# AN OPTIMIZED FAULT-TOLERANT SCHEDULING ALGORITHM BASED ON GROUPING STRATEGY FOR HETEROGENEOUS MULTI-CORE PROCESSORS

Shigan YU[1,2], Bing XIANG[2], Yuliang BIAN[1], Hui LIU[1,*]

*As the circuit density continues to increase, the possibility of charged element radiation turnover increases, so that the probability of transient failure in the execution of the computer task increases, resulting in unexpected errors in the operation results. Traditional Three Mode Redundancy (TMR) is the main method to solve the transient fault of the processor, which is characterized by low efficiency and high power consumption, this paper proposes a Fault Tolerant scheduling algorithm with High-performance and Low-power consumption for Heterogeneous multi-core processors based on Grouping strategy (FTHLHG) while ensuring system reliability. According to the task attributes, the Directed Acyclic Graph (DAG) task model is established to determine the priority, and the tasks are divided into two groups with/without fault tolerant requirements. For the tasks requiring fault tolerance, this paper proposes the Fault Tolerant Scheduling Algorithm based on the Speculation (FTSAS). For the tasks that do not require fault tolerance, this paper proposes the Competition Scheduling Algorithm (CSA). Simulation experiments show that the average performance of FTHLHG is 16.9% higher than that of the traditional fault tolerant method when executing test cases before injecting errors. When injected 200, 2000 and 6000 errors respectively, FTHLHG's fault tolerance was similar to that of the most advanced methods, but the average performance of the FTHLHG algorithm was improved by 11.7% and the average power consumption was reduced by 21.1%.*

**Keywords:** Fault Tolerant; Grouping; Heterogeneous multicore; Processors; Simulation

## 1. Introduction

In recent years, the rapid development of society has put forward more urgent requirements for high-performance computers, and high-performance computers depend on high-performance processors. In the past, high-performance processors relied on highly integrated transistors on a single chip, which has entered the limits of development. Now, they have turned to single-chip&multi-core processors, and the number of high-integration transistors has increased exponentially [1]. Multi-core chips have become the mainstream of processors, which is another way Moore's law continues. For a long time, homogeneous multi-core has always been the main architecture of processors, but in practice, the homogeneous big-core processor

[1] School of Information Engineering, Fuyang Normal University, Fuyang 236041, Anhui, China
[2] School of Computer and Information Engineering, Fuyang Normal University, Fuyang 236041, Anhui, China
*Corresponding author: Hui Liu, E-mail: liuhuiyeah@yeah.net

leads to low execution efficiency for threads with low priority and complexity, and homogeneous small-core processor will cause the decrease of throughput of the single thread execution [2,3], so either homogeneous big-core or small-core will cause low execution efficiency of program, which is not conducive to optimization of overall throughput and power consumption[4]. Heterogeneous multi-core processors (HMP) is composed of a single core with different performance. According to the characteristics of different cores, it can be dynamically adjusted for different applications to further optimize system performance and reduce power consumption. Therefore, the application range is gradually promoted, for example, ARM's big.LITTLE [5], NVidia's Tegra [6], Intel's QuickIA [7,8] and Huawei's Kirin-9000[9] are all the HMP.

On the other hand, the density of the circuit increases and the gate charge decreases in critical of the transistor, which raises the probability of radiation flip of electrically charged components [10]. BTI (Bias Temperature Instability) [11], HCI (Hot Carrier Injection) [12], and other integrated circuit aging [13] can induce the increase of the processor's transient fault which is the main reason for the 70-80% processor failure [14]. Therefore, both homogeneous and heterogeneous multi-core processors need to be improved reliability and it is important to make the system fault tolerance especially for the key tasks at important nodes, and to ensure that the system can produce correct results even when errors occur in the execution. Compared with the study on the reliability of homogeneous multi-core, there is less research on the reliability of HMP. This paper is to carry out research on fault-tolerant scheduling for HMP. A fault-tolerant system must be redundant, and the system's reliability can be improved by redundant modules.

The traditional method usually uses the TMR method to solve the transient fault of the processor. After all the three modules have completed the task, the majority rule is adopted. When the two modules are consistent, the wrong result of the inconsistent module is shielded, and the wrong module is synchronized to the correct state. After that, the entire TMR system starts to execute the next task from the same state simultaneously. This homogeneous TMR mainly adopts the spatial redundancy for the reliability of processor system. Each task is executed three times on a homogeneous system with three modules, which features low efficiency and high power consumption and are unable to make full use of the characteristic of the diversity of tasks. Because different cores have different properties, HMP have different high efficiency when processing different tasks.

In order to take full advantage of the characteristics of each core, this paper proposes a fault-tolerant scheduling method based on grouping strategy for HMP, which can improve the system reliability and realize high-performance and low-power scheduling at the same time. The contributions of this paper are summarized as follows.

(1) The shortcomings of existing TMR are analyzed in detail, and an optimal scheduling method FTSAS based on speculation mechanism is proposed.

(2) On the basis of studying the different properties of heterogeneous multicore and the existing tasks to be processed, a high performance HMP scheduling method CSA based on competition mechanism is proposed.

(3) According to the different characteristics of the tasks to be processed, a high performance HMP fault-tolerant scheduling method FTHLHG based on grouping strategy is proposed.

(4) The proposed FTHLHG method and other fault-tolerant methods are evaluated, and the experimental results show that FTHLHG can achieve higher performance and lower power consumption while ensuring reliability.

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 introduces the System model. The System working mechanism is presented in Section 4. Section 5 describes the system scheduling implementation. Section 6 evaluates the proposed method. Section 7 concludes the paper and plans the future work.

## 2. Related works

In order to improve the reliability of the system, fault-tolerant technology has been valued in different degrees and forms in the development of computers, which causes academia and industry to invest the important energy in this aspect of research. Traditional fault-tolerant technologies have redundant methods such as DMR(Dual Mode Redundancy) [15], TMR [16], Multi-version technology [17], LOCKSTEP [18], PB(Primary and Backup version) [19], SMT (Simultaneous Multi-threading) [20], checkpoint technology [21] and other Redundancy Check fault tolerance such as ECC (Error Correcting Code) [22], CRC (Cyclic Redundancy Check)[23], PCC(Parity Check Code) [24], HC(Hamming Code) [25]. Among these fault-tolerant methods, TMR has been used much longer and more widely, which has the effect of error detection and fault tolerance, but TMR method can correct only one error at a time. DMR can only detect errors without fault-tolerant ability, and NMR (N>3) can correct multiple errors at a time. Multi-version technology mainly focuses on using multiple versions of a part of software to realize fault tolerance. LOCKSTEP technology processes the same instructions by the redundant hardware at the same time and is realized by the method of redundant hardware to execute tasks repeatedly, which can keep multiple CPUs and memory exactly synchronized and execute the same instructions within the correct clock cycle. PB fault-tolerant technology means a task contains a primary version and a backup version which can work in the active mode, passive mode, and overlapping mode and the system schedules the primary version and the backup version to different processors. The result of the backup version will be adopted as the final

output when there are errors during the execution of the primary version. During the specific execution, tasks need to be divided first and a detection module needs to be added to judge results, which increases the scheduling overhead. SMT fault-tolerant technology is a chip multiprocessor technology that assigns each task to each independent thread, and has a separate pipeline [26,27]. It is a fine-grained fault-tolerant technology to realize fault tolerance by comparing the results of each thread. Checkpoint fault-tolerant technology saves the system in a different state as checkpoints during the different implementation stages. When an error occurs, the system rolls back to the previous checkpoint and re-executes the tasks from the checkpoint that is mainly used in the processing faults of fail-stop [28]. This method requires the setting of system checkpoints at irregular intervals, which causes inefficient in execution. The redundancy check bit is mainly used for error detection and error correction of data.

