On-line techniques to adjust and optimize checkpointing frequency

Dimitar Nikolov[†], Urban Ingelsson[†], Virendra Singh[‡] and Erik Larsson[†]

Department of Computer Science[†]
Linköping University
Sweden

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

Abstract[1]

Due to increased susceptibility to soft errors in recent semiconductor technologies, techniques for detecting and recovering from errors are required. Roll-back Recovery with Checkpointing (RRC) is one well known technique that copes with soft errors by taking and storing checkpoints during execution of a job. Employing this technique, increases the average execution time (AET), i.e. the expected time for a job to complete, and thus impacts performance. To minimize the AET, the checkpointing frequency is to be optimized. However, it has been shown that optimal checkpointing frequency depends highly on error probability. Since error probability cannot be known in advance and can change during time, the optimal checkpointing frequency cannot be known at design time. In this paper we present techniques that are adjusting the checkpointing frequency on-line (during operation) with the goal to reduce the AET of a job. A set of experiments have been performed to demonstrate the benefits of the proposed techniques. The results have shown that these techniques adjust the checkpointing frequency so well that the resulting AET is close to the theoretical optimum.

I. Introduction

Recent semiconductor technologies have enabled fabrication of integrated circuits (ICs) that contain a very large number of transistors. These ICs offer a wide range of functionalities and provide high performance, which is usually achieved by implementing several processor cores on the same silicon die. While recent technologies enable fabrication of such powerful ICs, it has been noted that ICs manufactured using recent technologies are becoming increasingly susceptible to soft errors, [4], [6]. Therefore techniques for dealing with soft errors are required.

One well known technique that detects and recovers from soft errors is Roll-back Recovery with Checkpointing (RRC). This technique proposes taking and storing checkpoints while a job is being executed. A checkpoint represents a snapshot of the current state of the job at the time when the checkpoint is taken. To enhance error detection, a job is executed simultaneously on two independent processor nodes. If an error occurs, the error is detected by comparing the checkpoints taken from both processor nodes, because it is highly unlikely that both processor nodes experience the same error. Once an error is detected the job is restarted (rolled-back) from the last taken checkpoint. The cost of employing this technique is the need of two processor nodes and the extra time for taking and comparing the checkpoints, and eventual roll-back. The time cost is related to the checkpointing frequency. Higher checkpointing frequency enables errors to be detected earlier, because of the shorter intervals between checkpoints, and reduces the time spent in re-execution, but it increases the overall time cost due to the more frequent checkpointing and comparison operations. Lower checkpointing frequency reduces the total amount of checkpoints during the execution of the job and thereby the cost imposed of the checkpointing and comparison operations, but the penalty paid in case of an error is increased since there is a larger interval between checkpoints which will have to be re-executed.

In [8] is shown that it is possible to calculate an optimal number of checkpoints, and thus an optimal checkpointing frequency, for a given error probability and a fault-free execution time. Employing the optimal checkpointing frequency minimizes the average execution time (AET), i.e. the expected time for a job to complete. Many papers have addressed the problem of finding the optimal checkpointing frequency, [1], [7], [8], [9], [10], [11], [12], [13], [14], [15] and they have reported that the optimal checkpointing frequency highly depends on the failure rate (error probability). However in reality, the error probability cannot be known at design time and it can further change during time in operation. Therefore in this paper we present techniques that adjust the checkpointing frequency on-line (during operation) with the goal to minimize the AET.

II. Preliminaries

We assume an MPSoC architecture, described in Figure 1, which consists of n processor nodes, a shared memory, and a compare & control unit. The processor nodes are general-purpose processor nodes that include

[1]The research is partially supported by The Swedish Foundation for International Cooperation in Research and Higher Education (STINT) by an Institutional Grant for Younger Researchers.

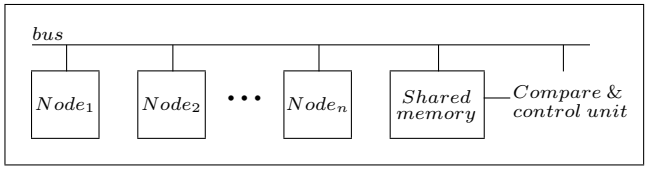

Figure 1: MPSoC architecture with n processor nodes, a shared memory and a compare & control unit

private memory, and the shared memory, which is common for all processor nodes, is used for communication between processors. The compare & control unit, added for fault tolerance, detects whether errors have occurred by comparing the contexts (checkpoints) of two processors executing the same job at predetermined intervals. We address errors that occur in the processors, and we assume that errors that occur elsewhere (buses and memories) can be handled by other fault-tolerant techniques such as error correction codes.

