Determination of the Optimum Degree of Redundancy for Fault-prone Many-Core Systems

Armin Runge, University of Würzburg, Am Hubland, 97074 Würzburg, Germany,
runge@informatik.uni-wuerzburg.de

Abstract

The increasing transistor integration capacity will entail hundreds of processors on a single chip. Further, this will lead to an inherent susceptibility to errors of these systems. To obtain reliable systems again, various redundancy techniques can be applied. Of course, the usage of those techniques involves a significant overhead. Therefore, the identification of the optimal degree of redundancy is an important objective. In this paper we focus on core-level redundancy and checkpointing rollback-recovery. A model to determine the optimal degree of spatial and temporal redundancy regarding the minimal expected execution time will be introduced. Further, we will show that in several cases, the minimal expected execution time is achieved just by a simultaneous combination of both techniques, spatial redundancy and temporal redundancy.

1 Introduction

Continued technology scaling leads to an ever increasing transistor integration capacity. Following this trend, tens of billions of transistors will be available on a single chip. Unfortunately, the performance gain is limited when the increasing number of transistors is used for larger processor cores. Performance increase is governed by Pollack's Rule [1], which states that microprocessor performance increase is roughly proportional to the square root of the increase in complexity (i.e. area). Applying Pollack's Rule inversely, a smaller core that requires half the area has only a performance loss of about 30%, but a large power reduction since power consumption decreases linearly with size. A system consisting of hundreds of small cores has therefore a higher computational throughput relative to a system with a single large core. Many small cores have also further benefits, such as the ability to turn off individual cores when they are not needed.

Otherwise, the shrinking transistor size leads to an increasing variability in performance and reliability [2]. This is due to static variations (e.g. random dopant fluctuations) and dynamic variations (e.g. heat flux), whereas the latter are time and context variant. That leads to irreversible device degradation over time as well as frequent and intermittent single-event upsets (so called soft-errors), caused by alpha particles or neutrons hitting a latch.

Therefore, our expectation of a future system on chip (SoC) architecture is a steadily aging, and highly error-prone system with hundreds of processing elements (PEs). Intermittent failures, like single-event upsets, cause that one-time factory testing becomes insufficient. Hence, it will be a challenge to yield reliable systems anymore.

A possible solution to enhance the reliability of such a system is the utilization of various forms of redundancy. In this paper we will focus on the combination of spatial and temporal redundancy at core-level. Besides the redundancy level, e.g. circuit-level [3] or core-level [4], another important parameter of spatial redundancy is the degree of redundancy, e.g. dual modular redundancy (DMR). However, DMR allows only error detection, triple modular redundancy (TMR) or even n modular redundancy (NMR) enables error correction. Temporal redundancy techniques, like checkpointing and rollback-recovery [5], perform the same task several times on the same resource in case of a failure. As the error probability of a task increases with its runtime, the task is partitioned into segments. After each segment the state of this task is stored, such that a rollback to the last validated state is possible. Therefore, each segment has a lower error probability as the whole task.

In this paper we will show that there exists an optimal number of PEs when using spatial redundancy, and an optimal number of checkpoints when using temporal redundancy. This is due to the communication overhead. We will further show that the minimal expected execution time will be achieved just by a combination of both techniques in many cases. This also holds for a resource constrained system, where only a restricted number of PEs is available.

The remainder of the paper is organized as follows. In Section 2 we present related work. Section 3 describes our system model and further assumptions. Section 4 discusses three different redundancy techniques and shows that an optimal degree of redundancy exists regarding the minimal expected execution time. In Section 5 we analyze a resource constrained system and a group of tasks. Finally, this paper closes with a conclusion and an outlook to future work in Section 6.

2 Related Work

Both techniques, spatial and temporal redundancy, are known research areas for some time. The foundati-

ons of reliability enhancement were already laid by J. von Neumann in the 1950s [6]. Currently, spatial redundancy is mainly deployed in safety-critical domains like avionics [7] or fault-tolerant server systems, such as the HP Integrity NonStop [8]. These systems often premise a dedicated connection between redundantly used PEs. LaFrieda et al. [9] and Sanchez et al. [10] examined the benefits as well as the communication overhead of dynamic coupled cores (DMR), which share their states over a bus and a network on chip (NoC), respectively. We also assume that there is no dedicated connection for state exchange between core groups. Sloan and Kumar [11] proposed a scalable NMR framework for error-prone chip multiprocessors, which supports in-network fault tolerance for low voting latency. They have assumed a constant checkpointing frequency as their focus was not on the determination of the optimal degree of redundancy. An analytical examination of the degree of spatial redundancy was also not performed by them. Huang and Xu [4] proposed an analytical method to characterize the lifetime reliability of many-core processors. They have analyzed the reliability of many-core processors and various redundancy configurations. Though, temporal redundancy was considered by neither of them.

