

Process Variation Aware Parallelization Strategies for MPSoCs

Abstract—Scaling of microprocessors is aggravating the gap between design and manufacturing expectations. Such variations may lead to manufacturing of processors cores with frequencies lower or higher than their expected frequencies. In particular, with the rapid advent of Multiprocessor System on Chips (MPSoC), such manufacturing uncertainties may lead to significant variations in the operating frequencies of different processors cores on the same chip. In this work, we demonstrate that traditional load balanced parallelization schemes need to be revisited to account for such variations. Specifically, we show the need for tuning the degree of parallelization and non-uniform workload generation to achieve lower power consumption in the presence of process variations.

I. INTRODUCTION

Variations in operating frequencies due to unmanageable fabrication needs have been the biggest concern of present day computer architects. Such variations directly impact the manufacturing yield and thereby the revenues of the industry [1]. The significant impact of process variations on the maximum operating frequencies and the power consumption of the architecture have been studied elaborately [2]. With growing demand for MPSoCs and the increasing number of cores on a single die, the probability of such parametric variations affecting the system operating frequencies of the different cores may vary quite significantly. Such systems with parametric variations are quite similar to heterogeneous systems, with each cores having similar architectural specifications. There have been many optimization schemes and tools to counter such variations using various circuit and architectural techniques [3]. However, such variations require solutions at different levels, starting from application design and mapping to architectural tuning and circuit optimizations. Particularly in case of MPSoCs, where the parallelization and synchronization of the processors decide the performance of the system, such algorithmic modulations are necessary in the applications.

The load balancing approaches for obtaining the best performance [4] in homogeneous systems may no more be sufficient to achieve the best results out of a MPSoC system with variable frequencies of the cores. Keeping this motivation in mind, we first observed the performance of such applications under variations, and analyzed the changes in the required parallelization schemes. We observed that across different benchmarks the average Energy Delay Product (EDP) degrades by a factor of 54%.

Another important observation that motivated our optimizations in such systems is the fact that the optimality on the number of processors with respect to the EDP value shifts under variations to a different domain in most of the benchmarks. This motivated us to frame an analytical model to determine the best set of processors to implement the application on. We propose a table lookup based analytical

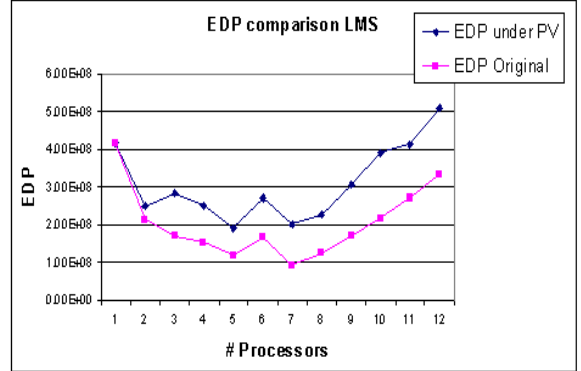


Fig. 1. Increase in EDP due to process variations

model which estimates the EDP of the system and find the best configuration for the given application implementation.

To tune the application to the system we also demonstrate a load balancing strategy based upon the frequencies of different cores. Different applications from the SPLASH suite [5] were tested on a cycle accurate virtual platform in systemC [6] for our experimentation. We demonstrate the modified parallelization, optimal processors selection scheme and a voltage scaling methodology to counter the problems arising due to variations.

II. IMPACT OF VARIATIONS

We demonstrate the impact of variability on the speedup and the energy consumed by the system with cores operating on different frequencies for numerous benchmarks. The impact is observed keeping the existing algorithmic implementation on the system, with the optimal load balancing under uniform frequency assumption. Since we are observing a system with variable frequency, we are using the total energy consumption and the delay incurred as a metric to evaluate the system. Therefore, throughout this work we use EDP as the metric for evaluating the quality of different systems and configurations proposed by us.

To demonstrate the impact of processors operating at variable frequencies, we first implemented different applications on such heterogeneous systems and obtained their EDPs. Figure 1 demonstrates the impact of EDP when one of the benchmarks, LU from the SPLASH suite is implemented on system with different cores having variable frequencies. The cores were assigned a frequency of current frequency ratings of ARM processors(200MHz, 100MHz and 66MHz). These three frequencies are assigned keeping the mean as 100MHz and assuming a Gaussian distribution.