## 3. Models

### 3.1 System definition

**Definition 1,** Reliability R(t): It refers to the probability that a complete system performs the expected function within a time range [0,t] in a running state. When a system with M modules, there are errors in E(t) modules at the end of the time range of [0, t], the other modules produces the desired results, the system unreliability UR(t) and reliability of R(t) can be calculated in accordance with the following definition(1) and (2).

$$UR(t) = \frac{E(t)}{M} \tag{1}$$

$$R(t) = \frac{(M - E(t))}{M} = 1 - \frac{E(t)}{M} = 1 - UR(t) \tag{2}$$

**Definition 2**, Failure Rate Z(t): It refers to the ratio between the number of the wrong module and normal module within the time range of [0, t]. For a module, Z(t) represents the failure probability within the working time range [0, t]. According to the actual statistics, the relationship between the failure rate Z(t) and time t are shown in Fig. 1.

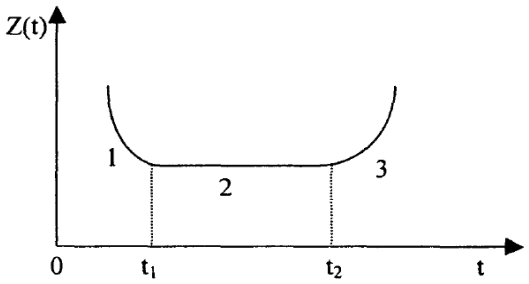


Fig 1. Relationship between module failure efficiency and time.

Fig. 1 can be divided into three phases, among which the first and the third section of the failure rate is higher, the second section of the failure rate is stable and lower. Therefore, the working time of the system should be set in the process of the second stage, which can improve the reliability of the system, reduce uncontrollable factors, and make the system work within our expected range. In the second stage $Z(t)= \lambda$ is usually given. It is generally assumed that the Mean time Between Failures of the system is MTBF. and $\lambda$ is 1/MTBF, which is usually $x*10^{-6}$/hour.

The relationship between the system reliability and the module's failure rate proved to be exponential as equation (3), the reliability of the system can be obtained from formula (4), which provides a basis for further design.

$$\lambda t = -\int_0^{R(t)} \frac{dR(t)}{R(t)} = -\ln[R(t)] \tag{3}$$

$$R(t) = \exp(-\lambda t) \tag{4}$$

**Definition 3:** In heterogeneous multi-core systems, the task can be defined as the DAG model. As a research object, it is a common method for researchers [29]. This paper defined DAG task model as $\Gamma$=(M, V, E , A , T , W).

$M=\{m_0, m_1, m_2, ..., m_i\}$ represents the set of core, $m_i$ represents i-*th* core in the system.

$V=\{v_0, v_1, ..., v_n\}$ represents the set of nodes in $\Gamma$, each node $v_i \in \Gamma$ represents a task. The $pre(v_i)$ represents the immediate predecessor of the current task $v_i$. $Succ(v_i)$ represents the immediate successor to the current task $v_i$. The first task with no $pre(v_i)$ is defined as $v_{entry}$, and the last task with no $succ(v_i)$ is defined as $v_{exit}$.

$E=\{e_{i,j}\}$ represents the set of edges, $e_{i,j} \in \Gamma$ means the WCRT (worst case response time) between $v_i$ and $v_j$ if they are not assigned to the same core and the $e_{i,j}$ is 0 if they are assigned to the same core. All the $e_{i,j}$ are known during the analysis and design phase.

$A=\{a_0, a_1, ..., a_n\}$ indicates whether the task has fault-tolerant properties, $a_i=\{0,1\}$, where $a_i=1$ indicates that the task is resilient task, which does not need fault

tolerance, and $a_i$=0 indicates that the task is sensitive to errors, which requires fault tolerance to improve reliability.

T= {$t_0$, $t_1$, ..., $t_n$} is the set of reliability thresholds, ti represents the reliability threshold of each task.

W={$w_{j,k}$} represents the WCET (worst execution time) of $v_j$ running on $core_k$. Each $v_i \in \Gamma$ has different WCET values on different processors due to heterogeneity of cores. All the $w_{j,k}$ are known during analysis and design phase.

The system task flow based on DAG is shown in Fig. 2. When executing tasks, the system need to sort the DAG task nodes by descending order method [30,31], and the calculation method is shown in formula 5 and 6.

$$\mathrm{sort}(v_{exit}) = c_{exit} \tag{5}$$

$$\mathrm{sort}(v_i) = c_i + \max\{e_{i,j} + sort(v_j)\} \tag{6}$$

Where $c_{exit}$ and $c_i$ represent the time at which the tail tasks node and the *i*-th task node are executed by the core, respectively, and $v_j$ is the successor of task $v_i$.

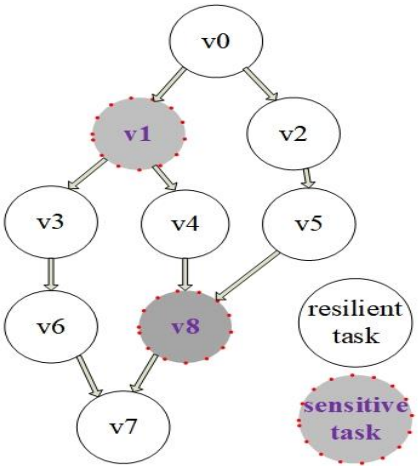


Fig 2. the task flow graph

### 3.2 System Reliability

$\lambda m_i$ represents the failure rate of the core $m_i$, The reliability $R(v_i, m_j)$ that the *i*-th task is executed on the *j*-th processor can be calculated according to formula (7)[32,33]. If $R(v_i, m_j)$ is greater than the current node reliability threshold $t_i$, it indicates that the execution result of the task is satisfactory. The first node is called $V_{entry}$, whose reliability is calculated by the formula (8). The system reliability can be calculated according to formula (9) when the *i*-th node is executed. If $R(\Gamma_i)$ is higher than expectation of reliability, it shows that the system reliability is satisfactory until the current node.

$$R(v_i, m_j) = \exp(-\lambda_{m_j} \cdot w_{i,k}) \tag{7}$$

$$R(v_{entry}, m_i) = \exp(-\lambda v_{entry} \cdot w_{v_{entry},k}) \tag{8}$$

$$R(\Gamma_i) = (\prod_{\substack{\mathrm{Pre}(i)\in\Gamma \\ a_i \neq 0}} R(\mathrm{v}_i, m_{i,k}) \cdot a_i) \cdot R(v_i, m_j) \cdot R(v_{entry}, m_k) \tag{9}$$

## 4. Methods

### 4.1 Traditional TMR mechanism

The fault-tolerant architecture of the traditional TMR is shown in Fig. 3, which consists of three redundant modules, U1, U2, U3, and a Voter. The task Ti is input into three modules, U1, U2 and U3 at the same time. After the three modules have completed the tasks, the results of the execution are sent to the Voter respectively.

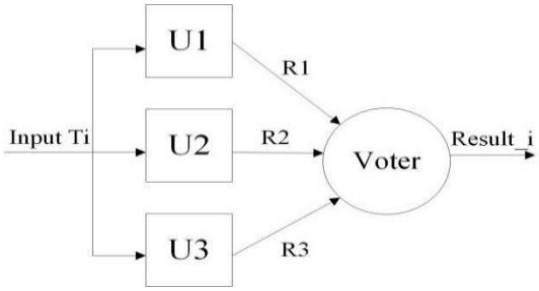


Fig 3. The basic structure of TMR

After the Voter receiving all three results, the majority rule is used to determine the output. When the three results are identical, Voter outputs Ri directly; When both are consistent in three modules, the inconsistent result is shielded and the system outputs the result of consistent modules. In this way, when an error occurs in system execution, the result of the system is free from interference by the wrong module and the fault-tolerant function can be realized.

