

Fault-Tolerant Average Execution Time Optimization for General-Purpose Multi-Processor System-On-Chips

Mikael Väyrynen[†], Virendra Singh[‡] and Erik Larsson[†]

Department of Computer Science[†]
Linköping University
Sweden

Supercomputer Education and Research Centre[‡]
Indian Institute of Science
India

ABSTRACT¹

Fault-tolerance is due to the semiconductor technology development important, not only for safety-critical systems but also for general-purpose (non-safety critical) systems. However, instead of guaranteeing that deadlines always are met, it is for general-purpose systems important to minimize the average execution time (AET) while ensuring fault-tolerance. For a given job and a soft (transient) error probability, we define mathematical formulas for AET that includes bus communication overhead for both voting (active replication) and rollback-recovery with checkpointing (RRC). And, for a given multi-processor system-on-chip (MPSoC), we define integer linear programming (ILP) models that minimize AET including bus communication overhead when: (1) selecting the number of checkpoints when using RRC, (2) finding the number of processors and job-to-processor assignment when using voting, and (3) defining fault-tolerance scheme (voting or RRC) per job and defining its usage for each job. Experiments demonstrate significant savings in AET.

I. INTRODUCTION

The semiconductor technology development makes it possible to fabricate integrated circuits (ICs) with billions of transistor. In order to enhance performance (the overall throughput of jobs), it is increasingly common to use the high transistor count to design ICs with multiple processors, so called multi-processor system-on-chips (MPSoCs).

Until recently, production test was sufficient to distinguish faulty ICs from fault-free ICs. However, for ICs fabricated in recent semiconductor technologies, production test *only* is no longer sufficient. Fault-tolerance, *detection* and *handling* of errors that occur during operation, is therefore increasingly important to consider [1]; not only for safety-critical systems but also for general purpose (non safety-critical) systems.

Fault-tolerance is an active area [2] [3], and has been such for a long time [4]. However, most work has focused on safety-critical systems. For example, the architecture for the fighter JAS 39 Gripen contains seven hardware replicas [5]. Much research has been proposed to optimize fault-tolerance for safety-critical systems [6] [7] [8]. Al-Omari

et al. proposed a technique to handle a single error in multiprocessor systems while ensuring timing constraints, and Izosimov *et al.* proposed for hard real-time safety-critical applications the optimization of re-execution and roll-back recovery under the assumption that errors are detected and that the maximal number of faults are known [8]. Ejlali *et al.* explore the resource conflict between time-redundancy and dynamic voltage scaling [9] and Cai *et al.* study fault-tolerant cache design [10].

For a general purpose system, for example a mobile phone, redundancy such as the one used in JAS 39 Gripen, seven hardware replicas, is unimaginable. Further, for general purpose systems, the average execution time (AET) is more important than meeting hard deadlines. For example, a mobile phone user can usually accept a slight and temporary performance degradation in order to be ensured fault-free operation.

In order to design a fault-tolerant general purpose MP-SoC there is a need of a mathematical framework for analysis, evaluation and exploration of fault-tolerance versus AET. In this paper we define a mathematical framework where we assume given is a number of *jobs* to be executed on an MPSoC and a *soft (transient) error probability*.

The main contributions of this paper are:

- given a job and an error probability, we define a mathematical formula that makes it possible to find the optimal number of *checkpoints* for the job in respect to AET when employing *rollback recovery with checkpointing (RRC)*. The AET formulation includes bus communication overhead.
- given a job and an error probability, we define a mathematical formula that makes it possible to analyze the AET for the job using various number of *processors* (replicas). The AET formulation includes bus communication overhead.
- given a set of jobs, an MPSoC with a number of processors, and an error probability, we make an integer linear programming (ILP) formulation to find the optimal fault-tolerance scheme and AET for the system. We define the problems P_v , P_{rrc} , and P_{v+rrc} where P_v assumes *voting* for each job, P_{rrc} uses *RRC* for each job, and P_{v+rrc} selects *voting* or *RRC* for each job. The objective for P_v , P_{rrc} , and P_{v+rrc} is to optimize respective fault-tolerance scheme and bus communication such that the system's overall AET is minimal.