Each of the points in the graph depict the original benchmark execution under such distribution of frequencies in different cores. It is quite evident that the overall EDP of the application execution increases on such variability affected

systems. In case of this benchmark the average increase in the EDP of the benchmark executed under variation-free and random frequencies is close to 35%. Another important observation in such a graph is the optimal number of processors required by the system for execution.

It is also seen that in a variation-free system, 7 processors provides the optimal EDP for the system, however, in case of system under variation 5 processor configuration has the minimal EDP. This provides the prime motivation behind our analysis for determining the optimal number of processors for the given application under variations.

III. OPTIMIZATION STRATEGIES

The disparity in processing frequencies induced by the process variations (PV) present a challenging problem to the compiler developers. They now need to take this additional performance parameter into consideration, while allocating load across processors. This section will present a novel solution to this problem, by employing exhaustive analytical modeling, along with supporting experimental analysis to identify task allocation schemes that can adapt to any available processor configuration.

A. Task Allocation schemes

To maintain consistency of results and ease of evaluation, we assume a processor set of 8 ARM cores, and a job of allocating the workload associated with the benchmark (problem size N) across the same. Load redistribution in different benchmarks was obtained keeping in mind the implementation styles of the benchmarks, without increasing the execution time of the system (decided by the processors operating at their lowest frequency). The task allocation schemes investigated by us are briefly described below.

1) *Frequency Scaling Scheme (FSS)*: To maintain synchronization and realize real-time responses, conventional task allocation schemes allocate tasks based on the assumption that all processors can operate uniformly at the lowest possible frequency (in case of our platform, 66MHz). This scheme goes by the name FSS.

2) *Analytical Lookup Based Optimization Scheme (ALBOS)*: ALBOS employs a exhaustive table lookup method to identify the best combination of cores that would generate minimal EDP. Input to the lookup method includes (i) Frequencies of the cores under evaluation (ii) Execution cycles and average power consumption of the benchmark, with cores operating at uniform frequencies. The algorithm behind ALBOS is simple and is as presented below.

Algorithm:

- 1) Generate all the possible permutations and combinations of the available P cores, operating at frequencies $(f_1, f_2 \dots f_P)$ as provided in the input
- 2) Estimate the EDP for each of the above configurations - Table Lookup
- 3) Identify the configuration with least EDP as the optimal one
- 4) Split the task uniformly (N/P) across the optimal configuration, and begin execution

The table lookup comprises of simple estimation of the execution time of different processors and computation of the energies. Note that computing the execution time just needs information about the cycles of execution of any benchmark on a given number of processors which is independent of the frequencies and hence is an accurate computation. The execution time of the benchmark however is dependent upon the processor with least frequency. The energy estimation again is accurate since the energy is picked up based on the parallelization and the frequency of the cores. We however add the energy associated with the idling time of the faster processors in presence of variations. This energy is considered to be just the leakage component and is approximated at 10% of the original leakage power consumption per cycle.

Note that an optimal processor configuration does not always need to involve 8 processors, it might turn out that a configuration with less than 8 processors depicts a much impressive EDP than the former. This could be justified looking at the parallelization chart of LMS benchmark depicted in figure 2. Assuming a variation given 8 processors with 4 processors operating at 100MHz, 2 at 200MHz and the rest 2 at 66MHz, the algorithm finds 5 processors as optimal solution. Since by picking 7 processors the system has at least one processor with frequency of 66MHz, which reduces the EDP of such a configuration lower than a 5 processor configuration having the fastest 5 processors operational. It also might be the case that the optimal configuration may not go for the fastest amongst all the processors since, the parallelization needs of the benchmarks may really not need the fastest processors. Consider the LU parallelization chart in figure 3. The EDP of configuration with all cores at the lowest frequencies are quite comparable or even lower than configurations with all fast cores. This encourages the algorithm to pick up the slower cores and the resultant configuration as demonstrated in table I in section IV, picks up 6 processors involving the slower processors.