Since it is a small probability event that the error occurs in two or three modules at the same time, thus, the output of TMR can be defined by logical expression Result_i= R1R2||R2R3||R1R3, which is a majority voting function. If only one error occurs in R1, R2, or R3, the output Result_i is right. The traditional TMR is shown in algorithm 1.

**Algorithm 1:** Traditional TMR Algorithm (TMR)
**Input:** every task Ti
**Output:** result_i of every task Ti
1.If the current task Ti is not a new one, it is finished,
2. Input the current task to each module at the same time;
3. Each module processes the current task Ti separately

4. outputs the execution results R1, R2, and R3 to Voter at the same time.
5. If both or three among R1,R2, and R3 are consistent, the voter output results.
6. If R1, R2 and R3 are different, the current task is given as a new task attribute, goto Step1 to continue.

### 4.2 Speculation working mechanism

Speculation technique has made important contributions to the parallelization of serial processor tasks so as to improve the execution efficiency of serial tasks and has become an important technology in the micro-architecture of multi-core processors. In addition, speculation techniques are also widely used in network communication [34], performance optimization of big data [35], and aviation trajectory judgment [36]. Generally speaking, speculation technology has developed rapidly in recent years, but it is rarely involved in the field of fault tolerance.

When solving the transient fault of the processor, TMR features low efficiency and high power consumption, which is mainly due to the fact that all three cores are always working at the same time. After each core has executed tasks, the voting module selects the correct result to output, and then all three cores start to implement the next stage of the task.

Due to differences in the architecture and properties, the faster core in HMP will wait for the backward core until the completion of the backward core when the system adopts TMR fault tolerance mechanism.

In order to take advantage of the characteristics of HMP and ensure system reliability to meet the requirements, a fault-tolerant scheduling algorithm with high performance and low power consumption based on the speculation mechanism is proposed.

Firstly, according to the DAG model, the program to be executed is divided into an ordered task sequence $T_1, T_2, ..., T_n$ ,whose priority is determined by formulas (5) and (6). At the beginning of the program, the same task $T_i$ ($0 \le i \le n$) was assigned to the three cores, and they started to execute the task $T_i$ at the same time. Since each core has a different execution speed, the result of the fastest core C1 is temporarily saved, denoted as R1 and assigned the next task $T_{i+1}$ to core C1 to continue, then the system saves the result of the second completed core C2 as R2 and assigns the next task $T_{i+1}$ to core C2 to continue. If R1 is equal to R2, the slowest core C3 is immediately terminated, and it is synchronized to R2's state, otherwise, the system waits for the result R3 of the slowest core. If R3 is equal to R1, the value of R1 is submitted, and core C2 is withdrawn and $T_{i+1}$ is executed by core C2 and core C3 simultaneously. If R3 is equal to R2, the value of R2 is submitted, core C1 is withdrawn and the next task $T_{i+1}$ is performed by core C1 and core C3 simultaneously. If R1, R2, and R3 are not equal with each other, then the core C1 is withdrawn so that the three cores re-execute the task $T_i$ simultaneously (this is a

small probability event). If the system still don't get the right result from the three core by the next time, the processor hardware may fail and need to be overhauled until all tasks are completed. The above implementation is shown in algorithm 2.

**Algorithm 2:** FTSAS (Fault-Tolerant Scheduling Algorithm with Speculative method)
**Input:** task flow $T_i$
**Output:** the result of task flow
1.Initialize the reliability of task and reliability threshold of the cores, respectively ;
2.Check the reliability of each core, replace the core if it fails to meet reliability threshold;
3.Set several fixed time $P_j$ to save temporary results for synchronizing;
4.Assign every task Ti to each core and they start to execute the task $T_i$;
5.Record the result of the fastest core as C1 at $P_j$, save the result of C1 as R1, and C1 continues to execute $T_{i+1}$ speculatively without waiting for the backward core.
6.Record result of the second core C2 at $P_j$, save the result of C2 as R2, and C2 continues to execute $T_{i+1}$ speculatively without waiting for the slowest core.
7.If R1==R2, then Interrupt the slowest core C3 immediately and synchronize C3 to the state of C2 and start to execute task $T_{i+1}$ at $P_j$ ;
8. else Record C3's result as R3 when C3 executes $T_i$ to time $P_j$;
9. If R3==R1, then synchronize C2 to C3 and perform the next task $T_{i+1}$ simultaneously.
10. If R3==R2, then synchronize C1 to C3 and they execute the next task $T_{i+1}$ simultaneously.
11. else C1 and C2 will be withdrawn and re-execute the current task $T_i$, and then go to Step5;
12. If all the tasks in the task flow have been completed, then Output the final result;
13. else go to step5.

### 4.3 Competition mechanism

The system architecture of HMP using a competition mechanism [37] is shown in Fig. 4, where three cores are selected for the convenience of introducing the mechanism of TMR in this system. The competition mechanism proposed in this paper for HMP's system is shown in Fig. 5.

Due to the different tasks, the speed of each core will be different at the end of each time $P_j$, the system chooses the result of the fastest core as a standard and synchronizes the backward cores, which improves the disadvantage that the fastest core needs to wait for the backward cores.

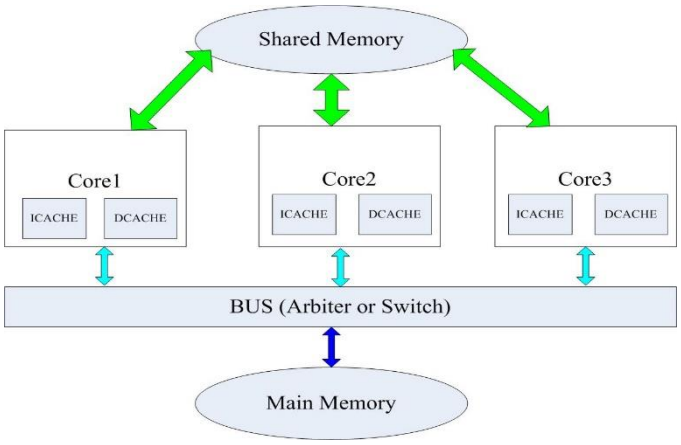


Fig 4. HMP's architecture

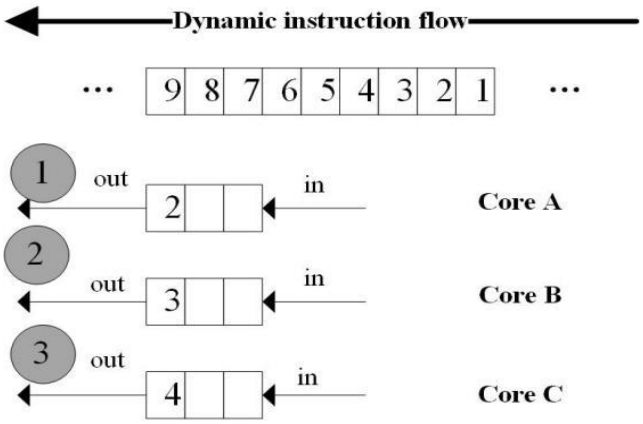


Fig 5. Competition mechanism between cores

When the system executes the current task $T_i$ to the time $P_j$ as it needs to synchronize, the state of each core is shown in Fig. 5. Every core has a different speed and the core C executes the task $T_i$ fastest if the core C matches the current task, and then the core C stores the results in the relevant registers or storage cells, and the laggard cores discard the unfinished tasks and accept the results of core C. Then, the system uses the status of core C as the starting stage of the next task $T_{i+1}$, instead of waiting for the completion of the slower A and B so that three cores can start to work from the same state at the beginning of the next phase, which can take advantage of different cores, and the core matching the task will lead again and then the leader core synchronizes the laggard until the program is completed. Therefore, the overall performance of the system can be improved obviously. In addition, the laggard core accepts the result of the fastest core and terminates the unfinished task directly, the power consumption of the system can be reduced to a certain extent.

