Reliability-Aware Task Mapping on Many-Cores with Performance Heterogeneity

Kuan-Hsun Chen[1], Jian-Jia Chen[1], Florian Kriebel[2], Semeen Rehman[2], Muhammad Shafique[2] and Jörg Henkel[2]
[1]Department of Informatics, TU Dortmund (TUD), Germany
[2]Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany
Corresponding Author's Email: kuan-hsun.chen@tu-dortmund.de

I. Introduction

Due to the architectural design, process variations and aging, individual cores in many-cores systems exhibit heterogeneous performance. In terms of the architectural design, such as ARM big.LITTLE architecture [1], [3] integrates different types of cores with the same instruction sets but different frequencies in the system to accommodate the performance requirement with tolerable chip temperature or power consumption. When considering the countermeasure for soft errors on many-cores, a commonly adopted technique is Redundant Multithreading (RMT) [9] that achieves error detection and recovery through redundant thread execution on different cores for an application. However, with the performance heterogeneity, how to achieve the resource-efficient reliability becomes a non-trivial problem, since *Task mapping* and *Determining the task execution mode* (i.e. a task executes in a reliable mode with RMT or unreliable mode without RMT) both are susceptible to the resiliency of tasks and the performance of cores. A straight-forward solution could be a greedy mapping of reliability-critical task onto a high-frequency core like we adopted in *dTune* [7]. However, such a greedy approach would lack efficiency as it suffers from its local decisions because the reliability degradation for each task does not necessarily proportional to the cores' frequency degradation. In addition, it cannot provide any guarantee with respect to the satisfaction of deadline miss rate. We provide an example and demonstrate that it is not always reliability-wise beneficial to assign the reliability-critical task to the highest-frequency core.

As shown in Fig. 1, in this paper we explore how to efficiently allocate the tasks onto many-cores by using RMT to improve the overall dependability, with respect to both timing and functional correctness while also accounting for application tasks with *multiple compiled versions*. Such multiple reliable versions can be generated by using the reliability-aware compilers like [2] and [8], exhibiting diverse performance and reliability properties. By applying multiple reliable task versions and RMT, we are able to exploit the optimization space at both software and hardware-levels while exploring different area, execution time, and achieved reliability tradeoffs. The timing correctness can be defined as the deadline miss rate, which is typically adopted as the quality of service (QoS) metric in many practical real-time applications.

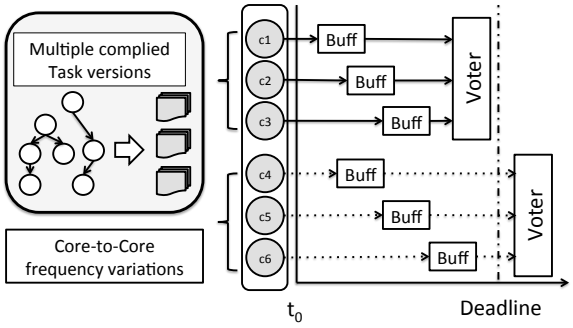

Fig. 1: Overview of interplay among performance heterogeneity, multiple task versions, and redundant multithreading. An example illustrates improper assigned cores group or version may lead to the deadline missing.

II. Problem Definition

Assume we are given a many-cores processor, which only has single thread per core, with ISA-compatible homogeneous RISC cores, and a set of tasks with multiple versions. The studied problem can be divided into two sub-problems, task mapping and execution mode adaptation. For the task mapping, assume we are given the execution modes and tolerable timing constraints, we consider *how to select the execution version and allocate the cores with different frequencies*, so that the overall reliability penalty is minimized. We observe that the problem can be connected to the well-known minimum weight perfect bipartite matching problem (MWPBM) and solved efficiently. The second sub-problem is the execution modes adaptation. The objective is to *determine the task execution modes without violating the satisfaction of deadline miss rate.* Without checking all the combinations, we propose an iterative mode adaptation to efficiently determine the execution modes of tasks with our mapping approaches so that the overall reliability penalty is minimized.

For the simplicity of presentation, the above approaches are all presented without data dependencies. After going through the approaches ideally, we discuss about how to incorporate the overhead of execution time for the data dependencies and communication with the model enhancement. Under the resource-constrained scenarios, the discussed scope of problem limits that the number of available cores is greater than or equal to the number of tasks without loss of generality.

III. METHODS

For the systems which require only a homogeneous execution mode for all the tasks (i.e. either all tasks with RMT or without RMT), we reveal that this problem can be connected to MWPBM. According to the perfect matching property, we can adopt Hungarian Algorithm to deliver a feasible mapping for the tasks and cores, where each core only be assigned to one task. When the tasks have the heterogeneous execution modes (i.e., some tasks can execute with RMT and some cannot be supported in RMT due to the resource-constraint), we propose an efficient approach to assign the tasks onto the cores to achieve the higher system reliability.