Most papers targeting checkpointing and rollback-recovery (temporal redundancy) presuppose error-detecting PEs. Therefore, there must be only one correct PE in the system and there is no need to exchange any process states. Bruno and Coffman [12] considered the optimal schedule of checkpoints for a multiprocessor system. But in their work it is assumed that a job will only run on a single PE, the remaining PEs are thus spare parts. The authors in [13] and [14] also determined the optimal checkpointing frequency regarding the minimal expected job execution time. However, their calculations supposed an instantaneous failure detection and an immediate repetition of the failed task. Since soft-errors are hard to detect by a PE itself, we assume that errors are only detectable by state comparison between PEs.

There are also some papers in the field of combined spatial and temporal redundancy: Bougeret et al. [15] and Yi et al. [16] also assumed instantaneous failure detection. Fereira et al. [17] examined processor replication as a primary fault tolerance mechanism and checkpoint/restart as a secondary mechanism. They developed a MPI library but no reliability analysis was performed. Väyrynen et al. [18] showed that there is an optimal number of redundancy regarding the minimal expected execution time and applying either temporal or spatial redundancy. Their work is most related to the work presented in this paper, due to the similarity of the system model. A simultaneous combination of both techniques was not analyzed by them, yet.

3 System Model and Assumptions

For the following calculations we assume a homogeneous many-core system. The considered system is highly error-prone and the errors of the processing elements are independent and identically distributed. For simplicity we assume that for Section 4 only one single task exists in the system. The available processing elements communicate via a NoC. This assumption is not adherent for our model but we expect that communication costs for a message increase with the number of recipients. The used architecture is therefore not inherently capable of performing broadcast transmissions. Each PE stores its context to a dedicated and reliable memory region to allow a rollback in case of failure. The PEs also generate a fingerprint or checksum of the stored context, which they exchange with each other periodically to compare their own context with the context of each other PE (coordinated checkpointing). For instance, a fingerprint can be calculated from a checksum on all register values and stored values since the last validation. These fingerprint comparisons are necessary, since we assume that the PEs possess no dedicated hardware for error detection.

The system is confirmed if at least k out of n possible fingerprints coincide with each other. In case of a failure, i.e. if less than k messages coincide, all PEs rollback to the last valid state to recalculate the recent task. The chosen model for our homogeneous many-core system is thus the k-out-of-n system model (cf. [19], [20]). The probability of k or more successes is given by:

$$p_{sys}(n,k) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}, \qquad (1)$$

wherein n is the number of PEs available in the system, k is the minimum number of correct working PEs, and p the probability for correct execution of a given task on a single PE. Function p is time-dependent, as the probability for correct execution decreases with the runtime of a given piece of work. Then, (1) can be written as:

$$p_{sys}(n,k,t) = \sum_{i=k}^{n} \binom{n}{i} p(t)^i (1-p(t))^{n-i}. \qquad (2)$$

We further assume that k depends on n, e.g. $k = \lceil \frac{n}{2} \rceil$. Therefore, we will abbreviate $p_{sys}(n,k,t)$ in the following $p_{sys}(n,t) = p_{sys}(n, \lceil \frac{n}{2} \rceil, t)$.

It should be noted that an increased number of used PEs does not necessarily lead to an increase in reliability. For instance, if the success probability of one PE is only $p(t) = 0.3$, and $k = \lceil \frac{n}{2} \rceil$ like previously assumed, p_{sys} decreases with n. This occurs because the expected number of coinciding values is only about $n \cdot p < k$. A possibility to enhance p_{sys} is a smaller k, but this also results in an increased probability for an undetected error. This is the reason why we have

chosen $k = \lceil \frac{n}{2} \rceil$, inspired by the known k-out-of-$2k$ system model (cf. [19]).

4 Redundancy Techniques

As already stated, it is our objective to enhance the reliability of an error-prone many-core system by the application of spatial as well as temporal redundancy. Thereby, an important aspect is the degree of redundancy, since a high degree involves a major overhead (e.g. communication costs). In the following subsections we calculate the expected execution time of a task on a highly error-prone many-core system using spatial, temporal, and combined spatial and temporal redundancy.

In the failure free case, the runtime of the task to be calculated is t. The arising overhead in case of a fault is t_f, which occurs due to the required rollback and restart. Transmitting a fingerprint of the context from one PE to another PE is position-independent and takes time t_c.

4.1 Spatial Redundancy

To enhance reliability, the task is calculated on n PEs in parallel (active replication). After the task termination (at time t), all PEs exchange their context states for fault detection. Obviously, the failure probability is high if n is small and therefore less redundancy is used. The probability for a successful error correction is also low, which leads to a high failure probability of the whole system. But please note that errors should be still identifiable.

Though, if a high degree of redundancy is used, much more errors could be corrected. That means, if less than k errors occurred (e.g. $k = \lceil \frac{n}{2} \rceil$), the system can calculate the task successfully. However, the communication costs are rising with the number of PEs used, since each PE has to share its context with all other PEs. In the following we address the question of which degree of redundancy results in the minimal expected execution time.