The implementation algorithm of the competition mechanism is shown in Algorithm 3.

**Algorithm 3:** CSA(Competition Scheduling Algorithm)
**Input:** task $T_i$
**Output:** result of $T_i$, Result_i.
1. Set synchronization time $P_j$;
2. System begins to execute $T_i$;
3. If the system reaches time $P_j$, save the results of each core individually;
4. Select result of the fastest core as Result_i and terminates the unfinished task;
5. Synchronize laggard cores to the state of fastest core;
6. Else each core continue to execute the current task until time $P_j$, and go to step3;

## 5. Implementation

### 5.1 System hypothesis

It is assumed that no more than one error is allowed to occur in three cores when each task $T_i$ is executed. Each core can execute tasks independently and can communicate with each other. Each core can broadcast the results to other cores via the bus, and can receive the results of other cores at each time period. If there are more than three cores, the system can be set to select three of them to perform tasks.

### 5.2 System scheduling mechanism

In today's computer system applications, there are two types of tasks including flexible tasks with ability of fault tolerance and sensitive tasks without ability of fault tolerance. A flexible task of an application is the one in which some calculations are not executed with 100% accuracy and the final output is still acceptable. Such applications exist in many fields, such as digital signal processing, image, audio and video processing, wireless transmission, web search, data analysis [38], etc. Therefore, no additional fault tolerance measures are required for these applications. However, there are some fault-sensitive control flow tasks. Serious errors may occur if fault-tolerant method is not adopted in the execution. The execution of the entire application will be wrong and even cause system crash. Therefore, a new fault-tolerant scheduling algorithm is proposed in this paper to improve the system execution efficiency while ensuring the system features high reliability and lower power consumption.

First of all, the reliability R(t) of the core detected by formula 4 should be higher than the threshold, or the new core should be replaced. Then, the system executes the task without fault-tolerant requirements by the algorithm CSA. At the end of each fixed detection time $P_j$, the result of the fastest core is used as the output. The system will execute the task again by the Algorithm FTSAS if the result fails to meet the reliability requirements. For sensitive tasks with fault-tolerant requirements, the algorithm FTSAS is used to execute the task directly, which not

only achieves the goal of reliability requirements, but also makes full use of the characteristics of HMP to improve performance and reduce power consumption. Combined with the speculative mechanism and the competition mechanism, the **FTHLHG** ( Fault-Tolerant scheduling algorithm with High performance and Low power consumption for Heterogeneous multi-core processor based on Grouping strategy) is proposed in Fig. 6:

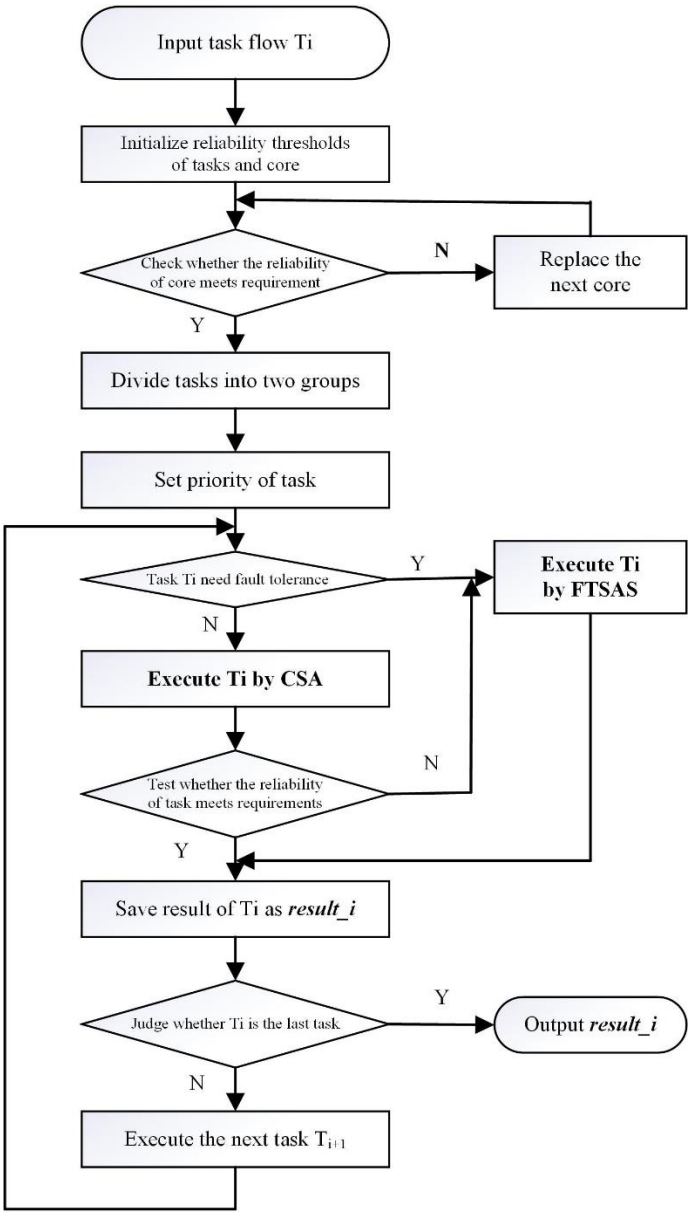


Fig 6. System execution flow

And implementation is shown in algorithm 4.

**Algorithm 4:** FTHLHG (Fault-Tolerant scheduling algorithm with High performance and Low power consumption for Heterogeneous multi-core processor based on Grouping strategy)
**Input:** Task flow $v_i$
**Output:** The result of each task $v_i$
1.Initialize reliability threshold of tasks and each core;
2. Check the reliability of each core, replace the core if it fails to meet reliability threshold;
3. Divide the current task into task flow ($v_0$,$v_1$, ..., $v_n$), including sensitive tasks and flexible tasks;
4. Set DAG-based task flow sequence by formula 5 and 6;
5. Execute task flow $v_i$;
6. If $v_i$ is a sensitive task, system execute $v_i$ by FTSAS, record the result as Result_i;
7. If $v_i$ is a flexible task, system execute $v_i$ by CSA, record the result as Result_i;
8. If Result_i fail to meet reliability threshold according to formulas 9 , re-execute the task $v_i$ by FTSAS and save the result as Result_i;
9. Synchronize the result Result_i to the other two cores as the initial state for next task $T_{i+1}$ and output Result_i;
10. If the task flow does not end, then go to Step5 to continue;
11. else, output the final result;

## 6. Experimental results

### 6.1 Experimental setup

Heterogeneous processor simulation platform is a real simulation of computer with a very close execution effect, which is a common approach to study the properties of processors. Therefore, this paper adopts heterogeneous multi-core simulator platform to execute each test case to realize the verification of the optimization algorithm.

The Simplescalar simulator designed by Intel is open source and an important simulator for high-performance processor architecture[39,40]. It features simulation function including executing drive, explanation execution, assembly line and instructions with out-of-order execution, system compiler, system test, and supports a variety of instruction sets such as PISA, ARM, X86, etc. Therefore, the HMP consisting of PISA, ARM1 and ARM2(same instruction set, different performance configurations) was selected to build the experimental platform based on the Simplescalar simulator, as shown in Fig. 7.