In RRC, each job is executed concurrently on two processors and a number of checkpoints are inserted to detect errors. A given job is divided into a number of *execution segments* and between every execution segment there is a *checkpoint interval*. The checkpoint interval represents the time required to take a checkpoint. Figure 2 illustrates the execution segments and the inserted checkpoint intervals. When a job is executed and a checkpoint is reached, both processors send their respective contexts to the compare & control unit. The compare & control unit compares the contexts. If the contexts differ, meaning that an error has occurred during the last execution segment, the last execution segment is to be re-executed. In the case that the contexts of the processors do not differ, meaning that there is no error, the execution proceeds with the next execution segment.

As discussed earlier in section I, checkpointing at higher or lower frequency impacts the time cost in different manners. In [8], Väyrynen *et al.* have addressed the problem of obtaining an optimal number of checkpoints that minimizes the average execution time (AET). They proposed a mathematical framework for the analysis of AET, and presented an equation for computing the optimal number of checkpoints. The AET when applying RRC on a job is given as:

$$AET(P,T) = \frac{T + n_c \times (2 \times \tau_b + \tau_c + \tau_{oh})}{\sqrt[n_c]{(1-P)^2}} \qquad (1)$$

where P is the error probability per time unit, T is the fault-free execution time, n_c is the number of checkpoints, and τ_b, τ_c and τ_{oh} are time parameters due to checkpoint overhead. Given Eq. (1), Väyrynen *et al.* showed that the optimal number of checkpoints (n_c) is given as:

$$n_c(P,T) = -\ln(1-P) + \sqrt{(\ln(1-P))^2 - \frac{2 \times T \times \ln(1-P)}{2 \times \tau_b + \tau_c + \tau_{oh}}} \qquad (2)$$

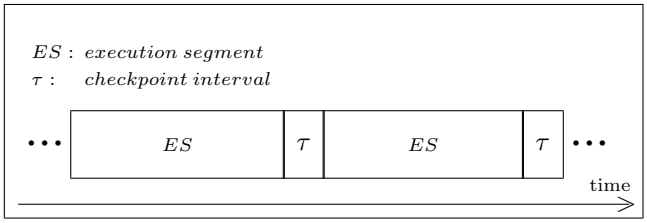

Figure 2: Execution segments and checkpoint intervals

Using the optimal number of checkpoints, n_c, (Eq. (2)), the optimal AET can be calculated with Eq. (1).

The computation of optimal number of checkpoints, and thus optimal checkpointing frequency, requires the following parameters: error probability (P), fault-free execution time (T), and parameters for checkpoint overhead (τ_b, τ_c and τ_{oh}). The parameters for checkpoint overhead can be estimated at design time; however it is difficult to accurately estimate error probability. The real error probability cannot be known at design time, it is different for different ICs, and it is not constant through the lifetime of an IC [1] [2] [3] [5].

III. Techniques for on-line adjustment of checkpointing frequency

As shown in the previous section, optimal checkpointing frequency depends on the error probability. Since error probability cannot be known in advance, and it can change during time in operation, the optimal checkpointing frequency cannot be known at design time. Therefore in this section we present two on-line techniques that adjust the checkpointing frequency during operation with the aim to optimize RRC. These techniques adjust the checkpointing frequency based on estimates on error probability generated during operation. One way to provide accurate error probability estimates is to extend the architecture described earlier (Figure 1) by employing a history unit that keeps track on the number of successful (no error) executions of execution segments (n_s) and the number of erroneous execution segments (execution segments that had errors) (n_e). Having these statistics, error probability can be estimated during time, periodically or aperiodically. Thus we propose a periodic approach,Periodic Probability Estimation (PPE), and an aperiodic, Aperiodic Probability Estimation (APE). For both approaches we need some initial parameters, *i.e.* initial estimate on error probability and adjustment period. It should be noted, that the adjustment period is kept constant for PPE, while for APE it is tuned over time.