As already mentioned, we assume that the fingerprint exchange between n PEs takes $n \cdot t_c$ time. This exchange is necessary at least once (failure free case), so that in the case of j failed attempts to compute the task, the fingerprint has been exchanged $j+1$ times. Each failed attempt takes also t time, as failures can not be detected until the fingerprints have been compared. The expected execution time for spatial redundancy is:

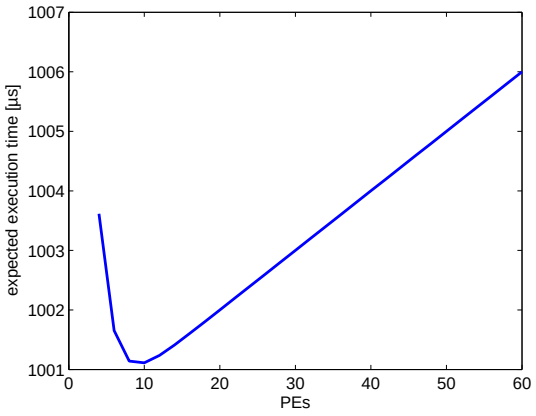


Figure 1 Expected execution time using spatial redundancy and n PEs.

$$
\begin{aligned}
&E[t_{run}(n)] \\
&= \sum_{i=1}^{\infty} (1 - p_{sys}(n))^{(i-1)} \cdot p_{sys}(n) \\
&\quad \cdot (i \cdot (t + n \cdot t_c) + (i-1) \cdot t_f) \\
&= p_{sys}(n) \cdot (t + n \cdot t_c) \cdot \sum_{i=1}^{\infty} (1 - p_{sys}(n))^{(i-1)} \cdot i \\
&\quad + p_{sys}(n) \cdot t_f \sum_{i=1}^{\infty} (1 - p_{sys}(n))^{(i-1)} \cdot (i-1) \\
&= \frac{p_{sys}(n)}{1 - p_{sys}(n)} \cdot (t + n \cdot t_c) \cdot \sum_{i=1}^{\infty} (1 - p_{sys}(n))^{i} \cdot i \\
&\quad + p_{sys}(n) \cdot t_f \sum_{i=0}^{\infty} (1 - p_{sys}(n))^{i} \cdot i \\
&= \frac{t + n \cdot t_c}{p_{sys}(n)} + \frac{t_f \cdot (1 - p_{sys}(n))}{p_{sys}(n)} \qquad (3)
\end{aligned}
$$

Please note that p_{sys} in (3) is a function of runtime t and n. However, since we want to minimize the duration of a task with a specific runtime, we can assume that this time is constant and so p_{sys} becomes a function of n.

Figure 1 shows the expected execution time for spatial redundancy on a many-core system with n PEs ranging from 4 to 60. To achieve a clearer depiction, only even numbers of PEs were considered here, since $k = \lceil \frac{n}{2} \rceil$ causes that $p_{sys}(2i-1) \leq p_{sys}(2i)$, $i \geq 2 \wedge i \in \mathbb{N}$. The expected execution time for $n = 2$ PEs is also not depicted here, as this time is very large (1224.1µs).

We assumed exponentially distributed errors with $\lambda := 0.0001$. This assumption was made for reasons of simplicity, i.e. also the Weibull distribution can be used. We further assumed $t := 1000$µs, $t_f := 1.0$µs, $t_c := 0.1$µs, and $k := \lceil \frac{n}{2} \rceil$.

With a small degree of redundancy the expected runtime is high, as the failure probability is high, too.

With the increase in redundancy the reliability improves. However, the overhead (here mainly the communication costs) increases also. So, after a certain degree of redundancy (for this example, about 10 PEs) the expected runtime even rises.

4.2 Temporal Redundancy

In this approach the task is calculated only on two PEs. Both PEs exchange their contexts for failure detection at regular intervals. Due to the lack of spatial redundancy (number of PEs), both fingerprints must coincide for a failure free computation. The task is split in n_c segments of equal size $t_{n_c} = t/n_c$, and the contexts are exchanged and compared after the computation of each segment. The probability of success for a single PE computing a task with length $t = 1000\mu s$ and with exponentially distributed failures is $p(t) = e^{-\lambda \cdot t} = 0.905$. Using 10 checkpoints, the PE has to compute only $1/10$ of the task between two checkpoints, which leads to a success probability of $p(t/10) = e^{-\lambda \cdot t/10} = 0.990$. The probability of a failure for the whole task does not change by the use of checkpointing as $p(t/10)^{10} = p(t)$, but the inevitable time loss in case of a failure changes. A high number of checkpoints entails a short inter-checkpoint time, and therefore the expected re-execution time in the event of a failure decreases. Unfortunately, the communication costs are rising with the number of checkpoints and hence the optimal number of checkpoints is unapparent. The expected execution time for temporal redundancy is:

$$\begin{aligned} E[t_{run}(n_c)] &= n_c \cdot \sum_{i=1}^{\infty} (1 - p_{sys}(t_{n_c}))^{(i-1)} \cdot p_{sys}(t_{n_c}) \\ &\quad \cdot (i \cdot (t_{n_c} + 2 \cdot t_c) + (i-1) \cdot t_f) \\ &= \frac{t + 2 \cdot n_c \cdot t_c}{p_{sys}(t_{n_c})} + \frac{n_c \cdot t_f \cdot (1 - p_{sys}(t_{n_c}))}{p_{sys}(t_{n_c})} \end{aligned} \tag{4}$$