The simulator of HMP is configured in the experimental environment as shown in Table 1[41]. Using System C as a development tool, C++ adds class library, introduces concurrency, timing events and hardware data types, simulates

the core, defines hardware and software components, and models the hardware. The core communicates and realizes synchronization with each other through shared storage units, and SimOutorder is adopted for functional simulation.

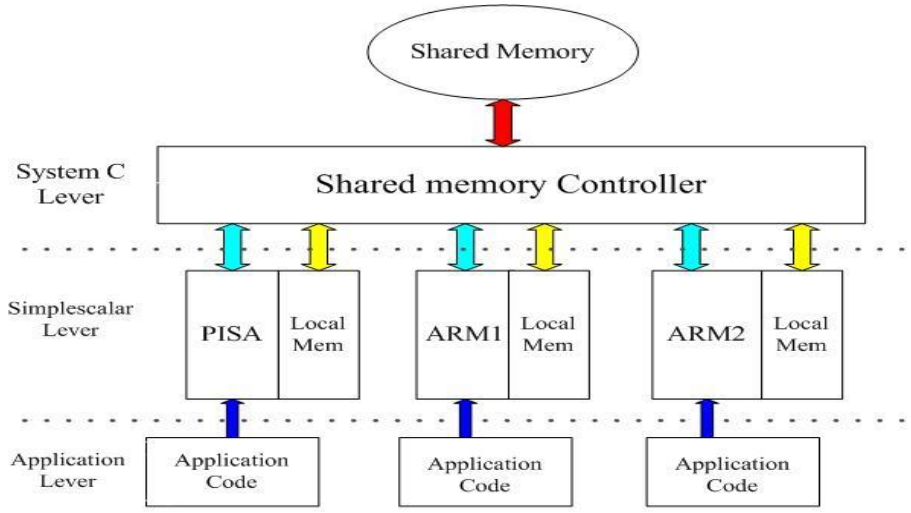


Fig 7. Architecture of HMP

*Table 1.*

**Configuration table of HMP**

| **Core Type** | PISA | ARM1 | ARM2 |
|---|---|---|---|
| **Fetch/Issue/commit** | 4/4/4 | | 4/2/2 |
| **ROB/LSQ Entries** | 128/64 | | 64/32 |
| **Int/Fp units** | 4/4 | | 2/3 |
| **RUU Size** | 16 | | |
| **Pipeline width** | 5 | | |
| **FUs** | 3 int add, 1 int mult, 1 int div, 1 fp add, 1 fp mult , 1 fp div | | |
| **ITLB** | 16-way, 4096 byte page, 4-way LRU 30 cycle miss penalty | | |
| **DTLB** | 32-way, 4096 byte page, 4-way LRU 30 cycle miss penalty | | |
| **Branch Prediction** | Gshare: 9, pht:4096, BTB:512, 2-way group-mapped, Random | | |
| **L1 Icache** | 64KB, 2-way group-mapped, Random | | |
| **L1 Dcache** | 64KB, 2-way group-mapped, Random | | |

Test cases consisting of SPEC2000, matrix multiplication, and sorting algorithms were selected for system testing, in which SPEC2000 was selected as an integer (denoted as SPEC2000_int) to test and take the average value; the scale of matrix multiplication was 64*64(denoted as MM64), and sorting algorithm was

used to sort 600 numbers (denoted as SORT600).

TPFTRM[42], which has the highest scheduling performance among PB fault tolerant methods at present, is adopted. After the execution time of each test case is standardized, the execution time of TMR, DMR and PB methods is compared with the proposed algorithm, and the efficiency of the same test case under different methods is obtained.

As FTSAS method has a fault-tolerant function, and the individual CSA method does not have fault-tolerant ability, the fault-tolerant experiment of FTSAS based on the speculative mechanism, and FTHLHG based on the grouping strategy were completed respectively. By comparing the performance and power consumption of existing fault-tolerant algorithms, the FTHLHG algorithm can be found to have prominent advantages.

### 6.2 Results and performance analysis

Fig. 8 shows the performance difference between the fault-tolerant algorithm of FTHLHG proposed in this paper and the fault-tolerant algorithm of TMR, DMR, PB and FTSAS. When executing the test cases of SPEC2000_int, MM64 and SORT600, the average performance of the FTHLHG algorithm was 16.9% higher than that of TMR, DMR, PB and FTSAS. Compared with other algorithms, FTHLHG yields the highest optimization performance than traditional TMR, especially when SPEC2000_int is executed, the performance of FTHLHG is improved to 27.2%. Because TMR needs to wait until all three modules have completed the task comparison before the next task can be performed, which reduces the performance of TMR algorithm.

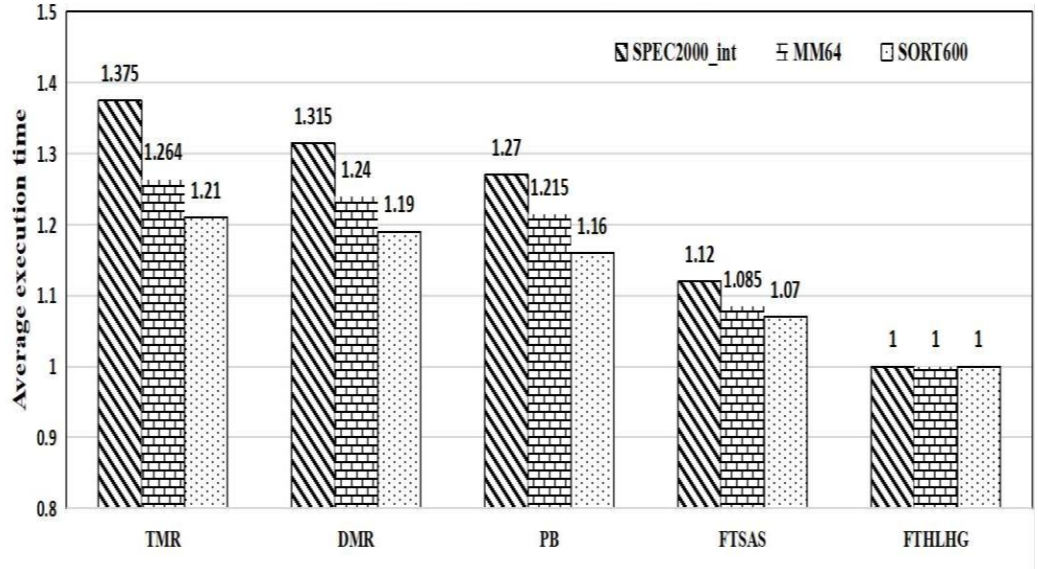


Fig 8. performance comparison between FTHLHG and other algorithms before injecting errors

Compared with other test cases, FTHLHG has a higher performance when executing SPEC2000_int. SPEC2000_int contains a variety of integer tasks, which makes it easier to give play to the characteristics of HMP, highlighting the advantages of grouping competitive calls. The core can execute the task faster when the attribute of a task matches the core's characteristics. Since each task's type is similar, the advantage of the FTHLHG method is diminishing when executing the MM64 and SORT600. This shows that the proposed algorithm FTHLHG is more suitable for the diversity of tasks and can make full use of HMP to improve the system execution efficiency.

In order to truly simulate the environment in which the error occurred in the system, the method of data modification in the storage space with a certain probability of simulating the transient fault of the processor is adopted[43,44]. After 200, 2000, and 6000 errors were injected, SPEC2000_int, MM64 and SORT600 were executed to compare the performance of various fault-tolerant algorithm scheduling, respectively. It can be found that DMR can only detect errors but cannot correct errors by experiments. Therefore, the performance of FTHLHG scheduling is compared with that of TMR, PB and FTSAS scheduling. The reliability of TMR, PB and FTSAS can be obtained after error injection.