A. Periodic Probability Estimation

PPE assumes a fixed adjustment period,T_{adj}, and estimates the error probability,p_{est}, using the following expression:

$$p_{est} = \frac{n_e}{n_e + n_s} \qquad (3)$$

where n_s is the number of successful (no error) executions of execution segments and n_e is the number of erroneous

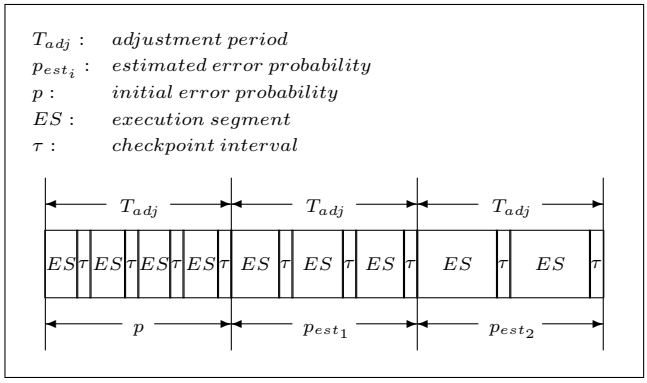

Figure 3: Graphical presentation of PPE

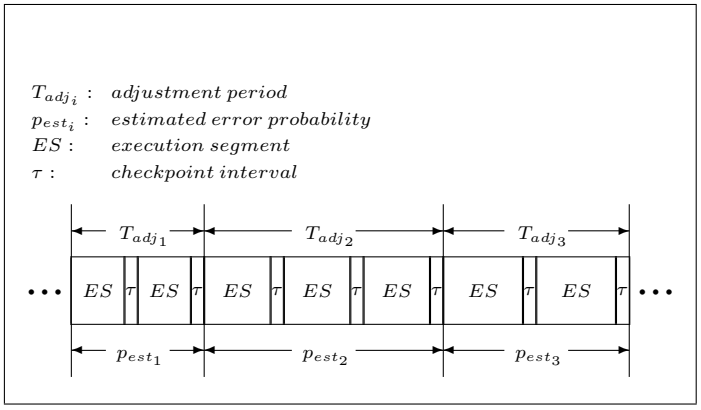

Figure 4: Graphical presentation of APE

execution segments. As can be seen from Figure 3 estimates on error probability, p_{est}, are calculated periodically at every T_{adj}. The value of p_{est} is used to obtain the optimal number of checkpoints, n_c, by applying Eq. (2). During an adjustment period, n_c equidistant checkpoints are taken. So the checkpointing frequency is adjusted periodically (after every T_{adj}) according to the changes of the error probability estimates.

B. Aperiodic Probability Estimation

APE adjusts the checkpointing frequency by elaborating on both T_{adj} and p_{est}. The idea for this approach comes from the following discussion. As this approach estimates the error probability, it is expected that during operation the estimates will converge to the real values, so we should expect changes on the estimated error probability during time. These changes can be used to adjust the length of T_{adj}. If the estimates on error probability start decreasing, that implies that less errors are occurring and then we want to decrease the checkpointing frequency, so we increase the adjustment period (the checkpointing frequency is determined as number of checkpoints during adjustment period, n_c/T_{adj}). On the other hand, if the estimates on error probability start increasing, that implies that errors occur more frequently, and to reduce the time spent in re-execution we want to increase the checkpointing frequency, so we decrease the adjustment period.

If the control & compare unit encounters that error probability has not changed in two successive adjustment periods, it means that during both adjustment periods the system has done a number of checkpoints which is greater than the optimal one. This can be observed by the following relation which is derived from Eq. (2):

$$2 \times n_c(P, T_{adj}) > n_c(P, 2 \times T_{adj}) \tag{4}$$

In APE, error probability is estimated in the same manner as PPE, *i.e.* using Eq. (3). What distinguishes this approach from PPE, is that the adjustment period is updated during time. Eq. (5) describes the scheme for updating the adjustment period.

$$\begin{aligned} &if \quad p_{est_{i+1}} > p_{est_i} \quad then \\ &\quad T_{adj_{i+1}} = T_{adj_i} - T_{adj_i} \times \alpha \\ &else \\ &\quad T_{adj_{i+1}} = T_{adj_i} + T_{adj_i} \times \alpha \end{aligned} \tag{5}$$

The APE approach is illustrated in Figure 4. After every T_{adj} time units, the control & compare unit, computes a new error probability estimate ($p_{est_{i+1}}$) using the Eq. (3). The latest estimate ($p_{est_{i+1}}$) is then compared against the previous value (p_{est_i}). If estimation of error probability increases, meaning that during the last adjustment period, (T_{adj_i}), more errors have occurred, the next adjustment period, ($T_{adj_{i+1}}$), should be decreased to avoid expensive re-executions. However, if the estimation of error probability decreases or remains the same, meaning that less or no errors have occurred during the last adjustment period, (T_{adj_i}), the next adjustment period, ($T_{adj_{i+1}}$), should be increased to avoid excessive checkpointing. The new value for the adjustment period ($T_{adj_{i+1}}$) together with the latest estimate on error probability ($p_{est_{i+1}}$) are used to calculate the optimal number of checkpoints, n_c, that should be taken during the following adjustment period.

