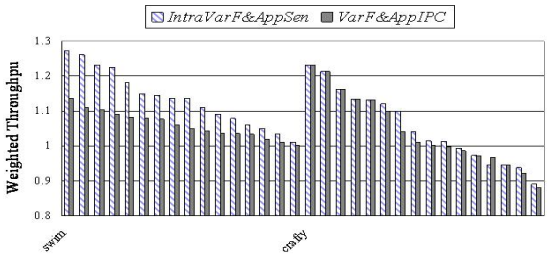

Figure 4. Weighted Throughput Comparison - an Extreme Case

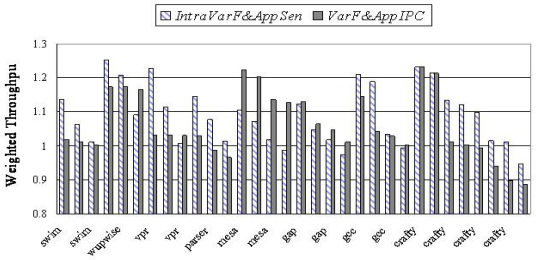

Figure 5. Weighted Throughput Comparison - an Average Case

Therefore, if $IntraVar_i \times F_{lead,i} > IntraVar_j \times F_{lead,j}$, we have $AS2IV_S < AS2IV_T$. Then,

$$(AS2IV_S - AS2IV_T) \times (IntraVar_i \times F_{lead,i} - IntraVar_j \times F_{lead,j}) < 0.$$

We can conclude after expansion that

$$\begin{aligned} & IntraVar_i \times F_{lead,i} \times AS2IV_S \\ & \quad + IntraVar_j \times F_{lead,j} \times AS2IV_T \\ < & \; IntraVar_i \times F_{lead,i} \times AS2IV_T \\ & \quad + IntraVar_j \times F_{lead,j} \times AS2IV_S. \end{aligned} \tag{5}$$

Equation (5) implies that if we changed the solution achieved by *IntraVarF&AppSen*, i.e., dispatching applications T and S to core-pair i and j respectively instead, the term $\sum IntraVar_j \times F_{lead} \times AS2IV_i$ in Equation (4) will be increased, thus reducing the system Weighted Throughput.

To sum up, the *IntraVarF&AppSen* algorithm is efficient and optimal in finding the maximum system weighted throughput, when the number of applications is equal to the number of core pairs in TLR-CMP architecture. Other cases will be considered in our future work.

V. Experimental Results

In this section, we evaluate the *IntraVarF&AppSen* scheduling algorithm on a 64-core TLR-CMP (32pairs). The 32 applications to be scheduled consist of the 9 SPEC2000 benchmark programs listed in Section 3. The core frequencies are randomly generated with 7GHz as the expectation and 20% as the variation. Teodorescu and Torrellas proposed a *VarF&AppIPC* algorithm in [5] to map threads with highest IPC on cores with highest frequency, thus only considering inter-pair variation. In the following, we compare with *VarF&AppIPC* to show the effectiveness of the proposed *IntraVarF&AppSen* algorithm.

We study two cases in this section, the extreme case and the average case. In the extreme case, half of the applications to be scheduled are *crafty* and the other half are *swim*, which are the most and the least sensitive benchmarks to variation respectively. The system Weighted Throughput achieved by *IntraVarF&AppSen* is improved by 4% compared to

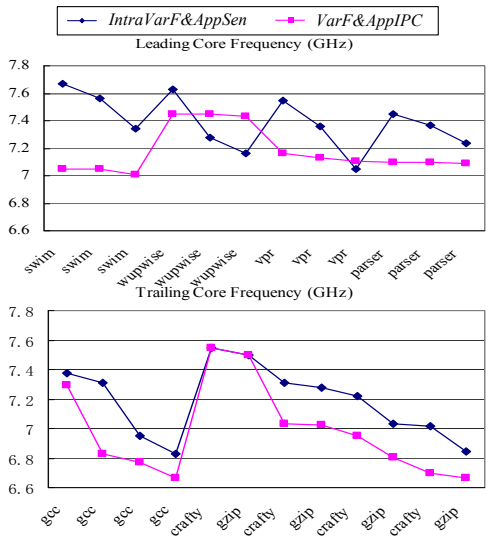

Figure 6. Comparison of Leading and Trailing Cores' Frequencies

VarF&AppIPC. Detailed throughputs of all applications are depicted in Figure 4.

It can be seen from Figure 4 that the weighted throughputs of *crafty* by using *IntraVarF&AppSen* and *VarF&AppIPC* are almost the same. However, *IntraVarF&AppSen* leaves more appropriate core pairs to *swim* and thus achieves much higher improvement.

In the second case, applications to be scheduled are average mixed. The system Weighted Throughput achieved by *IntraVarF&AppSen* is improved by 2.8% in this case. Figure 5 shows the details. Applications are rearranged according to their *AS2IV* in this figure.

We can conclude that when applications to be scheduled have distinct intrinsic sensitivity to intra-pair variation, *IntraVarF&AppSen* can achieve much higher improvement. In such circumstance, intra-pair variation has much greater impact on scheduling in TLR-CMP and should be taken into consideration carefully.