¹The research is supported by The Swedish Foundation for International Cooperation in Research and Higher Education (STINT) through Institutional Grant for Younger Researchers for collaboration with Indian Institute of Science (IISc), Bangalore, India.

The rest of the paper is organized as follows. The preliminaries for the MPSoC, jobs, error probability, error detection and error handling are described in Section II. In Section III, the job-level optimization is presented. Given is a job, an error probability, and a number of processors, and we define formulas for AET for RRC and voting. The system-level optimization is in Section IV. We assume given is an MPSoC, a set of jobs, and an error probability, and the objective is to assign jobs to the processors such that AET is minimal while ensuring fault-tolerance. The paper is concluded in Section V.

II. PRELIMINARIES

For the MPSoC, we assume that n processors are connected by a bus as shown in Figure 1. Each processor node has its own private memory and there is a common shared memory. Bus communication, processor to shared memory, involves an overhead denoted by τ_b , which we assume to be equal to $5\mu s$.

We assume that given is P_τ , which denotes the probability of error-free execution for τ time units on any of the processors in the MPSoC. The probability that a job with longer execution time is impacted by an error is higher than that of a shorter job. For each job, we assume given is the error-free execution time T . And, for a given job with error-free execution T , the probability of fault-free execution P is given by:

$$P = P_\tau^{\frac{T}{\tau}}, 0 < P_\tau < 1 \quad (1)$$

In the paper, we make use of the fault-tolerant schemes rollback-recovery with checkpointing (RRC) and voting (both detailed in Section III). Both schemes require test evaluation to determine if errors have occurred. For test evaluation, we assume a common *test evaluator* for the MPSoC, which is connected to the shared memory (as shown in Figure 1). The test evaluator can compare the contexts of two or more processors. A context is a snapshot of a processor's registers. Such snapshots are used by operating systems when switching jobs, that is saving the state of one job and restoring another preempted job. We make use of the contexts to detect if errors have occurred. For example, if a given job is executed on two processors, the contexts of the processors should after the execution in the error-free case not differ. However, if the contexts differ, at least one error has occurred.

We use the following two assumptions regarding errors:

- An error may occur at any time during the execution of a job on a processor. Hence, we do not consider errors that occur during bus communication and test evaluation. For such errors, we assume that error-tolerant techniques can be applied.
- The fault effect (captured by the context of a processor) from any error is different. We assume that it is so *extremely* unlikely that the fault effect (the context of a processor) of errors will be *exactly* the same and will not differ in a single bit that such case can be neglected. Hence, if a given job is executed on

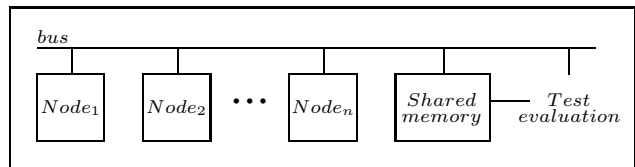


Figure 1: MPSoC architecture with n processor nodes, a shared memory and test evaluation

two processors, and if one error or more errors occur in the two processors, we assume that the processor's contexts, which contain the fault effect, are different.

The overhead involved for comparing contexts is denoted by τ_c , which we in this paper assume to be equal to $10\mu s$.

III. JOB-LEVEL OPTIMIZATION

In this section, we define formulas useful for the AET analysis of a job's requirements on fault-tolerance when using *rollback recovery with checkpointing (RRC)* and *voting (active replication)*.

A. Rollback-Recovery with Checkpointing (RRC)

In RRC, a given job with fault-free execution time T is divided into n_c execution segments (ESs). The error-free execution time of an ES is therefore $t = T/n_c$. In conjunction to each ES, a checkpoint is inserted, hence n_c checkpoints are added. The job is simultaneously executed on two processors. When an ES has been executed and a checkpoint is reached, the status of the two processors are compared against each other. As discussed above, we assume that test evaluation (Figure 1) handles the error detection by comparing the contexts of the two processors.