Here, p_{sys} is a function of time, or rather the time to calculate a checkpoint segment, as the number of used PEs is constant $n = 2$. The expected execution time for temporal redundancy with n_c checkpoints is depicted in **Fig. 2**. We assumed exponentially distributed errors with $\lambda := 0.0001$, $t := 1000\mu s$, $t_f := 1.0\mu s$, $t_c := 1.0\mu s$, and $n = k = 2$.

As already seen at spatial redundancy, the expected execution time function has a global minimum. First, the execution time decreases, as the time loss in case of a failure decreases. However, communication costs predominate the execution time when too many checkpoints are inserted. Here, the minimal expected execution time is achieved with $n_c = 10$ checkpoints.

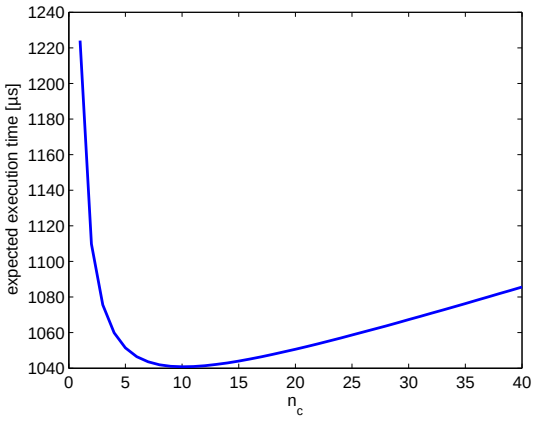


Figure 2 Expected execution time using temporal redundancy and n_c checkpoints.

4.3 Combination of Temporal and Spatial Redundancy

In the previous subsections we have shown that there is a non-trivial global minimum for the expected execution time using both spatial and temporal redundancy. We will now address the problem of determining the optimal number of PEs and checkpoints regarding the minimal expected execution time with combined spatial and temporal redundancy.

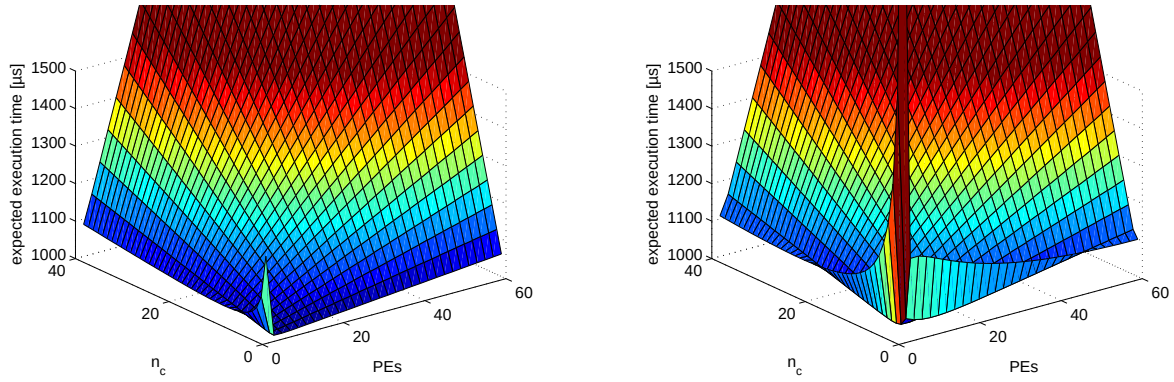
The task will be executed on n PEs, whereas n_c checkpoints are inserted. After each checkpoint the PEs exchange their context-fingerprints among each other, whereby the communication costs increase linear with n and n_c. To enhance a given reliability p_{sys} both the number of used PEs and the number of checkpoints can be varied (taking into account the remark to (2)).

To determine the minimal expected execution time for a task, the global minimum of the binary function $E[t_{run}(n, n_c)]$ has to be calculated:

$$\begin{aligned} &E[t_{run}(n, n_c)] \\ &= n_c \cdot \sum_{i=1}^{\infty} (1 - p_{sys}(n, t_{n_c}))^{(i-1)} \cdot p_{sys}(n, t_{n_c}) \\ &\quad \cdot (i \cdot (t_{n_c} + n \cdot t_c) + (i-1) \cdot t_f) \\ &= \frac{t + n \cdot n_c \cdot t_c}{p_{sys}(n, t_{n_c})} + \frac{n_c \cdot t_f \cdot (1 - p_{sys}(n, t_{n_c}))}{p_{sys}(n, t_{n_c})} \end{aligned} \tag{5}$$

The expected execution time for combined redundancy is given by (5), which is a combination of (3) and (4). But now, the minimal expected execution time is achieved for a tuple (n, n_c), indicating the number of used PEs and the number of inserted checkpoints. **Figure 3** shows the expected execution time for exponentially distributed errors with $t := 1000\mu s$, $tf := 1.0\mu s$, $tc := 1.0\mu s$, $\lambda = 0.0001$ (**Fig. 3a**) and $\lambda = 0.0005$ (**Fig. 3b**).