IV. Experimental Results

To conduct experiments and thus evaluate the accuracy of the presented techniques we have developed a simulator tool. The simulator uses the following inputs: initial estimated error probability, p, adjustment period, T_{adj}, real error probability distribution, P, and fault-free execution time for the simulated job, T. We have simulated three approaches: Periodic Probability Estimation (PPE), Aperiodic Probability Estimation (APE), and Baseline Approach (BA). Each approach uses the following inputs: initial estimated error probability, p, and adjustment period, T_{adj}. PPE and APE were described earlier in III-A and III-B respectively. BA uses its inputs, *i.e.* the initial estimated error probability, p, and the adjustment period, T_{adj}, and computes an optimal number of checkpoints, n_c,

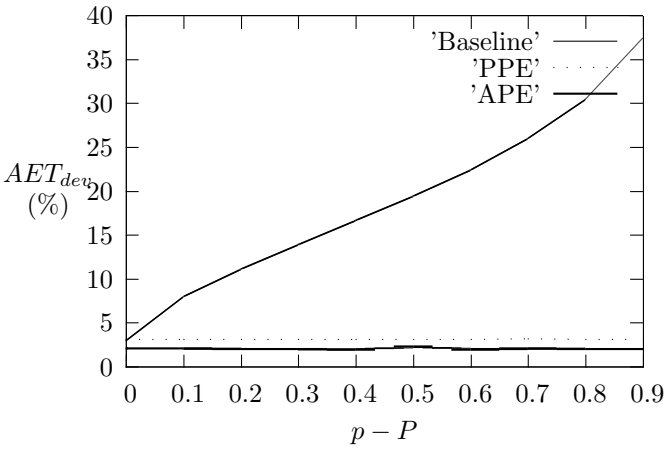

Figure 5: Relative deviation from optimal AET (%) for constant *real* error probability $P = 0.01$

for these inputs using Eq. (2). Further, it takes checkpoints at a constant frequency n_c/T_{adj}, so no adjustments are done during execution.

We made experiments to determine an appropriate value for α parameter in APE. The experiment was repeated for different values for the real error probability and for the adjustment period T_{adj}, and it was found that out of the considered values, $\alpha = 0.15$ provided the best results, *i.e.* the lowest deviation from optimal AET.

We conducted two sets of experiments. In the first set, we have examined the behavior of the approaches when the real error probability is constant during time, while in the second set, the real error probability changes over time, following a predefined profile.To get the AET for the simulated approaches, each approach is simulated for 1000 times with the same inputs.

In the first set of experiments, we compare the three simulated approaches: PPE, APE and BA against the optimal solution in terms of AET (%). The optimal solution is obtained by using the equations proposed by Väyrynen *et al.* [8] and using the real error probability, P, and fault-free execution time, T, as inputs for these equations. In Figure 5 we present on the y-axis the deviation of the AET obtained from the simulated approaches, relative to the optimal AET in %. On the x-axis we present the difference between the initial estimated error probability, p, and the real error probability, P. We assume a constant *real* error probability, $P = 0.01$, and fault-free execution time $T = 1000000$ time units. We choose the adjustment period to be $T_{adj} = 1000$ time units, and then simulate the approaches with different values for the initial estimated error probability, p. One can observe from Figure 5 that APE and PPE always perform better than the BA approach, and they do not deviate much from the optimal solution. Furhter, Figure 5 shows that APE performs slightly better than PPE.

In the second set of experiments, we have examined the behavior of the approaches when *real* error probability changes over time. For this purpose, we define different error probability profiles showing how error probability changes over time, and then we run simulations for each of these profiles. Three probability profiles are presented in Table I. We assume that the probability profiles are repeated periodically over time. The results in Table II present the deviation of the AET obtained from the simulated approaches, relative to the fault-free execution time in %. For these simulations, we choose the adjustement period to be $T_{adj} = 1000$ time units and the initial estimated error probability to be equal to the real error probability at time 0, *i.e.* $p = P(0)$. We assume fault-free execution time of $T = 1000000$ time units. As can be seen from Table II, both PPE and APE perform far better than BA, with a very small deviation in average execution time relative to the fault-free execution time. Again we notice that APE gives slightly better results than PPE approach.

$$P1(t) = \begin{cases} 0.01, & 0 \le t < 200000 \\ 0.02, & 200000 \le t < 400000 \\ 0.03, & 400000 \le t < 600000 \\ 0.02, & 600000 \le t < 800000 \\ 0.01, & 800000 \le t < 1000000 \end{cases}$$

$$P2(t) = \begin{cases} 0.02, & 0 \le t < 350000 \\ 0.01, & 350000 \le t < 650000 \\ 0.02, & 650000 \le t < 1000000 \end{cases}$$

$$P3(t) = \begin{cases} 0.01, & 0 \le t < 90000 \\ 0.10, & 90000 \le t < 100000 \end{cases}$$