Checkpointing imposes an overhead that comes from the bus communication for transferring the context of a processor to the shared memory (τ_b), the time to compare contexts (error detection) (τ_c), and the checkpoint overhead (τ_{oh}), which is to load and start a new ES in case of no error or to load and restart at the previous checkpoint if one or more errors have been detected.

Figure 2 exemplifies the execution of ESs and checkpoints. Execution segment ES_1 is error-free and after checkpoint overhead, execution segment ES_2 is executed. An error occurs during ES_2 , and the error is detected by the test evaluation during the following checkpoint. Due to the error, ES_2 has to be re-executed, which means the processors should start the execution from the previous checkpoint, which is known to be error-free. Restarting at a checkpoint means that the context of the processors have to be loaded with context saved at the checkpoint.

Given a job with error-free execution time T , the probability of no error P is given from Eq. (1). The error-free execution time of any of the n_c execution segments is $t = T/n_c$ and the probability of error-free execution of an ES executed on *one* processor is given by:

$$P = p^{n_c} \Leftrightarrow p = \sqrt[n_c]{P}, 0 < P < 1 \quad (2)$$

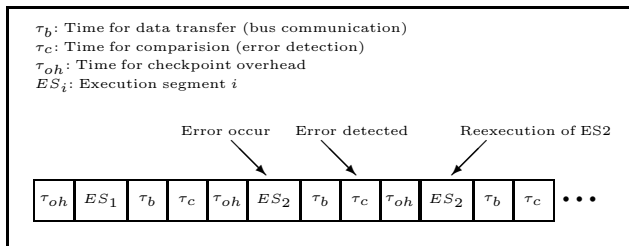


Figure 2: Error occurrence, detection and handling

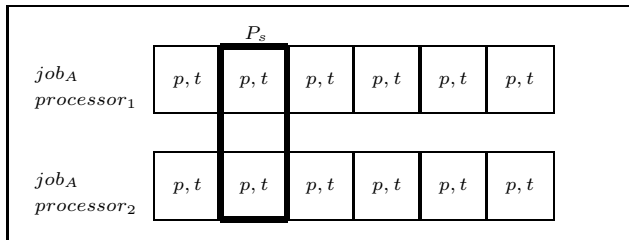


Figure 3: Rollback recovery with checkpointing

The probability of error-free execution of an ES implies that both processors are error-free, hence $P_s = p^2$ (P_s is illustrated in Figure 3). Given P_s , the error probability is given as: $Q_s = 1 - P_s$.

A high number of checkpoints gives less time in each execution segment (t), and in case of an error there is therefore a lower re-execution penalty as less instructions are to be re-executed. However, a high number of checkpoints increases the bus communication overhead. In order to be able to analyze the AET for RRC, we need expressions for the AET ($E[T_{exe,rr}]$) of a job executed with n_c checkpoints and total AET ($E[T_{tot,rr}]$) when also bus communication overhead is included.

The AET ($E[ES]$) for one execution segment is computed by Eq. (3) and Eq. (4).

$$\begin{aligned}
 E[N] &= \sum_{k=1}^{\infty} k \times (1 - P_s)^{k-1} \times P_s = \\
 P_s \times \frac{d}{dQ_s} \sum_{k=1}^{\infty} Q_s^k &= P_s \times \frac{d}{dQ_s} \frac{Q_s}{1 - Q_s} = \\
 P_s \times \frac{1}{1 - Q_s} + P_s \times \frac{Q_s}{(1 - Q_s)^2} &= \\
 1 + \frac{Q_s}{1 - Q_s} &= \frac{1}{1 - Q_s} = \frac{1}{P_s}
 \end{aligned} \tag{3}$$

where k is an iteration variable as an error may occur at any time.

$$E[ES] = t \times E[N] = \frac{t}{P_s} \tag{4}$$

Given AET for one ES (Eq.(4)), the AET ($E[T_{exe,rr}]$) for a job at n_c checkpoints is:

$$E[T_{exe,rr}] = n_c \times E[ES] = \frac{n_c t}{P_s} = \frac{n_c t}{p^2} = \frac{T}{n_c \sqrt{P^2}} \tag{5}$$