As it can be seen from Fig. 9, the reliability of TMR, PB, FTSAS and FTHLHG is higher than the preset goal and these algorithms have similar fault-tolerant ability.

As shown in Fig. 10-12, it is found that the average performance of FTHLHG is improved by 11.7% compared with that of TMR, PB and FTSAS, when SPEC2000_int, MM64 and SORT600 are executed after injecting 200, 2000, and 6000 errors.

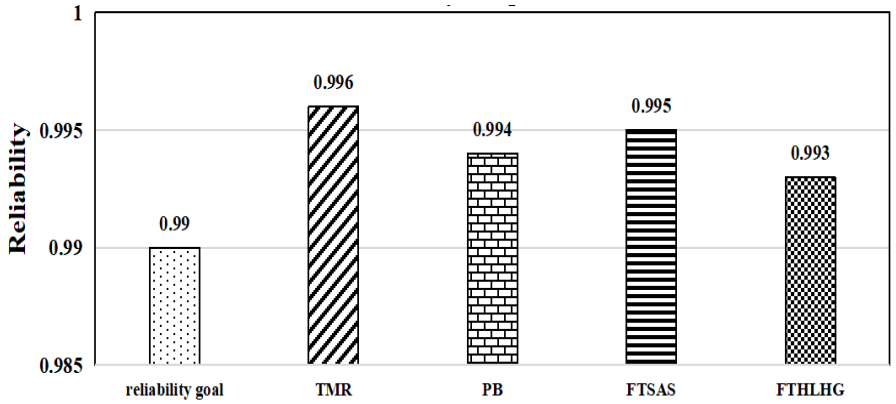


Fig 9. Reliability comparison of different fault-tolerant algorithms

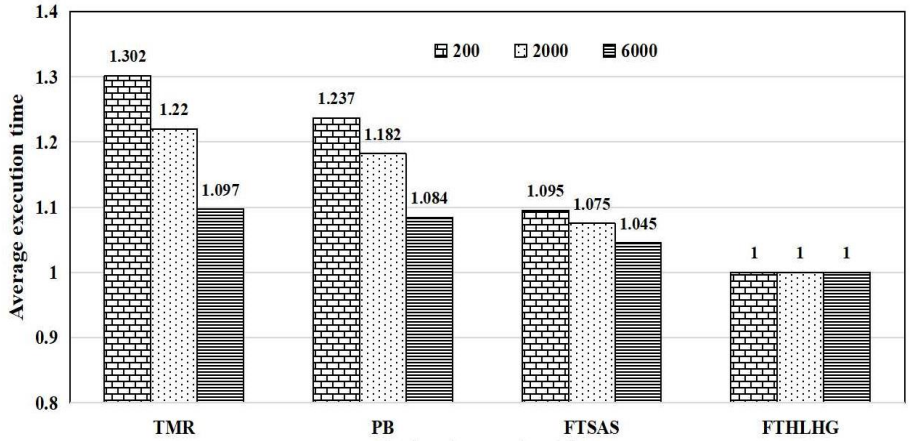


Fig 10. After injecting a different number of errors, performance comparison of fault-tolerant algorithms when executing test case SPEC_2000

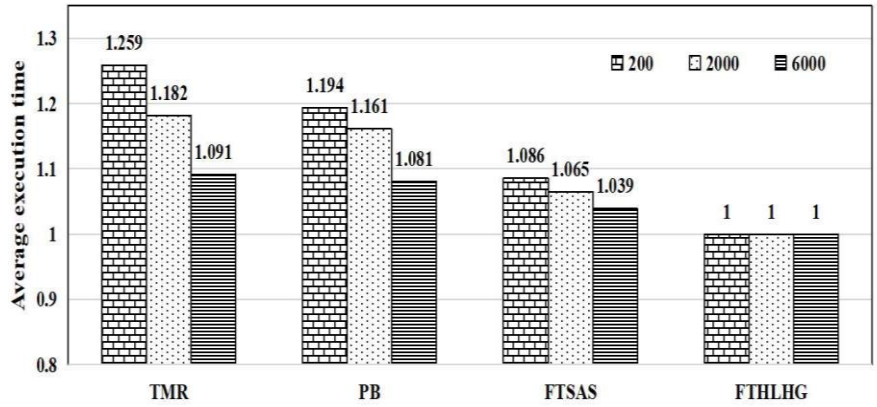


Fig 11. After injecting a different number of errors, performance comparison of fault-tolerant algorithms when executing test case MM64

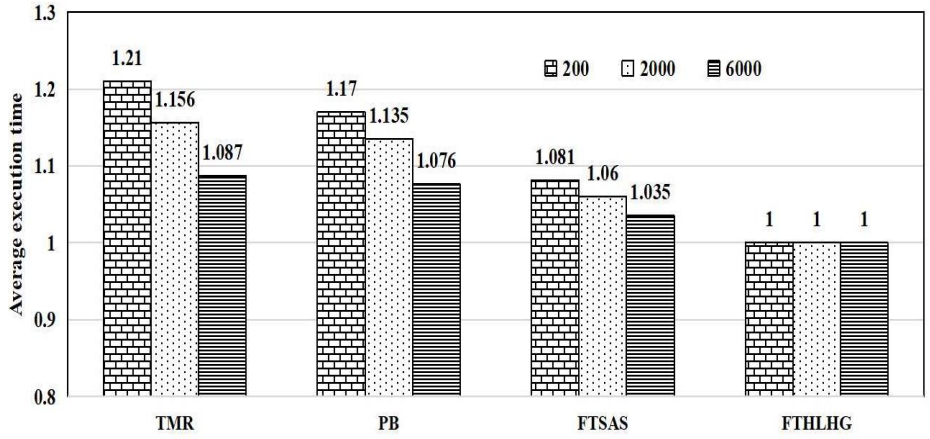


Fig 12. After injecting a different number of errors, performance comparison of fault-tolerant algorithms when executing test case SORT600

It can be found that the advantage of FTHLHG is reducing with the increase of injection error. More errors weakened the role of competition mechanism and the number of tasks requiring speculative scheduling is increasing, especially when 6000 errors are injected, the average performance of FTHLHG is reduced to 6.5%, which is very rare in practical application. If so many errors occur during system execution, there must be a failure in the hardware module, which needs to be checked and replaced with the right core module.

### 6.3 Power consumption analysis

In order to better display the performance advantages of FTHLHG, it is necessary to analyze the power consumption of FTHLHG, TMR, PB and FTSAS in the same system environment. Wattch [45,46] is used to analyze the power consumption on the basis of Simplescalar, and the power consumption of FTHLHG, TMR, PB and FTSAS is calculated by modifying the Simplescalar simulator in combination method of J.Xu [47] and Ahmed [48]. As the number of injection errors increases, fewer tasks need to be competitive scheduling during the execution of FTHLHG, so the corresponding power consumption is increasing and the power consumption advantage of FTHLHG is weakening.

As shown in Fig. 13, when 200 errors are injected, the average power consumption of FTHLHG is 21.1% lower than that of TMR, PB and FTSAS, respectively. When 6000 errors are injected, the average power advantage of FTHLHG decreases to 10.8%.