Table I: Error probability profiles

Probability Profile	Approaches		
	Baseline	PPE	APE
P1	55.93%	4.50%	2.84%
P2	50.69%	4.53%	2.74%
P3	56.02%	4.65%	2.50%

Table II: Relative deviation from fault-free execution time (%) for variable real error probability

V. Conclusion

Fault tolerance becomes a challenge with the rapid development in semiconductor technologies. However, many fault tolerance techniques have a negative impact on performance. For one such technique, Roll-back Recovery with Checkpointing, which inserts checkpoints to detect and recover from errors, the checkpointing frequency is to be optimized to mitigate the negative impact on performance. However, the checkpointing frequency depends on error probability which cannot be known in advance.

In this paper we have proposed two techniques that adjust the checkpointing frequency during operation with the aim to reduce the average execution time. These two techniques are a periodic approach, where the adjustment is done periodically based on the error probability that is estimated after every T_{adj}, and an aperiodic approach where T_{adj} is tuned over time. To perform experiments we have implemented a simulator. The simulator runs the proposed approaches given the following inputs: initial estimated error probability, adjustment period, real error probability and expected fault-free execution time of the simulated job. By presenting the results from the simulator, we demonstrate that both proposed techniques achieve results comparable to the theoretical optimum. From the results we also notice, that the proposed aperiodic approach gives slightly better results than the periodic approach, in terms of average execution time.

References

[1] I. Koren and C. M. Krishna, "Fault-Tolerant Systems", Morgan Kaufman, 1979.

[2] E.H. Cannon, A. KleinOsowski, R. Kanj, D. D. Reinhardt, and R. V. Joshi, "The Impact of Aging Effects and Manufacturing Variation on SRAM Soft-Error Rate", *IEEE Trans. Device and Materials Reliability*, vol. 8, no. 1,pp. 145-152, March 2008

[3] V. Lakshminarayanan, "What causes semiconductor devices to fail?", Centre for development of telematics, Bangalore, India – *Test & Measurement World*, 11/1/1999.

[4] V. Chandra, and R. Aitken, " Impact of Technology and Voltage Scaling on the Soft Error Susceptibility in Nanoscale CMOS", *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pp. 114-122, Oct. 2008

[5] T. Karnik, P. Hazucha, and J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes", *IEEE Trans. on Dependable and secure computing*, vol. 1, no. 2, April-June 2004

[6] J. Borel, "European Design Automation Roadmap", 6th Edition, March 2009

[7] D. K. Pradhan and N. H. Vaidya, "Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture", *IEEE Transactions on computers*, vol. 43, no. 10, pp. 1163-1174, October 1994

[8] M. Väyrynen, V. Singh, and E. Larsson, " Fault-Tolerant Average Execution Time Optimization for General-Purpose Multi-Processor System-on-Chips", *Design Automation and Test in Europe (DATE 2009)*, Nice, France, April, 2009.

[9] Y. Ling, J. Mi and X. Lin, "A Variational Calculus Approach to Optimal Checkpoint Placement", *IEEE Transactions on computers*, vol. 50, no.7, pp. 699-708, July 2001.

[10] J.L. Bruno and E.G. Coffman,"Optimal Fault-Tolerant Computing on Multiprocessor Systems", *Acta Informatica*, vol. 34, pp. 881-904, 1997.

[11] E.G. Coffman and E.N. Gilbert, "Optimal Strategies for Scheduling Checkpoints and Preventive Maintenance", *IEEE Trans. Reliability*, vol. 39, pp. 9-18, Apr. 1990.

[12] P. L'Ecuyer and J. Malenfant, "Computing optimal checkpointing strategies for rollback and recovery systems", *IEEE Trans. Computers*, vol. 37, no. 4, pp. 491-496, 1988.

[13] E. Gelenbe and M. Hernandez, "Optimum Checkpoints with Age Dependent Failures", *Acta Informatica*, vol. 27, pp. 519-531, 1990.

[14] C.M. Krishna, K.G. Shin, and Y.H. Lee, "Optimization Criteria for Checkpoint Placements", *Comm. ACM*, vol. 27, no. 10, pp. 1008-1012, Oct. 1984.

[15] V.F. Nicola, "Checkpointing and the Modeling of Program Execution Time", *Software Fault Tolerance*, M.R. Lyu, ed., pp. 167-188, John Wiley&Sons, 1995.