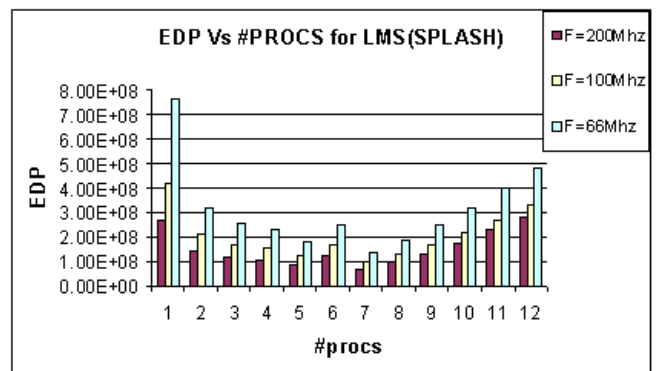


Fig. 2. EDP of LMS for different number of processors at same frequencies

3) *eXtended Analytical Lookup Based Optimization Scheme (xALBOS)*: This scheme is on the same lines as the previous one, except that instead of blindly distributing the tasks uniformly across processors, we do the same relative to the cores frequency of operation as shown in figure 4. Distributed this way, we can minimize the idling time of faster processors, as they now need to process an additional workload of $(N/P)\{f/\sum_{i=0}^n f_i\}$.

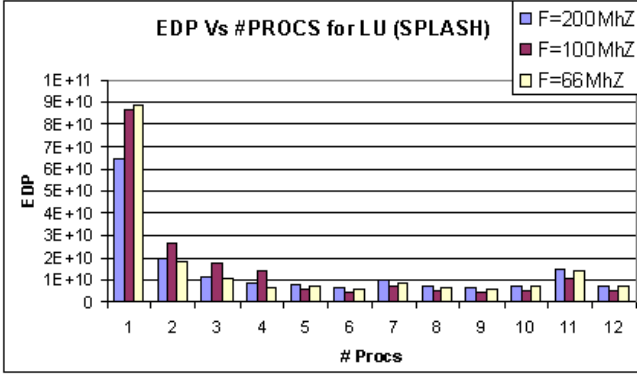


Fig. 3. EDP of LU for different number of processors at same frequencies

The effectiveness of our analytical processor picking schemes and the load balancing strategy is then tested by creating the required tables and exhaustively searching the optimal processors for application implementation. This is followed by load balancing of the benchmarks over the optimal processor configuration.

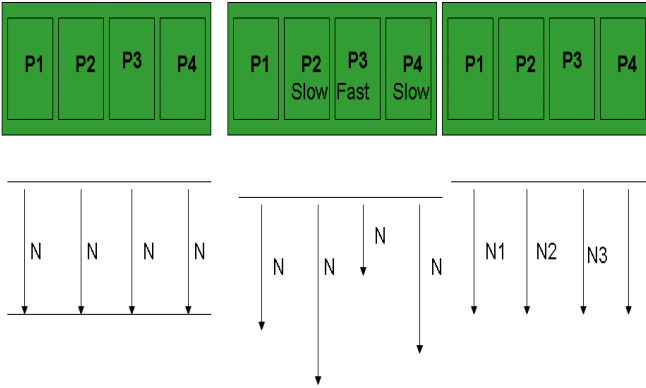


Fig. 4. Need for load balancing under variations

B. Load distribution methodology

This subsection briefly describes the strategy for creating an unbalanced load distribution of each of the benchmarks. The primary allocation scheme is a weighted allocation where the load is allocated to the processors based upon the frequency of operation. The operations in 1D-FFT benchmark were in the form of blocks of identical computation communicating at the end of their tasks. The communication is in the form of a well connected network also termed as butterfly connection [7]. These blocks are distributed to different processors evenly for balanced operation and maximum efficiency. We distribute such blocks based on the frequencies of the processors to obtain load imbalances based on the weighted allocation scheme proposed by our xALBOS strategy. As demonstrated in figure 5 the root N (square root of N, where N is the number of samples for the FFT algorithm), blocks are redistributed unevenly based on the weighted distribution scheme discussed in previous section. In LU benchmark the parallelization is in the form of columns of matrices allocated to different processors, which, once again, can be distributed unevenly [7]. Similar unbalanced distribution of arrays and matrices are used for

Quick Sort and the PIL Filter benchmarks. However, in case of the LMS benchmark the operations of computing and applying the filter are performed by different processors and they operate in a producer consumer manner. This prevents creation of load imbalances due to the such synchronization constraints. The benchmark however, if written in a different style of parallelization where each processors has independent operations to some extent, may use the uneven distribution scheme.