Checkpoints impose communication overhead. At each of the n_c checkpoints, each of the two processors has overhead corresponding to $\tau_b + \tau_c + \tau_{oh}$ (as discussed above). Given Eq. (5), the communication overhead $E[T_{com,rr}]$ is given as:

$$E[T_{com,rr}] = n_c \times \frac{2 \times \tau_b + \tau_c + \tau_{oh}}{n_c \sqrt{P^2}} \tag{6}$$

by exchanging T to $2 \times \tau_b + \tau_c + \tau_{oh}$ (overhead) and multiply by n_c (checkpoints).

The total AET including communication overhead ($E[T_{tot,rr}]$) is given by combining Eq. (5) and Eq. (6) as:

$$\begin{aligned}
 E[T_{tot,rr}] &= \\
 E[T_{exe,rr}] + E[T_{com,rr}] &= \\
 \frac{T}{n_c \sqrt{P^2}} + (2 \times \tau_b + \tau_c + \tau_{oh}) \times \frac{n_c}{n_c \sqrt{P^2}} &=
 \end{aligned} \tag{7}$$

As Eq. (7) is in explicit form, the optimal number of checkpoints is given by finding the derivative. The derivative of Eq. (7) is given as:

$$\begin{aligned}
 \frac{dT_{tot,rr}}{dn_c} &= \frac{d}{dn_c} \left(\frac{T}{n_c \sqrt{P^2}} + \overbrace{(2 \times \tau_b + \tau_c + \tau_{oh}) \times \frac{n_c}{n_c \sqrt{P^2}}}^C \right) = \\
 &= C \times P^{-\frac{2}{n_c}} + (T + C \times n_c) \times \frac{d}{dn_c} \left(P^{-\frac{2}{n_c}} \right) = \\
 &= \left\{ x = \frac{-2}{n_c} \Rightarrow \frac{dx}{dn_c} = \frac{2}{n_c^2} \Leftrightarrow dn_c = \frac{n_c^2}{2} dx \right\} = \\
 &= C \times P^{-\frac{2}{n_c}} + (T + C \times n_c) \times \frac{2}{n_c^2} \times \frac{d}{dx} (P^x) = \\
 &= C \times P^{-\frac{2}{n_c}} + (T + C \times n_c) \times \frac{2}{n_c^2} \times \ln P \times P^x = \\
 &= C \times P^{-\frac{2}{n_c}} + (T + C \times n_c) \times \frac{2}{n_c^2} \times \ln P \times P^{-\frac{2}{n_c}} = \\
 &= \frac{P^{-\frac{2}{n_c}} \times (C + (T + C \times n_c) \times \frac{2}{n_c^2} \times \ln P)}{1}
 \end{aligned} \tag{8}$$

In order to find the minimum of Eq. (7), we set the derivative (Eq. (8)) equal to zero and we get:

$$\begin{aligned}
0 &= P^{\frac{-2}{n_c}} \times (C + (T + C \times n_c) \times \frac{2}{n_c} \times \ln P) = \\
&C + (T + C \times n_c) \times \frac{2}{n_c} \times \ln P = \\
&\frac{C}{\ln P} + (T + C \times n_c) \times \frac{2}{n_c} = \\
&\frac{C \times n_c^2}{\ln P} + 2 \times T + 2 \times C \times n_c = \\
&n_c^2 + 2 \times \ln P \times n_c + \frac{2 \times T \times \ln P}{C} = \\
(n_c + \ln P)^2 - (\ln P)^2 + \frac{2 \times T \times \ln P}{C} &\Leftrightarrow \\
\underline{n_c} = -\ln P + \sqrt{(\ln P)^2 - \frac{2 \times T \times \ln P}{C}} &= \\
\underline{-\ln P + \sqrt{(\ln P)^2 - \frac{2 \times T \times \ln P}{2 \times \tau_b + \tau_c + \tau_{oh}}}} & \quad (9)
\end{aligned}$$

We have now defined an expression for finding the optimal number of checkpoints when using RRC for a given job and an error probability.