For the sake of clarity, execution times greater than $1500\mu s$ are not plotted in Fig. 3. Here it becomes evi-



(a) $\lambda = 0.0001$ **(b)** $\lambda = 0.0005$

Figure 3 Expected execution time using combined redundancy.

fault rate	spatial redundancy	temporal redundancy	combined redundancy
$\lambda = 0.0001$	**1007µs** $(6, 1)$	1041µs $(2, 10)$	**1007µs** $(6, 1)$
$\lambda = 0.0005$	1099µs $(60, 1)$	1093µs $(2, 23)$	**1022µs** $(4, 4)$
$\lambda = 0.0020$	1103µs $(4, 1)$	1198µs $(2, 40)$	**1061µs** $(4, 11)$

Table 1 Expected execution time using spatial, temporal, and combined redundancy with several fault rates. The degree of redundancy (n, n_c) is also depicted. The minimal runtime is printed bold for each fault rate λ.

dent that a high degree of both spatial and temporal redundancy leads to a significant communication overhead, and therefore also a high expected execution time. But even if a small $E[t_{run}(n, n_c)]$ can be achieved with a pure form of redundancy (regardless of spatial or temporal redundancy), the minimal expected execution time is achieved only with a combination of both redundancy techniques for several examples (e.g. Fig. 3b). The minimal $E[t_{run}(n, n_c)]$ is reached with 1 checkpoint and 6 PEs for $\lambda = 0.0001$ and with 5 checkpoints and 4 PEs for $\lambda = 0.0005$.

4.4 Comparison

The expected execution times depicted in Fig. 1 - 3 were plotted with parameters for which a non-trivial global minimum exists and becomes most apparent. In order to allow a better comparison of the three redundancy techniques, **Table 1** shows the minimal expected execution time with identical parameters for all three techniques. The results were obtained with the parameters $t := 1000$µs, $t_f = 1.0$µs, and $t_c = 1.0$µs. Table 1 illustrates also the degree of spatial and temporal redundancy, for that the minimal expected execution times have been reached.

Obviously, the minimal expected execution time is achieved with combined redundancy for every fault rate. This is due to the fact that n and n_c can be chosen as $n \geq 2$, $n_c = 1$, which corresponds to spatial redundancy (cf. Table 1, $\lambda = 0.0001$), and also as $n = 2$, $n_c \geq 1$, which corresponds to temporal redundancy. For $\lambda = 0.0005$ and $\lambda = 0.002$ the minimal expected execution time is only accomplished with combined redundancy, what confirms the results from Fig. 3. Furthermore, due to the application of combined redundancy, the number of necessary PEs to achieve a certain accuracy can be diminished compared to spatial redundancy. The retrenchment of resources (PEs in this case) saves energy and allows better parallelization. Combined redundancy requires also fewer checkpoints for the same accuracy compared to temporal redundancy, which leads to a lower NoC load. These results encourage the assumption that combined redundancy becomes particular important for increasing fault rates, as we expect them for future semiconductor manufacturing processes.

5 Effects of Resource Constrained Systems

In Section 4, we have shown that a combination of spatial and temporal redundancy can lead to a significant reduction of the expected execution time of a single task on a resource unconstrained system. In this Section we examine how the results from Section 4 change for a resource constrained embedded system. We now also assume that a group of several tasks $T_1, \ldots, T_m$ is running on this system.

In such a system, the degree of spatial redundancy of the individual tasks affects each other. This implies that a high degree of spatial redundancy for task T_i leads to a lower maximum feasible redundancy for all other tasks running at the same time. The separate determination of the minimal expected execution time for every task is therefore no longer sufficient.

The chosen evaluation function for the following simulated systems is the makespan of the investigated task group. The makespan is the maximum expected task completion time with respect to all PEs. That means, we determine a state s for that applies:

$$\min_{s \in S} \left\{ \max_{1 \leq i \leq \#PE} \{t_{PE_i}(s)\} \right\},$$

S denotes the set of states, $\#PE$ indicates the number of available PEs, and $t_{PE_i}(s)$ is the latest termination time of all tasks running on PE PE_i, with the degree of redundancy given by state s. A state for m tasks consist of $2m$ variables $[(n_1, n_{c,1}), \ldots (n_m, n_{c,m})]$, indicating the degree of spatial and temporal redundancy for each task. Obviously, the state space is enormous, even for a small number of tasks. Therefore, we have employed a simulated annealing based algorithm to evaluate the state that achieves the least makespan. The evaluation function considers the makespan as the prior evaluation criterion. To achieve a reasonable degree of redundancy even for the tasks, which are not responsible for the makespan, the evaluation function takes into account all task termination times, weighted in descending order.

