Journal of System Simulation

Volume 30 | Issue 11 Article 20

1-4-2019

High Performance Fault-tolerant Scheduling Method and Simulation for Heterogeneous Multicore

Shigan Yu
1.State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; ;2.School of Computer and Control Engineering, University of Chinese Academy of Sciences, Being 100049, China; ;3. Information Engineering college, Fuyang Normal University, Fuyang 236041, China;

Zhimin Tang
1.State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; ;2.School of Computer and Control Engineering, University of Chinese Academy of Sciences, Being 100049, China; ;

Xiaochun Ye
1.State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; ;2.School of Computer and Control Engineering, University of Chinese Academy of Sciences, Being 100049, China; ;

Zhimin Zhang
1.State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China; ;2.School of Computer and Control Engineering, University of Chinese Academy of Sciences, Being 100049, China; ;

High Performance Fault-tolerant Scheduling Method and Simulation for Heterogeneous Multicore

Abstract

Abstract: To deal with the problem of low performance and high power consumption when solving the transient fault of the processor with three mode redundancy (TMR), *a task execution algorithm for heterogeneous multicore considering fault tolerant (TEAHFT) is proposed. The tasks to be executed are divided into sensitive tasks and fault-tolerant tasks. The sensitive tasks are executed in a TMR mode, and the fault-tolerant tasks are executed in a competitive scheduling mode. The task will be rerun in TMR method if the results of the fault-tolerant tasks do not meet the reliability threshold.* The simulation experimental results show that TEAHFT achieves a 16.4% average efficiency improvement over TMR and PB methods when running test cases. TEAHFT, TMR and PB have similar fault-tolerant results, but TEAHFT's average execution efficiency has increased by 14.1% and the power consumption decreased by 22.1% when 100 errors were injected.

Keywords

TMR, heterogeneous multicore, competitive mechanism, fault tolerance, simulation

第 30 卷第 11 期 系统仿真学报© Vol. 30 No. 11
2018 年 11 月 Journal of System Simulation Nov., 2018

异构多核的一种高性能容错调度方法与仿真

余世干[1,2,3]，唐志敏[1,2]，叶笑春[1,2]，张志敏[1,2]
（1.中国科学院计算技术研究所计算机体系结构国家重点实验室，北京 100190；
2.中国科学院大学计算机控制与工程学院，北京 100049；3.阜阳师范学院信息工程学院，安徽 阜阳 236041）

摘要：针对 TMR（Three mode redundancy）在解决处理器瞬态故障时存在性能低下，功耗较高等问题，*提出了一种考虑容错的面向异构多核的调度方法 TEAHFT，把任务划分为敏感性任务和具有容错性任务，其中敏感性任务以 TMR 方式执行，具有容错性任务以竞争调度方式执行，如果未达到可靠性阈值的任务，则以 TMR 方式重新执行以满足系统可靠性要求。*仿真实验表明，执行测试用例，TEAHFT 比 TMR，PB 方法的性能平均提高了 16.4%，注入 100 个错误时，TEAHFT 与 TMR，PB 具有相近的容错效果，但是 TEAHFT 平均执行效率提升了 14.1%，功耗平均降低了 22.1%。

关键词：三模冗余；异构多核；竞争机制；容错；仿真
中图分类号：TP391.7 文献标识码：A 文章编号：1004-731X (2018) 11-4210-11
DOI: 10.16182/j.issn1004731x.joss.201811020

High Performance Fault-tolerant Scheduling Method and Simulation for Heterogeneous Multicore

Yu Shigan[1,2,3], *Tang Zhimin*[1,2], *Ye Xiaochun*[1,2], *Zhang Zhimin*[1,2]
(1.State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China;
2.School of Computer and Control Engineering, University of Chinese Academy of Sciences, Being 100049, China;
3. Information Engineering college, Fuyang Normal University, Fuyang 236041, China)

Abstract: To deal with the problem of low performance and high power consumption when solving the transient fault of the processor with three mode redundancy (TMR), *a task execution algorithm for heterogeneous multicore considering fault tolerant (TEAHFT) is proposed. The tasks to be executed are divided into sensitive tasks and fault-tolerant tasks. The sensitive tasks are executed in a TMR mode, and the fault-tolerant tasks are executed in a competitive scheduling mode. The task will be rerun in TMR method if the results of the fault-tolerant tasks do not meet the reliability threshold.* The simulation experimental results show that TEAHFT achieves a 16.4% average efficiency improvement over TMR and PB methods when running test cases. TEAHFT, TMR and PB have similar fault-tolerant results, but TEAHFT's average execution efficiency has increased by 14.1% and the power consumption decreased by 22.1% when 100 errors were injected.