For a given job with $T = 500\mu s$ and a processor with $P_\tau = 0.85$, Figure 4 shows the AET for RRC at various number of checkpoints. Figure 4 clearly shows that there is a trade-off between the number of checkpoint and the bus communication in respect to AET. A high number of checkpoints reduces the AET; however, only to a given point where bus communication starts to dominate the AET.

The formulas defined in this section make it possible to define the optimal number of checkpoints for a given job and an error probability.

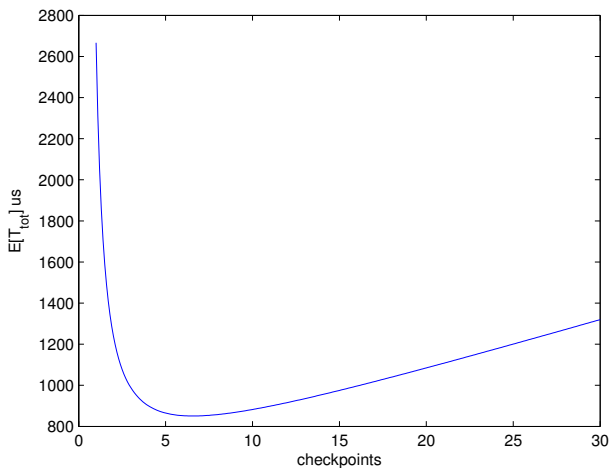


Figure 4: Average execution time including bus communication for rollback recovery with checkpointing at various number of check-points

B. Voting (Active replication)

In voting, a given job is executed on three or more processors, and at the end of the execution, majority

voting determines the correct response. If triple modular redundancy is used, the job is executed on three processors, and if an error occur in one of the processors and no error occur in the others, the correct result is given by majority voting. If two errors occur in one processor and the other two operate correctly, a final result is produced. However, if two errors occur in two different processors, three different responses exist, and no final result can be produced. For safety-critical applications, the redundancy can be increased such that the probability of not being able to determine correct response is very low. However, for general purpose systems such redundancy is too costly and re-execution is the alternative. However, formulas for re-execution using voting for AET is needed.

We assume given is a job with error-free execution time T , an error probability P (computed by Eq. (1)), and a number of n_p processors, and we define expressions for AET ($E[T_{exe,vot}]$) when not including communication overhead and the AET ($E[T_{tot,vot}]$) when communication overhead is included.

The probability that voting *cannot* be used is when the result from all processors are different. It occurs in two cases:

- an error in all n_p processors and
- an error in all but one processor,

and it is given as:

$$Q_s = \underbrace{(1 - P)^{n_p}}_{all_fail} + \underbrace{n_p \times P \times (1 - P)^{n_p - 1}}_{only_one_success} \quad (10)$$

Eq. (10) gives all cases when voting cannot be used, hence re-execution is required. We use this to compute the expected execution time (AET), which is:

$$\begin{aligned}
E[T_{exe,vot}] &= T \times E[N] = \frac{T}{1 - Q_s} = \\
&\frac{T}{1 - ((1 - P)^{n_p} + n_p \times P \times (1 - P)^{n_p - 1})} \quad (11)
\end{aligned}$$

for the execution of a job with error-free execution time of T on n_p processors.

The communication overhead is given by:

$$E[T_{com,vot}] = \frac{n_p \times \tau_b + \tau_c + \tau_{oh}}{1 - ((1 - P)^{n_p} + n_p \times P \times (1 - P)^{n_p - 1})} \quad (12)$$

where $n_p \times \tau_b$ is due to that each processor needs to communicate its response over the bus to the shared memory.