The initial state for our algorithm was $[(2,1), \ldots, (2,1)]$, i.e. dual modular redundancy without temporal redundancy for each task. Due to the resource constraint, there is also an upper bound for the maximum degree of spatial redundancy for any task. The following applies: $n_i \leq n_{max} = \#PEs$, $i \in \{1, \ldots, m\}$, whereas $\#PEs$ indicates the number of available PEs. n_i specifies the number of utilized resources, i.e. $n_i = 4$ implies that four different PEs are used. A schedule that maps a task on a resource twice, is therefore an invalid schedule. This restriction was made to distinguish between the fault-related re-execution and the spatial redundancy. In scientific work covering the scheduling of redundant tasks, this restriction is frequently assumed, e.g. in [21]. The allocation to different PEs also has the advantage that a permanent failure affects only a single instance of a task.

Figure 4 depicts three schedules for a system with four PEs and two tasks, a long-running task T_1, and a short-running task T_2. The long-running task T_1 requires triple modular redundancy, whereas the short-running T_2 requires only dual modular redundancy. Even if the schedule in **Fig. 4a** is completed faster than the schedule from **Fig. 4b** and **Fig. 4c**, it is an invalid schedule, as T_2 is scheduled twice on PE_4. The schedule from Fig. 4b is also an invalid schedule, as it schedules both redundant versions of T_2 at different times. Only the schedule from Fig. 4c is valid, since in this schedule all redundant tasks are mapped to different PEs and scheduled simultaneously. The scheduling principle from Fig. 4b executes all versions as soon as possible. This can lead to better utilized PEs and a smaller makespan compared to the schedule principle from Fig. 4c, in which all redundant versions of a tasks are executed at the same time. However, the redundant versions of a tasks have to exchange their states and confirm each other, which is not possible with the principle depicted in Fig. 4b. In the following we have therefore assumed that redundant tasks are scheduled always simultaneously.

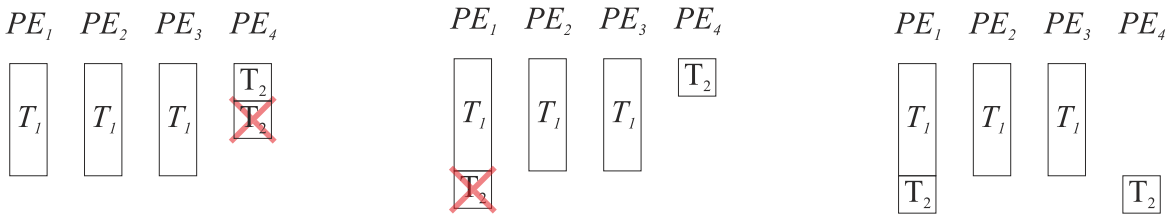
Besides the problem of the optimal degree of redundancy, a task mapping and schedule has to be calculated to determine the makespan. Multiprocessor scheduling is a NP-hard optimization problem (cf. [22], [23]). The problem we consider is denoted as $P||C_{max}$ in literature, where a set of independent tasks is scheduled non-preemptively on identical PEs in order to minimize the makespan. To be able to investigate a significant number of tasks and PEs, we have used the LPT algorithm [23]. The LPT algorithm is list scheduling with tasks scheduled in decreasing order of their processing times. For non-redundant tasks, LPT achieves a performance ration of $\frac{4}{3} - \frac{1}{3 \cdot \#PEs}$ (cf. [24]). Peng [21] has deduced an upper bound for the performance ratio of redundant task scheduling on multi-processors and has also shown that LPT is even optimal in certain cases. However, Peng has premised that redundant tasks are scheduled as soon as possible. Since we assume that redundant tasks are scheduled simultaneously, the upper bounds are not necessarily valid for our results. For a large number of tasks, as we examine them in the following, both the *as soon as possible* and the *simultaneously* scheduling principle have achieved similar results for our examples. But the determination of upper bounds of the performance ratio is still pending.

To investigate the improvements of combined redundancy over pure spatial or temporal redundancy, we have compared the makespan of a group of tasks on a resource constrained system. The considered task group consists of eighteen independent tasks with various failure free execution times:

- $t(T_1) = t(T_2) = t(T_3) = 100$µs,
- $t(T_4) = 200$µs,
- $t(T_5) = t(T_6) = 250$µs,
- $t(T_7) = t(T_8) = t(T_9) = 300$µs,
- $t(T_{10}) = t(T_{11}) = 400$µs,
- $t(T_{12}) = 450$µs,
- $t(T_{13}) = t(T_{14}) = t(T_{15}) = 900$µs,
- $t(T_{16}) = 950$µs,
- $t(T_{17}) = 1000$µs,
- $t(T_{18}) = 1200$µs.