As stated in Section 3, the performance of applications that are not sensitive to intra-pair variation is approaching that running on the leading core. We demonstrate the frequencies of leading cores for 12 least sensitive applications in the experiment. In Figure 6, it is clear that, except

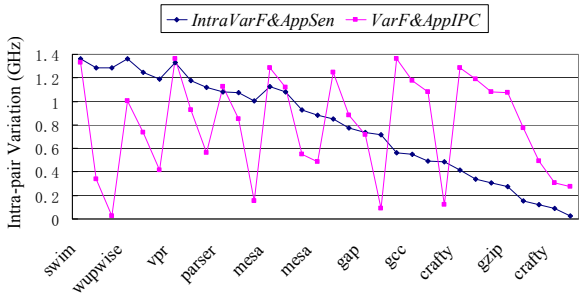

Figure 7. Intra-pair Variations Comparison

3 applications, the frequencies of leading cores running these applications are higher in *IntraVarF&AppSen* than in *VarF&AppIPC*. We also show the frequencies of trailing cores for 12 most sensitive applications as in Figure 6. All the frequencies of trailing cores are higher in *IntraVarF&AppSen* than in *VarF&AppIPC*. Since the performance of these applications is approaching that running on the trailing core, *IntraVarF&AppSen* is superior to *VarF&AppIPC*.

Finally, we show the intra-pair variations of core pairs running different applications in Figure 7. Applications are also rearranged according to their *AS2IV* in X-axis. It is clear that *IntraVarF&AppSen* dispatches the variation-sensitive applications to less divergent core pairs. *VarF&AppIPC* does not consider intra-pair variation and applications' sensitivity to such kind of variation, and will result in suboptimal scheduling solutions in TLR-CMP architecture.

VI. Conclusion

Transient faults represent a critical reliability concern in current and future technology. The inherent hardware redundancies in Chip Multiprocessors provide opportunities to be explored for reliability purpose. Thread-level redundancy (TLR) is considered to be an efficient soft error tolerant approach. Meanwhile, process variation causes core performance differ significantly across a chip. Application scheduling without considering C2C variation will result in suboptimal solutions. In this paper, we introduce two other types of variations beyond C2C in TLR-CMP architecture, i.e., inter-pair and intra-pair variation. We evaluate the impact of intra-pair performance asymmetry on SPEC2000 benchmarks. We find that some applications are greatly affected by intra-pair variation while others are not. Based on this observation, we formalize the variation-aware application scheduling problem for TLR-CMP to maximize the system weighted throughput. The scheduling problem appears to be a 0-1 programming problem. An efficient scheduling algorithm, *IntraVarF&AppSen*, is then proposed to solve this problem. The proposed algorithm can be proved to be optimal when the number of applications to be scheduled is equal to the number of core pairs. We compare the algorithm with the prior *VarF&AppIPC* algorithm. Simulation results on a 64-core TLR-CMP (32 pairs) show 2.8%-4% improvement in system weighted throughput.

Acknowledgment

The work was supported in part by National Natural Science Foundation of China (NSFC) under grant No.(60633060, 60806014, 60831160526), in part by National Basic Research Program of China (973) under grant No. 2005CB321604, and in part by Hi-Tech Research and Development Program of China (863) under grant No.(2007AA01Z109, 2007AA01Z113, 2009AA01Z126).

References

[1] The SESC simulator, *http://sourceforge.net/projects/sesc/*.

[2] E. Humenay, D. Tarjan, and K. Skadron, *Impact of parameter variations on multi-core chips*, Proc. Wkshp. on Architecture Support for Gigascale Integration, June 2006.

[3] S. S. Mukherjee, M. Kontz, S. K. Reinhardt, *Detailed Design and Evaluation of Redundant Multithreading Alternatives*, Proc. of International Symposium on Computer Architecture, 2002.

[4] B. Greskamp and J. Torrellas, *Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking*, Proc. of International Conference on Parallel Architectures and Compilation Techniques, 2007.

[5] R. Teodorescu, J. Torrellas, *Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors*, Proc. of International Symposium on Computer Architecture, 2008.

[6] P. Ndai, S. Bhunia, A. Agarwal, K. Roy, *Within-Die Variation-Aware Scheduling in Superscalar Processors for Improved Throughput*, IEEE Transaction on Computer, 2008.

[7] N. Lakshminarayana, S. Rao, H. Kim, *Asymmetry Aware Scheduling Algorithms for Asymmetric Multiprocessor*, Workshop on the Interaction between Operating Systems and Computer Architecture, 2008.

[8] Compaq Computer Corporation, *Data integrity for Compaq Non-Stop Himalaya servers*, http://nonstop.compaq.com, 1999.

[9] T. J. Slegel, et al., *IBM's S/390 G5 microprocessor design*, Proc. of Annual IEEE/ACM International Symposium on Microarchitecture, 1999.

[10] C. LaFrieda, E. Ipek, J. Martinez, R. Manohar, *Utilizing Dynamically Coupled Cores to Form a Resilient Chip Multiprocessor*, Proc. of International Conference on Dependable Systems and Networks, 2007.

[11] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, *Transient-fault recovery for chip multiprocessors*, Proc. of International Symposium on Computer Architecture, 2003.

[12] K. Srinivasan, K. S. Chatha, *Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints*, Integration VLSI, vol. 40, no.3, 2007.

[13] A. Shye, V. J. Reddi, T. Moseley, D. A. Connors, *Transient Fault Tolerance via Dynamic Process-Level Redundancy*, Proc. of Workshop on Binary Instrumentation and Applications, 2006.

[14] T. M. Austin, *DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design*, Proc. of International Symposium on Microarchitecture, 1999.

[15] K. Sundaramoorthy, Z. Purser, E. Rotenberg, *Slipstream Processors: Improving both Performance and Fault Tolerance*, Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[16] J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe, *Reunion: Complexity-Effective Multicore Redundancy*, Proc. of Annual IEEE/ACM International Symposium on Microarchitecture, 2006.

[17] S. Borkar, T. Karnik, et al., *Parameter Variations and Impact on Circuits and Microarchitecture*, Proc. of Design Automation Conference, 2003.

[18] E. Humenay, D. Tarjan, K. Skadron, *Impact of Process Variations on Multicore Performance Symmetry*, Proc. of Design, Automation, and Test in Europe, 2007.