The total expected execution time is given from Eq. (11) and Eq. (12) as:

$$\begin{aligned}
E[T_{tot,vot}] &= E[T_{exe,vot}] + E[T_{com,vot}] = \\
&= \frac{T + n_p \times \tau_b + \tau_c + \tau_{oh}}{1 - ((1 - P)^{n_p} + n_p \times P \times (1 - P)^{n_p - 1})} \quad (13)
\end{aligned}$$

Figure 5 shows for a job with $T = 500\mu s$ and processors with $P_\tau = 0.85$, the AET for voting at various number

of processors. It comes clear that in voting there is a trade-off between the number of processors and the bus communication overhead. A high number of processors reduces the AET; however, only to a given point where bus communication starts to be the dominant contributor to AET. By the formulas defined in this section, it is possible to find the optimal number of processors for a given job at a certain error probability.

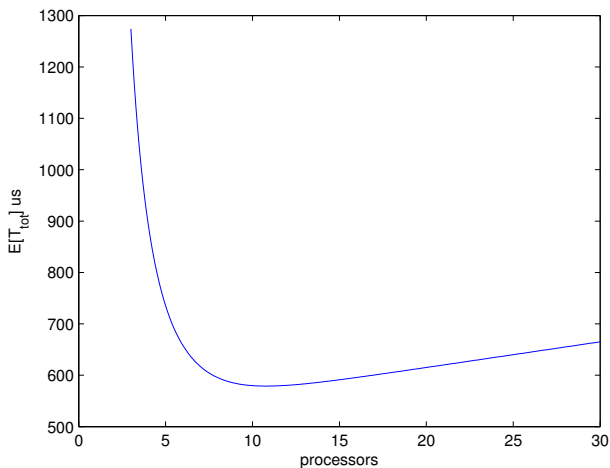


Figure 5: Average execution time including bus communication for voting at various number of processors

IV. SYSTEM-LEVEL OPTIMIZATION

Above, we defined formulas to optimize the AET using RRC and voting for a given job and an error probability. In this section, we assume given is the formulas defined above, a set of jobs, an MPSoC, and an error probability. We define three problems, P_v , P_{rrc} , and P_{v+rrc} , as:

- P_v : Given that *voting* is to be used, the problem P_v is to define the groups (clusters) of processors and assign jobs to each group (cluster),
- P_{rrc} : Given that *RRC* is to be used, the problem P_{rrc} is to select processor and the number of checkpoints for each job, and
- P_{v+rrc} : The problem P_{v+rrc} is to select voting or RRC for each job, and for each job using voting define clusters and assign jobs to the clusters, and for each job using RRC select the number of checkpoints and assign jobs to processors

where the objective for each problem is to optimize the fault-tolerance such that the overall AET including bus communication is minimized.

The problems P_v , P_{rrc} , and P_{v+rrc} are NP-complete as equivalence between them and multiprocessor scheduling problem (MSP), which is known to be NP-complete [11], can be shown as: a job is a job and a machine is a cluster of processors (group of processors).

We make use of integer linear programming (ILP) to exactly solve the problem. An ILP model can be described as follows:

minimize: $\mathbf{c}\mathbf{x}$,
subject to: $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, such that $\mathbf{x} \geq 0$,
where \mathbf{c} is a cost vector, \mathbf{A} is a constraint matrix, \mathbf{b} is a column vector of constants and \mathbf{x} is a vector of integer variables.

We make use of the following ILP model:

$$\begin{aligned} &\text{minimize: } T \\ &\text{subject to: } \sum_{j=1}^n A_{ij} \times x_{ij} - T \leq 0, 1 \leq i \leq m \\ &\quad \sum_{i=1}^m x_{ij} = 1, 1 \leq j \leq n \\ &\quad x_{ij} \in \{0, 1\}, 1 \leq i \leq m, 1 \leq j \leq n \\ &\quad T \geq 0 \end{aligned}$$

where

A_{ij} = processing time if cluster i is assigned job j
 $x_{ij} = 1$, if cluster i is assigned job j , otherwise 0
 T = average execution time (AET).

From the model, m clusters and n jobs need $m \times n + 1$ variables and $m \times n$ constraints. The problem size grows with m and n ; however, in practice there is a limit on the number of jobs and especially on the number of processors in an MPSoC. And therefore ILP is justified.