For every redundancy technique, the maximum expected completion time was calculated for three different numbers of available PEs (10, 20, and 50), and exponentially distributed faults with three different fault rates ($\lambda = 0.0001$, $\lambda = 0.0005$, and $\lambda = 0.002$). **Table 2** shows the obtained results.

The results for spatial redundancy and DMR show that



(a) invalid schedule **(b)** invalid schedule **(c)** valid schedule

Figure 4 Three different schedules for four PEs and two tasks.

Number of PEs	fault rate	spatial redundancy	temporal redundancy	combined redundancy	initial state (DMR)
n = 10	$\lambda = 0.0001$	2160.2µs	1925.6µs	**1925.5µs**	2160.2µs
	$\lambda = 0.0005$	3708.4µs	2023.5µs	**2023.1µs**	4440.0µs
	$\lambda = 0.0020$	33714.1µs	**2277.3µs**	**2277.3µs**	146176.0µs
n = 20	$\lambda = 0.0001$	1224.1µs	1249.0µs	**1208.9µs**	1528.3µs
	$\lambda = 0.0005$	1854.0µs	1317.9µs	**1226.5µs**	3993.1µs
	$\lambda = 0.0020$	27630.9µs	1581.8µs	**1274.7µs**	146176.0µs
n = 50	$\lambda = 0.0001$	**1208.5µs**	1249.0µs	**1208.5µs**	1528.3µs
	$\lambda = 0.0005$	1591.0µs	1317.9µs	**1226.5µs**	3993.1µs
	$\lambda = 0.0020$	27630.9µs	1581.8µs	**1273.5µs**	146176.0µs

Table 2 Makespan for a group of tasks and a resource constrained system. The minimal makespan is printed bold for every case.

a single context comparison is not suitable for a high fault rate like $\lambda = 0.0020$. Table 2 shows also that a simple technique like DMR leads to an unnecessary high makespan even for $\lambda = 0.0005$ or smaller. Spatial redundancy achieves a little makespan for a low fault rate and high number of available PEs, but is not suitable for a small number of PEs or a high fault rate. Good results can be obtained with temporal redundancy, however with an increasing number of PEs, the advantage of combined redundancy rises. Like for the previous findings for unconstrained systems, combined redundancy achieves always the best results, as spatial and temporal redundancy are special cases of combined redundancy. But for the majority of our results, combined redundancy even outperforms both, spatial and temporal redundancy. The advantage of combined redundancy becomes particular evident for a high failure rate as $\lambda = 0.002$. A simple redundancy technique like dual modular redundancy achieves an expected completion time of 146176µs, which is more than 120 times of the failure free completion time. Whereas combined redundancy achieves an expected completion time of 1273.5µs, which is only an increase of about 6%. Unfortunately, the states, with which the results form Table 2 have been obtained, can not be shown due to space restrictions. The states for combined redundancy, $n = 50$ PEs, and $\lambda = 0.0001$ as well as $\lambda = 0.002$ are shown in **Table 3** and **Table 4**, respectively. As can be seen in Table 3 and 4, the determined makespan results from the expected execution time of the long-running task T_{18}. For $\lambda = 0.0001$, the expectation for a successful computation of task T_{18} without redundancy is about 89%. Therefore a high degree of spatial redundancy achieves the least expected execution time and also the best makespan in this case. Additional temporal redundancy would only lead to an unnecessary overhead. For $\lambda = 0.002$, the expectation for a successful computation of task T_{18} is only about 0.9%. A high degree of temporal redundancy is therefore necessary to enhance the expectation for a successful computation of a task segment. Due to the high fault rate and the resource restriction, the degree of spatial redundancy of the tasks affects each other. An example is the determined degree for task T_8, which is $(2, 14)$, but optimal for an unconstrained system would be $(3, 3)$.

Task	n	n_c	expected execution time
T_1	2	1	104.1µs
T_2	2	1	104.1µs
T_3	2	1	104.1µs
T_4	2	2	208.2µs
T_5	2	3	260.4µs
T_6	2	3	260.4µs
T_7	2	3	312.2µs
T_8	2	3	312.2µs
T_9	2	3	312.2µs
T_{10}	3	1	404.8µs
T_{11}	3	1	404.8µs
T_{12}	4	1	454.1µs
T_{13}	4	1	906.2µs
T_{14}	4	1	906.2µs
T_{15}	4	1	906.2µs
T_{16}	4	1	956.7µs
T_{17}	6	1	1007.1µs
T_{18}	6	1	1208.5µs

Table 3 State for combined redundancy, $\lambda = 0.0001$, $n = 50$ PEs and the corresponding expected execution time.