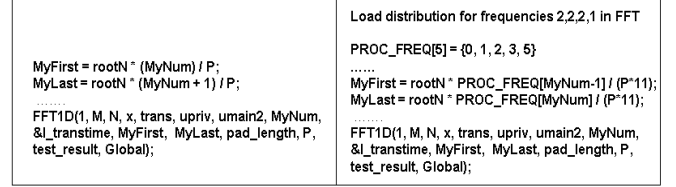


Fig. 5. Code to redistribute the load unevenly

IV. EXPERIMENTAL ANALYSIS

The experimental platform consists of MPARM [6], a cycle-accurate multi-processor architectural simulation framework, developed to support design and architectural exploration on the MPSoC platforms. For the purpose of this study, we use the ARM7TDMI core, over AMBA interconnects, operating in the RTEMS environment. The cores can be tweaked to operate at three different frequencies of 200, 100 and 66MHz. Interested readers are referred to [6] for a more comprehensive handling of the simulator. For benchmarking the experimental platform, we used three benchmarks from the SPLASH suite [5] namely, LU decomposition, FFT, LMS, and two manually parallelized benchmarks, PIL filter and quick sort.

A. Results

The benchmarks were executed on a system of 8 cores with, 4 cores at 100MHz, 2 at 200MHz and the rest 2 at 66MHz. We tested the different schemes proposed by us and compared the EDP with the Original EDP (OEDP) of the system running on 8 processors each at 200MHz. Figure 6 demonstrates the EDP under variations (PV), and the EDP of different optimization schemes proposed by us, normalized to the OEDP. Table I, presents the processor configuration picked by our ALBOS scheme, where 1 represents

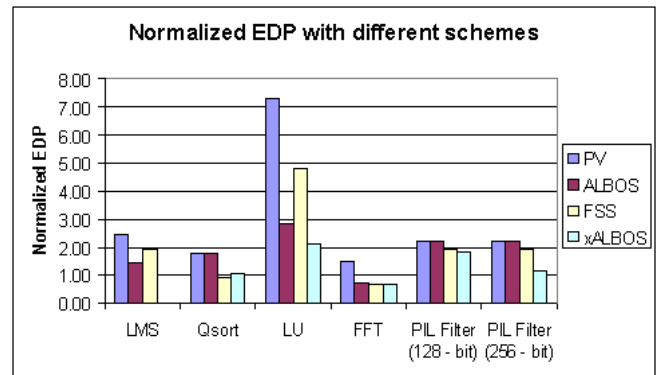


Fig. 6. EDP of different benchmarks with different schemes normalized to original EDP on 8 processors with no variations.

| Benchmark | #Procs | Configuration |
|-----------|--------|-----------------|
| Qsort | 8 | 2,2,2,2,3,3,1,1 |
| PILFILTER | 8 | 2,2,2,2,3,3,1,1 |
| FFT | 4 | 2,2,2,2 |
| LMS | 5 | 2,2,2,1,1 |
| LU | 6 | 3,2,2,2,2,3 |

TABLE I

CONFIGURATIONS PICKED UP BY THE ALBOS SCHEME FOR DIFFERENT BENCHMARKS

200MHz, 2 represents 100MHz and 3 represents 66MHz. As demonstrated in figure 6 the ALBOS scheme always shows significant improvement except for the case of Quick Sort.

Such an observation reflects the fact the ALBOS scheme is heavily dependent on the parallization of the algorithm. Quick sort algorithm gains much significantly on the number of cycles with increase in number of processors as compared to other applications. In other cases ALBOS picks up number of processors less than 8 and is able to take advantage due to execution time benefits by picking up faster processors. In Quick sort however ALBOS obtained the best combination as 8 processors and the only benefit obtained is due to the permutation amongst the 8 processors. The load balanced scheme however provides advantage in all cases due to balanced workloads and shows an average improvement of close to 23% over ALBOS and close to 54% over variation affected EDP (PV). Note that this improvement is significant primarily due to the comparison with a naive original algorithm on 8 processors. Frequency Scaling Scheme (FSS) in such processors itself reduces close to 46% of the increased EDP under EDP. However, the gains of frequency scaling are completely dependent on the applications' original EDP with at lower frequency processors. The xALBOS performs almost same or better, as FSS in almost all the benchmarks, on the other hand FSS may perform really bad as compared to xALBOS in benchmarks like LU, which are significantly impacted due to frequency of processors.