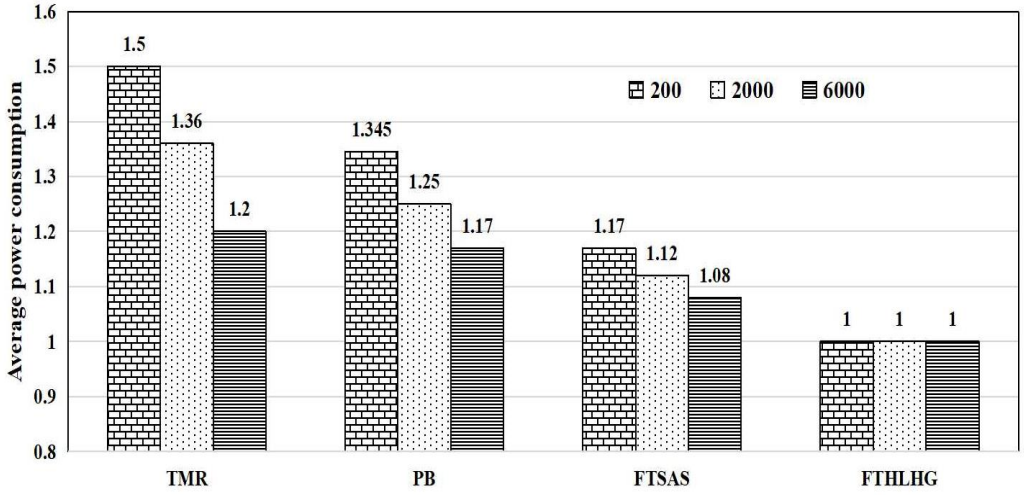


Fig 13. Power consumption relationship of the three algorithms after injecting error

However, 6000 errors occur in the system is a small probability event during system execution, therefore, the FTHLHG algorithm not only improves the performance, but also reduces the total power of the system, which fully shows the performance and power advantage of FTHLHG.

## 7. Conclusion and future work

As the density of the integrated circuit increases and the critical gate charge of the transistor decreases, the probability of radiative flip of the charged element increases, which leads to the increase of the transient fault probability of the computer during the execution of the task, causing unexpected errors. In order to ensure the reliable and efficient execution of the task of the system, the existing fault-tolerant scheduling method needs to be optimized. This paper is to carry out research work under such background. HMP is an important component of high performance computer. HMP shows different performance when executing different tasks, the execution speed of matched tasks is accelerated, and the execution speed of unmatched tasks is slowed down. Therefore, this paper proposes a fault-tolerant scheduling algorithm for HMP based on Grouping strategy. The tasks to be executed are divided into fault-tolerant requirements and fault-tolerant requirements. FTSAS method and CSA method are used for scheduling, respectively. Simulation results show that FTHLHG has better performance and lower power consumption than the existing fault-tolerant algorithms.

The background of this paper is to improve the performance of single-threaded execution and reduce the overall power consumption of the system while ensuring the system's reliability under the von neumann architecture. According to Amdahl theorem, the parallel execution acceleration of the program depends on the performance of the serial application. This approach can be extended to parallel computing and high reliability and high-performance system fault tolerance, such as distributed systems and cloud computing fault tolerance.

In future studies, the characteristics of different core architectures and different tasks will be further analyzed, and tasks will be grouped and assigned to the matched core, so as to further improve system reliability and execution efficiency and reduce power consumption.

## Acknowledgment

The work was supported in part by Quality project of Anhui Province with Grant number 2020zdxsjg260, 2022sx111 and 2021sx117, and in part by Talent research launch start-up project of Fuyang Normal University under grant 2019kyqd0018, and in part by Anhui Province university key research project under grant number 2022AH052820.

## R E F E R E N C E S

[1] *C. McNairy*. "Exascale fault tolerance challenge and approaches". IEEE International Reliability Physics Symposium (IRPS), Burlingame, CA, USA, 2018:3C.4.1-3C.4.10.

[2] *X. Wen, G. Liu, D. Li,et al*, "Federated Scheduling Optimization Scheme for Typed Tasks With Power Constraints in Heterogeneous Multicore Processor Architectures," in IEEE Access, vol. 11, pp. 85728-85746, 2023.

[3] *V.Chau, C.K.K.Fong , S.Liu, et al*. “Minimizing Energy on Homogeneous Processors with Shared Memory”. In: Li, M. (eds) Frontiers in Algorithmics. FAW 2020. Lecture Notes in Computer Science, vol 12340.

[4] *K. Baital, A. Chakrabarti*. “Dynamic Scheduling of Real-Time Tasks in Heterogeneous Multicore Systems”. IEEE Embedded Systems Letters.2018:1-4.

[5] *A. Butko, F. Bruguier, A.bdoulaye, et al*., "Full-System Simulation of big.LITTLE Multicore Architecture for Performance and Energy Exploration," 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), Lyon, France, 2016, pp. 201-208,

[6] *K. -Y. Yeh, H. -J. Cheng, J. Ye, et al*. "Constructing a GPU cluster platform based on multiple NVIDIA Jetson TK1," 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Shenzhen, China, 2016, pp. 917-922,

[7] *N. Chitlur, Ganapati, G. Srinivasa,, et al*. “QuickIA: Exploring heterogeneous architectures on real prototypes”. IEEE International Symposium on High-Performance Computer Architecture(HPCA), New Orleans, LA, USA, 2012,1-8.

[8] *A. O. Munch, N. Nassif, C. L. Molnar, et al.* "2.3 Emerald Rapids: 5th-Generation Intel® Xeon® Scalable Processors," 2024 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 2024, pp. 40-42.

[9] *Jani A.* KIRIN 9000 IS HUAWEI'S FIRST 5NM CHIP[J].Microprocessor report, 2020(12):34.

[10] *A. Naithani; S. Eyerman; L. Eeckhout*. “Optimizing Soft Error Reliability Through Scheduling on Heterogeneous Multicore Processors”. IEEE Transactions on Computers, 2018,67(6):830-846.

[11] *T. -T. Kuo,Y.-C. Chen,Y.-S. Chien, et al.* "A Comprehensive Negative Bias Temperature Instability Model for Gallium-nitride Metal-insulator-semiconductor High Electron Mobility Transistors From 77K to 393K," 2021 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA), Singapore, Singapore, 2021, pp. 1-4,

[12] *P. Dherbecourt, O.Latry, K.Dehais, et al.* “Aging Power Transistors in Operational Conditions”. Embedded Mechatronic System 2 (Second Edition) ,2020,pp23-49.

[13] *P.Chowdhury, U. Guin.* “Estimating Operational Age of an Integrated Circuit”. J Electron Test 37, 25–40 (2021).

[14] *J. Karlsson, P. Liden, P. Dahlgren, et al.* “Using Heavy-ion Radiation to Validate Faulthandling Mechanisms”. IEEE Micro, 1994:14(1): 8-23.

[15] *X. Wang, K. Xu and Y. Xu.* "Design of Airborne Bus Communication Platform based on Isomorphic Dual Redundancy Channels," 2023 7th International Conference on Electrical, Mechanical and Computer Engineering (ICEMCE), Xi'an, China, 2023, pp. 1-6.

[16] *Z. Pan, Y. Hu and S. Zhu*. "Fault Redundancy Active Removal Fault-tolerant Strategy for Triple-redundancy PMSMs," 2023 IEEE 6th Student Conference on Electric Machines and Systems (SCEMS), HuZhou, China, 2023, pp. 1-7.

[17] *Y. Xiong, J. Zhou, L. Su, W.* Wang, *et,al.*, "ECCH: Erasure Coded Consistent Hashing for Distributed Storage Systems," 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, 2021, pp. 177-184.

[18] *K. Liu, Y. Li and L. Ouyang*, "Fast recoverable heterogeneous quad-core lockstep architecture," 2021 International Conference on Advanced Computing and Endogenous Security, Nanjing, China, 2022, pp. 1-6,

[19] *A. Roy, H. Aydin, D.K. Zhu.* “Energy-aware primary/backup scheduling of periodic real-time tasks on heterogeneous multicore systems”,Sustainable Computing: Informatics and Systems,Vol. 29, Part A,2021.