For the experiments we assume: ten jobs with fault-free execution time (T) of 500, 600, 700, ..., 1400 μs per job, respectively, an MPSoCs where the number of processors (proc) is ranging from two to nine, the overhead to be: $\tau_b = \tau_{oh} = 5\mu\text{s}$ and $\tau_c = 10\mu\text{s}$, and the probability of no error $P_\tau = 0.96$. We make use of `lp_solve` for the optimization [12].

As reference point in the experiments, called P_{ref} , we make use of a scheme using RRC with minimal bus communication; a single checkpoint (at the end of the job). We compare the results from the three problems (P_v , P_{rrc} and P_{v+rrc}) against P_{ref} .

The results are collected in Tables 1 and 2, and Figures 6 and 7. There is no result for P_v at two processors as voting requires a minimum of three processors. Table 1 and Figure 6 show the AET for P_v , P_{rrc} , P_{v+rrc} and P_{ref} . The results show that P_v , P_{rrc} and P_{v+rrc} produce better results than P_{ref} . The results for P_v is slightly better and in some cases much better than those of P_{ref} , while the results for P_{rrc} and P_{v+rrc} are significantly better than the results from P_{ref} .

Table 2 and Figure 7 show the relative improvement of P_v , P_{rrc} and P_{v+rrc} against P_{ref} . P_{rrc} and P_{v+rrc} produce results that are almost 50% better than those produced by P_{ref} . The results from P_v are in some cases almost as good as the results produced by P_{rrc} and P_{v+rrc} . However, often the results are only slightly better than P_{ref} . The results indicate that voting is a costly fault-tolerant technique; hence P_v does not generate as good results as P_{rrc} , and the combination of voting and RRC (P_{v+rrc}) does not result in significantly better AET than P_{rrc} . However, the results show that effective usage of P_{rrc} significantly reduces AET.

# proc	P_{ref}	P_v	P_{rrc}	P_{v+rrc}
2	23251	—	12521	12521
3	23251	13744	12521	12521
4	11627	11363	6325	6325
5	11627	10536	6325	6325
6	7785	6878	4218	4218
7	7785	6145	4218	4217
8	5871	5692	3164	3164
9	5871	4623	3164	3163

Table 1: AET for P_{ref} , P_v , P_{rrc} , and P_{v+rrc} at various number of processors

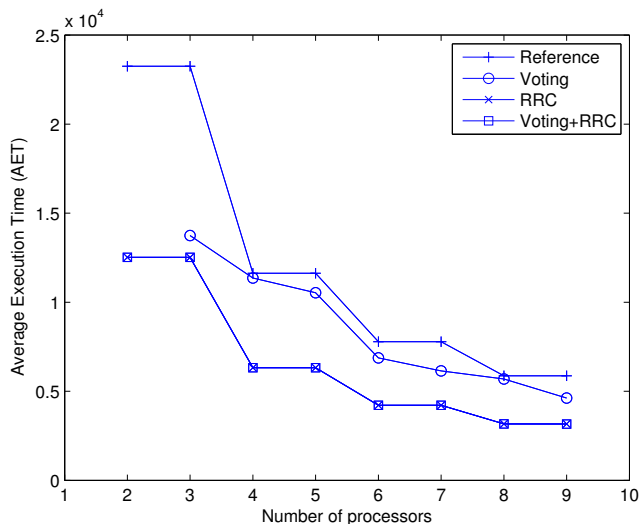


Figure 6: AET for P_{ref} , P_v , P_{rrc} , and P_{v+rrc}

V. CONCLUSIONS

As semiconductor technology, on the one hand, makes it possible to increase the performance (throughput) by enabling multi-processor System-on-Chip (MPSoC), but, on the other hand, increases the sensitivity to errors, especially soft (transient) errors, there is a need of mathematical framework for cost-effective fault-tolerance analysis for MPSoCs that are to be used in general purpose applications.