Task	n	n_c	expected execution time
T_1	2	5	119.6µs
T_2	2	5	119.6µs
T_3	4	1	106.2µs
T_4	2	9	239.1µs
T_5	2	12	298.9µs
T_6	2	12	298.9µs
T_7	2	14	358.6µs
T_8	2	14	358.6µs
T_9	2	14	358.6µs
T_{10}	4	4	424.8µs
T_{11}	4	4	424.8µs
T_{12}	4	5	477.6µs
T_{13}	4	10	955.1µs
T_{14}	4	10	955.1µs
T_{15}	4	10	955.1µs
T_{16}	4	11	1008.3µs
T_{17}	4	11	1061.2µs
T_{18}	4	13	1273.5µs

Table 4 State for combined redundancy, $\lambda = 0.0020$, $n = 50$ PEs and the corresponding expected execution time.

6 Conclusion

In this paper a model for the calculation of the minimal expected execution time was introduced. We have exhibited that there is an non-trivial degree of redundancy regarding the minimal expected execution time using both, either spatial or temporal redundancy. Certainly, the minimal expected execution time can be achieved only with a combination of both techniques in several cases. Thus, also resource consumption can be reduced, as opposed to a single form of redundancy. We have shown that this also applies for a group of eighteen tasks and a resource constrained system.

Our plans for future work include a further investigation of task allocation and scheduling under utilization of the gained redundancy information. That implies the consideration of the parallelization capability. A high degree of spatial redundancy involves a less expected execution time, but also constrains the maximum degree of parallelization. As parallelization has an impact on the failure free runtime, it has to be considered as another parameter that affects the expected execution time. A further aspect that can be improved is the rating function for a redundancy configuration, which currently considers only the PE restriction. E.g. we intend to employ the NoC utilization, to restrain the number of planed checkpoints.

7 References

[1] S. Borkar, "Thousand core chips - a technology perspecitve," in *Proceedings of the 44th annual Design Automation Conference*, ser. DAC '07. ACM, 2007, pp. 746–749.

[2] ——, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, pp. 10–16, November 2005.

[3] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 7–.

[4] L. Huang and Q. Xu, "Characterizing the lifetime reliability of manycore processors with core-level redundancy," in *ICCAD*, 2010, pp. 680–685.

[5] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, Sep. 2002.

[6] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, vol. 34, pp. 43–99, 1956.

[7] K. Ahlstrom and J. Torin, "Future architecture for flight control systems," *Proceedings of the The 20th Conference on Digital Avionics Systems*, vol. 1, pp. 1–10, 2001.

[8] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen, "Nonstop advanced architecture," in *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, ser. DSN '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 12–21.

[9] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar, "Utilizing dynamically coupled cores to form a resilient chip multiprocessor," *Intl. Symp. on Dependable Systems and Networks (DSN)*, vol. 37, pp. 317 – 326, 2007.

[10] D. L. Sanchez, J. L. Aragon, and J. M. Garcia, "Evaluating dynamic core coupling in a scalable tiled-cmp architecture," in *In Proc. of the 7th Int. Workshop on Duplicating, Deconstructing, and Debunking (WDDD'08)*, 2008.

[11] J. Sloan and R. Kumar, "Towards scalable reliability frameworks for error prone cmps," in *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*, ser. CASES '09, 2009, pp. 261–270.

[12] J. L. Bruno and E. G. C. Jr., "Optimal fault-tolerant computing on multiprocessor systems." *Acta Informatica*, vol. 34, pp. 881–904, 1997.

[13] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," INRIA, Research Report RR-7520, Apr. 2011.

[14] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *PPAM (1)*, 2009, pp. 206–215.

[15] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni, "Using replication for resilience on exascale systems," INRIA, Tech. Rep. RR-7830, Dec. 2011.

[16] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho, "Using replication and checkpointing for reliable task management in computational grids," in *HPCS*, 2010, pp. 125–131.

[17] K. B. Ferreira, J. Stearley, J. H. L. III, R. Oldfield, K. T. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, "Evaluating the viability of process replication reliability for exascale systems," in *SC*, 2011, p. 44.

[18] M. Väyrynen, V. Singh, and E. Larsson, "Fault-tolerant average execution time optimization for general-purpose multiprocessor system-on-chips," in *DATE*, 2009, pp. 484–489.

[19] W. Kuo and M. J. Zuo, "The k-out-of-n system model," in *Optimal Reliability Modeling - Principles and Applications*. John Wiley & Sons, 2003, ch. 7.

[20] C. E. Ebeling, "Reliability of systems," in *An introduction to reliability and maintainability engineering*. Waveland Press, 2005, ch. 5.

[21] D.-T. Peng, "Performance bounds in list scheduling of redundant tasks on multiprocessors," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, jul 1992, pp. 196 –203.

[22] A. Percus, G. Istrate, and C. Moore, *Computational complexity and statistical physics*, ser. Santa Fe Institute Studies on the Sciences of Complexity. Oxford University Press, 2006.

[23] J. Błażewicz, *Scheduling computer and manufacturing processes*. Springer, 2001, ch. 5, pp. 137–196.

[24] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM JOURNAL ON APPLIED MATHEMATICS*, vol. 17, no. 2, pp. 416–429, 1969.