[20] *M. Barbirotta, A. Cheikh, A. Mastrandrea,et,al.* "A Fault Tolerant soft-core obtained from an Interleaved-Multi- Threading RISC- V microprocessor design," 2021 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Athens, Greece, 2021, pp. 1-4.

[21] *D. A. Santos, P.M. Aviles, A.M. Mattos, et al*. "Hybrid Hardening Approach for a Fault-Tolerant RISC-V System-on-Chip," in IEEE Transactions on Nuclear Science, 27 May 2024.

[22] *S. T. Ahmed, S. Hemaram and M. B. Tahoori*."NN-ECC: Embedding Error Correction Codes in Neural Network Weight Memories using Multi-task Learning," 2024 IEEE 42nd VLSI Test Symposium (VTS), Tempe, AZ, USA, 2024, pp. 1-7,

[23] *D. D. Sharma and S. Choudhary*. "Pipelined and Partitionable Forward Error Correction and Cyclic Redundancy Check Circuitry Implementation for PCI Express 6.0 and Compute Express Link 3.0," in IEEE Micro, vol. 44, no. 2, pp. 50-59, March-April 2024.

[24] *A. M. Ahmed, A. Patel and M. Z. A. Khan*, "Parity Check Coded Super-MAC for Reliability Enhancements in Next-Generation Networks," 2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS), Bengaluru, India, 2024, pp. 1116-1121.

[25] *M. R. Alom, M. N. Shakib and M. A. Rahaman*, "Enhanced Hamming Codes: Reducing Redundant Bit for Efficient Error Detection and Correction," 2023 5th International Conference on Sustainable Technologies for Industry 5.0 (STI), Dhaka, Bangladesh, 2023, pp. 1-6.

[26] *Z.S.,Li , X.C.,Xu , W.W.,Hu, et,al*. "Design of the Simultaneous Multi-threading Godson-2 Processor". CHINESE Journal of computers,2009,32(11):2265-2273.

[27] *K. -C. Hsu and H. -W. Tseng*. "Simultaneous and Heterogenous Multithreading,"56th IEEE/ACM International Symposium on Microarchitecture (MICRO), Toronto, ON, Canada, 2023, pp. 137-152.

[28] *Z. Zhang, T. Liu, Y. Shu, et,al.* "Dynamic Adaptive Checkpoint Mechanism for Streaming Applications Based on Reinforcement Learning," 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS), Nanjing, China, 2023, pp. 538-545,

[29] *M. T. Rana, S. Chakraborty and M. V. Salapaka*. "Causal Discovery in Electronic Circuits and Its Application in Fault Diagnosis," 2023 62nd IEEE Conference on Decision and Control (CDC), Singapore, Singapore, 2023, pp. 8223-8228.

[30] *Benoit, M. Hakem and Y. Robert*. "Multi-criteria Scheduling of Precedence Task Graphs on Heterogeneous Platforms," in The Computer Journal, vol. 53, no. 6, pp. 772-785, July 2010

[31] *Li, X. Tang, B. Veeravalli and K. Li*. "Scheduling Precedence Constrained Stochastic Tasks on Heterogeneous Cluster Systems," in IEEE Transactions on Computers, vol. 64, no. 1, pp. 191-204, Jan. 2015.

[32] *H. Lim, T. Kim, D. Lee, et al.* "LARECD: Low area overhead and reliable error correction DMR architecture". 2017 International SoC Design Conference (ISOCC). Seoul, South Korea,2018, pp.27-28.

[33] *Z. Pan , Q. Zheng, Z. Zeng*. "The Signal Integrity Design and Simulation of Triple Modular Redundant (TMR) Computer. IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics", Auto-mation and Mechatronics (RAM), Ningbo, China,2017, pp. 758-776.

[34] *Y. Zhang, L. Zhao, C. Che, et al.* "SpecFL: An Efficient Speculative Federated Learning System for Tree-based Model Training," 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Edinburgh, United Kingdom, 2024, pp. 817-831

[35] *Z. Wang, H. Wang and J. Li.* "A Speculative Parallel Optimization Method for Industrial Big Data Algorithms," 2019 IEEE International Conference on Industrial Internet (ICII), Orlando, FL, USA, 2019, pp. 417-422,

[36] *Q. Li, D. Du, W. Cao, et al.* "A Quality Evaluation Model for Approach in Initial Flight Training Utilizing Flight Training Data," 2023 International Conference on Computer Applications Technology (CCAT), Guiyang, China, 2023, pp. 36-40,

[37] *H. Hashem Najaf-abadi, R. Eric.* "Architectural Contesting". IEEE 15th International Symposium on High Performance Computer Architecture, Raleigh, NC, USA, 2009:189-200.

[38] *J. Yi, Q. Zhang, T. Ye , et al*. "Approx Map: On task allocation and scheduling for resilient applications" , in Proceedings of the 21th Asia South Pacific Design Automation Conference (ASP-DAC '16), Macau, China, Jan.25-28, 2016: 318-323.

[39] *Z. Du, B. Xia, F. Qiao, et al.* "System-Level Evaluation of Video Processing System Using SimpleScalar-Based Multicore Processor Simulator,"Tenth International Symposium on Autonomous Decentralized Systems, 2011, pp. 256-259.

[40] *X. Ren, Y. Tang, T. Tang, et al.* "Sim-spm: A SimpleScalar-Based Simulator for Multi-level SPM Memory Hierarchy Architecture," 2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC), Melbourne, VIC, Australia, 2010, pp. 17-23

[41] *T. Jiang, N. Wu, F. Zhou, et al.* "Design of a High Performance Branch Predictor Based on Global History Considering Hardware Cost," 2021 IEEE 4th International Conference on Electronics Technology (ICET), Chengdu, China, 2021, pp. 422-426.

[42] *J. Wang, J.-L. Sun, X.-Y. Wang, et al.* "Efficient Scheduling Algorithm for Hard Real-Time Tasks in Primary-backup Based Multiprocessor Systems". Chinese Journal of Software, 2009, 20(10):2628–2636.

[43] *S. K. S. Hari, T. Tsai, M. Stephenson, et al.* "SASSIFI: an architecture-level fault injection tool for GPU application resilience evaluation," in 2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2017, Santa Rosa, CA, USA, April 24-25, 2017, 2017, pp. 249–258.

[44] *B.S, Prabakaran,M. Dave,F. Kriebel, et al.* "Architectural-Space Exploration of Heterogeneous Reliability and Checkpointing Modes for Out-of-Order Superscalar Processors". IEEE Access, 2019, PP(99):1-1

[45] *D. H. Albonesi*, "2015 International Symposium on Computer Architecture Influential Paper Award," in IEEE Micro, vol. 36, no. 6, pp. 60-61, Nov.-Dec. 2016.

[46] *M. Ansari, M. Salehi, S. Safari, et al.* "Peak-Power-Aware Primary-Backup Technique for Efficient Fault-Tolerance in Multicore Embedded Systems," in IEEE Access, vol. 8, pp. 142843-142857, 2020.

[47] *J. Xu, Y. Zhu, J. Ni, et al.* "A Simulator for Multicore Processor Micro-architecture Featuring Inter-core Communication", Power and Thermal Behavior. International Conference on Embedded Software and Systems Symposium, 2008:237–242.

[48] *K. Ahmed, S. Tasnim and K. Yoshii.* "Energy-Efficient Heterogeneous Computing of Parallel Applications via Power Capping," 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2020, pp. 1237-1242.