For general purpose applications the *average execution time (AET)* for a system is most important while ensuring fault-tolerance. In this paper, we have defined formulas for the AET for a job using two fault tolerance techniques, rollback recovery with checkpointing (RRC) and voting (active replication) where we for each scheme include bus communication overhead. We also take the system

# proc	P_v	P_{rrc}	P_{v+rrc}
2	—	46.15%	46.15%
3	40.88%	46.15%	46.15%
4	2.27%	45.60%	45.60%
5	9.38%	45.60%	45.60%
6	11.65%	45.82%	45.82%
7	21.07%	45.82%	45.83%
8	3.05%	46.11%	46.11%
9	21.26%	46.11%	46.12%

Table 2: Relative AET (%) for P_v , P_{rrc} , and P_{v+rrc} at various number of processors against P_{ref}

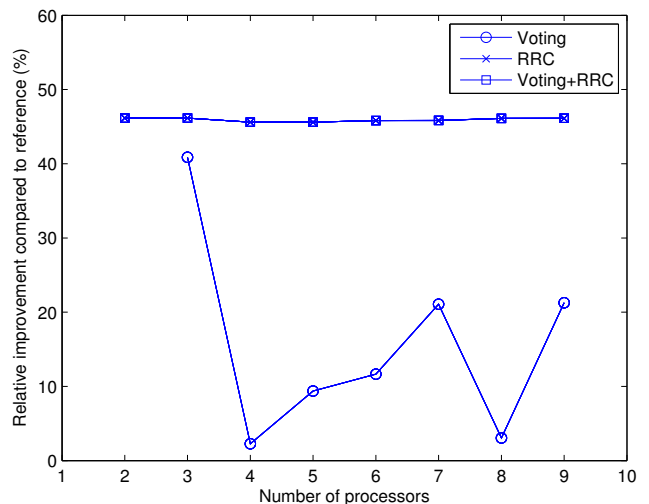


Figure 7: Relative AET (%) savings for P_v , P_{rrc} , and P_{v+rrc} against P_{ref}

perspective and optimize using ILP the usage of fault-tolerance such that the overall AET is minimal. The results show that voting is a costly fault-tolerant technique while RRC is shown to be a technique that if optimized properly can lead to significant savings of AET.

REFERENCES

- [1] S. Borkar, "Design challenges of technology scaling," *Micro*, *IEEE*, vol. 19, no. 4, pp. 23–29, Jul-Aug 1999.
- [2] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, 2007.
- [3] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann, 2008.
- [4] J. von Neumann, *Probabilistic logics and synthesis of reliable organisms from unreliable components*, C. Shannon and J. McCarthy, Eds. Princeton University Press, 1956.
- [5] K. Alstrom and J. Torin, "Future architecture for flight control systems," *Proceedings of the The 20th Conference on Digital Avionics Systems*, vol. 1, pp. 1B5/1–1B5/10, 2001.
- [6] A. Bertossi, A. Fusiello, and L. Mancini, "Fault-tolerant deadline-monotonic algorithm for scheduling hard-real-time tasks," *Proceedings of Parallel Processing Symposium*, pp. 133–138, April 1997.
- [7] R. Al-Omari, A. Somani, and G. Manimaran, "A new fault-tolerant technique for improving schedulability in multiprocessor real-time systems," *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, p. 10032.1, 2001.
- [8] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, 2005, pp. 864–869.
- [9] A. Ejlali, B. M. Al-Hashimi, M. T. Schmitz, P. Rosinger, and S. G. Miremadi, "Combined time and information redundancy for seu-tolerance in energy-efficient real-time systems," *IEEE Transactions on Very Large Scale Integrated Circuits and Systems*, vol. 14, no. 4, pp. 323–335, 2006.
- [10] Y. Cai, M. T. Schmitz, A. Ejlali, B. M. Al-Hashimi, and S. M. Reddy, "Cache size selection for performance, energy and reliability of time-constrained systems," in *ASP-DAC '06: Proceedings of Asia South Pacific design automation*, 2006, pp. 923–928.
- [11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [12] M. Berkelaar, "lpsolve 3.0," *Eindhoven University of Technology, Eindhoven, The Netherlands*, ftp://ftp.ics.ele.tue.nl/pub/lp_solve.