After addressing the task-mapping problem, we consider how to decide the suitable execution modes under the resource-constraints. Again, the greedy strategy is not that beneficial. Some of tasks may suffer from their higher vulnerability, whereas some may suffer from their tighter tolerance of timing constraint. As it is not possible to check all the combinations of execution modes, we propose an iterative approach exploiting our task mapping approaches as the subroutine to guarantee the feasibility and efficiency of execution modes. We also consider how to model the communication between the individual tasks and in the redundant threads. By using software pipelining, we can quantify the maximum communication overhead among all the dependent tasks under the communication fabric with XY routing on 2-Dimension mesh, in which all the redundant cores are utilized concurrently.

IV. RESULTS AND DISCUSSION

To evaluate the performance of our schemes fairly, we use the same setting in *dTune* [7] to obtain the reliability penalty of tasks by using a a real-world embedded benchmark MiBench and many-cores simulator for LEON3 ISA. The value of reliability penalties are obtained under fault rate 10^{-6} (in the unit of $\#fault/cycles$) to realize the high fault scenarios as adopted by the related works [4, 5]. We set the timing constraints as miss rate ρ. We normalize our results to the greedy mapping and compare the efficiency with the same set of tasks versions and core configurations, in which the normalized ratio is calculated as the resulting solution divided by the result of greedy mapping. By definition, the lower normalized penalty ratio is better.

The preliminary evaluation is performed by Grouping Frequency Levels with variations ω for evaluating architectures with heterogeneous performance, e.g., ARM big.LITTLE architecture [1]. We evaluated four different frequency levels in a multi-core processor. Assume the performance variation is ω, where the cores are with frequencies f_1, $(1-\omega)f_1$, $(1-2\omega)f_1$, and $(1-3\omega)f_1$, in which ω is up to 30% [3] due to the real-word scenarios on performance variations. As shown in Fig.2, we can observe that our approach outperforms the greedy mapping significantly. If the task mapping is not decided properly, the total reliability penalties will be increased dramatically. Since the difference of frequencies between different grouping levels is large enough, the greedy mapping may suffer from the sequential assignment of cores, in which the RMT tasks

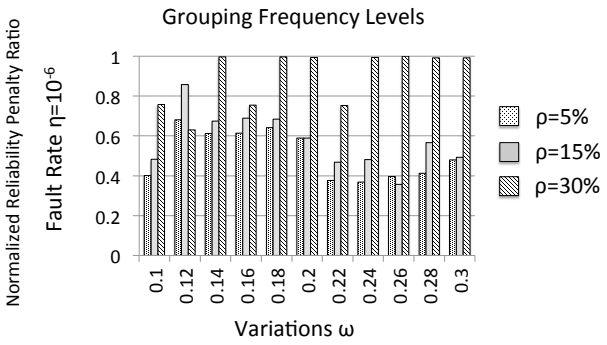

Fig. 2: Comparing the reliability penalty ratio by normalizing our result to the greedy mapping under 10^-6 fault rates.

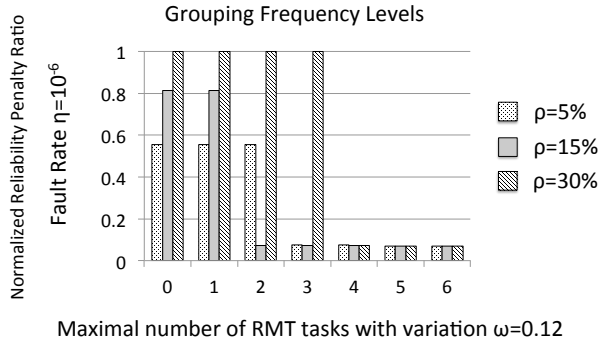

Fig. 3: Evaluation of the execution modes adaptation with variation $\omega = 0.14$ under 10^-6 fault rates.

may have serve performance degradation. For the execution mode adaptation, we can see that the trends in Fig. 3 with the delivered execution modes still follow the previous observation in the task mapping. If the frequencies variation among the cores is not negligible as the case of grouping frequency levels, the effectiveness of proposed mapping approach is illustrious.

After all, we can conclude our proposed approaches provide a better overall reliability in terms of greedy strategy without violating both software and hardware-levels constraints.

REFERENCES

[1] ARM. big.little technology: The future of mobile, 2013.

[2] A. Benso, S. Chiusano, P. Prinetto, and L. Tagliaferri. A c/c++ source-to-source compiler for dependable applications. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 71 –78, 2000.

[3] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Solid-State Circuits, IEEE Journal of*, 37(2):183–190, 2002.

[4] J. Hu, S. Wang, and S. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 281 –290, june 2006.

[5] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. Irwin. Soft error and energy consumption interactions: A data cache perspective. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on*, pages 132–137, Aug 2004.

[6] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel. dtune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects. In *DAC*, pages 1–6, 2014.

[7] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Reliable software for unreliable hardware: embedded code generation aiming at reliability. In *CODES+ISSS*, pages pp. 237–246, 2011.

[8] J. Smolens, B. Gold, B. Falsafi, and J. Hoe. Reunion: Complexity-effective multicore redundancy. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 223–234, 2006.