B. Fault tolerance

The approach adopted in the proposed scheme provides a lot of scope for realizing Fault Tolerance (FT). It is evident from the results presented in the previous section that in case of 3 of the above discussed benchmarks, a combination of less than 8 processors provide an optimal EDP. This in turn indicates that the remaining idle processors can potentially be employed to introduce redundancy in the target platform.

We performed an initial analytical analysis to determine the practicality of the concept and to analyze the overhead incurred. We assumed that the benchmark checkpoints after successive completions of 20% of overall runtime. Lets say a core fails after the benchmark ran for 21% of its effective run time. At the 40% checkpoint the failed core (C_f) gets detected, and the task is migrated to the other redundant/idle core (C_i), where the task continues execution. The overheads incurred under the above conditions for three of the discussed benchmarks is as shown in figure 7.

The overhead incurred is calculated from the simple formula of "Cycles executed on C_f + cycles to be executed C_i - idle cycles of C_i after C_f fails". Note that the overhead of checkpointing and saving state has not been considered in overhead calculations. This is because the benchmarks

operate on input vectors from the shared memory and store the results again in the shared memory.

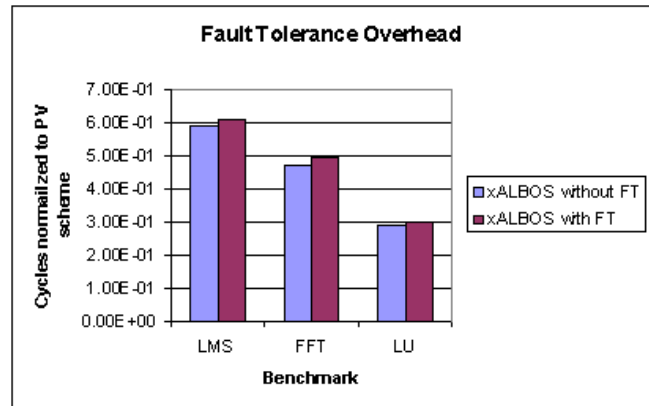


Fig. 7. Overhead incurred due to FT, normalized to PV scheme.

We observe a very meagre increase in the cycle count compared to the unmodified xALBOS scheme, indicating that the scheme is flexible enough to incorporate FT. Even after the additional redundant cycles the FT based scheme has lesser cycle count than the PV scheme.

V. CONCLUSION

The aggravation of parametric variations can lead to non-uniform processor frequencies in MPSoCs and therefore calls for judicious schemes of parallelization in such systems. In this paper we have demonstrated the significant impact of process variations on MPSoCs in the form of increase in the EDP of the applications if mapped without considering variations. We present novel solutions to tackle and subsidize the impact of the such drastic EDP increase using load redistribution and a scheme to pick up the right configuration of processors. We compare the different schemes with a simple frequency scaling scheme where the frequencies of the processors are scaled to the minimum frequencies. Future work in this direction may look into developing compiler techniques to perform dynamic mapping of applications based on run-time traces.

REFERENCES

- [1] S. Nassif "Modeling and analysis of manufacturing variations" *Proceedings of IEEE Custom Integrated Circuits Conference*, 2001.
- [2] S. Borkar, T. Karnik and V. De "Design and reliability challenges in nanometer technologies" *Proceedings of Design Automation Conference (DAC)*, 2004.
- [3] D. Blaaw and K. Chopra "CAD Tools for Variation Tolerance" *Proceedings of Design Automation Conference (DAC)*, 2005.
- [4] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker. "Solving Problems on Concurrent Processors" Prentice-Hall, Englewood Cliffs, 1988.
- [5] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta "The SPLASH-2 programs: Characterization and methodological considerations" *In proceedings 22nd annual International Symposium on Computer Architecture*, ISCA-1995.
- [6] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini and R. Zaffalon. "Analyzing On-Chip Communication in a MPSoC". *In proceedings of Design Automation and Test in Europe*, DATE-2004.
- [7] D. Culler, J. P. Singh, and A. Gupta. "Parallel Computer Architecture: A Hardware/Software Approach" *Morgan Kaufmann Publishers*, 1998