Keywords: TMR; heterogeneous multicore; competitive mechanism; fault tolerance; simulation

引言

云计算，大数据等现代新技术的日新月异，各类科学与工程应用、数据中心以及互联网的发展对高性能处理器的迫切需求，多核处理器已经成为目前市场的主流。同构多核具有结构设计简单，但是面对各类不同特点的负载应用时，同构小核会导致单线程执行的吞吐量的减少，同构大核会使低优先级低复杂度线程执行效率的降低[1]，所以无论同构大核还是同构小核都会导致程序执行效率低下的

收稿日期：2018-05-30 修回日期：2018-07-30；
基金项目：国家重点研发计划(2018YFB1003500)，国家自然科学基金(61332009,61521092)，安徽省高校自然科学重点项目(KJ2017A837,KJ2018A0669)；
作者简介：余世干(1982-)，男，安徽定远，博士生，副教授，研究方向为计算机体系结构与容错，并行处理。

缺点，于是异构多核应运而生，异构多核是由不同类型和不同特点的单核构成，这在面对多种类型的负载和多种特点的应用时，无论系统吞吐量，程序执行效率，还是系统性能都会带来提升，因此在目前得到了广泛应用。

由于市场的需求和技术的进步，处理器的性能得到了快速发展，芯片集成度越来越高。频率的提高，功耗密度增大，处理器核温度快速提升，使得微处理器的故障发生率不断升高，微处理器可靠性呈降低趋势[2]。单粒子翻转 SEU(Single Event Upset)影响增大，电路技术的集成度提高使得处理器系统的瞬态故障率 SER(Soft Error Rate)急剧增加[3]。研究表明，处理器的 70%~80%失效都是由于瞬态故障引起的[4]，随着处理器的集成度越来越高，这一趋势越来越明显[5]，因此，微处理器系统可靠性，瞬态故障的容错问题引起了人们的极大关注。

从第一台计算机 ENIAC 诞生起，设计者一直追求设计高可靠性系统，容错技术就一直用来提高系统的可靠性[6]。传统的方法是采用 TMR 解决处理器的瞬态故障，利用三种相同的处理系统来完成对系统错误的识别与处理，即采用三种同构系统完成系统的容错。当三种同构处理系统执行某一应用程序完成时，通过仲裁器判断，采用多数一致原则。当有两者一致时，把不一致的第三个模块执行错误结果屏蔽，同时把第三个模块同步到另外两个模块一致的状态，然后三个模块又从一致状态执行下一任务。这种同构三模冗余执行过程主要采用以空间冗余换取处理器系统可靠性。对于任何类型的任务都是在三个同构系统模块上执行三次，效率低下，功耗较高，不能充分利用任务多样性的特点，这与提倡高性能计算的当代社会的要求格格不入。考虑到异构系统在面对不同类型任务能表现出不同的高效执行效率，因此，本文提出了面向异构系统的一种基于竞争机制的高性能低功耗容错调度方法，在实现系统容错的同时又提高系统执行任务的效率，通过仿真实验说明本文的方法能够实现高性能低功耗容错调度。

1 相关工作

瞬态故障是引起处理器失效的主要错误，通常采用容错的方法来解决瞬态故障，提高处理器的可靠性。冗余机制广泛存在于容错系统中，以便检测并使系统正常运行不受错误影响。冗余的方法主要分为信息冗余，时间冗余，空间冗余[7]。信息冗余主要通过正常数据基础上增加冗余校验码实现数据的查错和纠错，例如 ECC(Error Checking and Correcting)校验[8-9]，CRC(Cyclic Redundancy Check)校验等方式[10-11]，主要用来纠正存储器中数据的错误。Yupeng Hu 等[12]提出一种弹性的 ECC 校验码，不仅满足内存中不同页面的纠错要求，而且获得了良好的可靠性性能权衡。时间冗余主要通过定时设置系统正确时刻的检查点，当发生错误时卷回恢复使系统恢复到正确状态，以牺牲时间换取系统的正确结果[13-14]。Tuo Li 等[15]提出细粒度的检查点容错方法。Nosayba 等[16]提出了一种面向高性能计算的改进的检查点优化算法，实现进一步能耗优化和性能提高。空间冗余主要是以 TMR，双模冗余 DMR (Dual Mode Redundant)，或 LOCKSTEP 技术为代表的应用[17-18]，提高系统的可靠性。空间冗余在较大的方面如空间卫星系统，数据计算，实时系统，在细粒度方面如指令级，线程级等方面的容错发挥了重要的作用。Luca Sterpone 等[19]在面向空间与电子计算领域，针对 SRAM 和 FPGA 的固有特性，开发一种基于 TMR 的方法，用于解决 SEU 问题，高效的评估对故障的干扰。Jyothish Soman 等[20]提出了指令级冗余实现对在指令执行阶段的查错与纠错，而不需要修改硬件配置，提高系统执行效率，但是增加了指令编译与结构设计的复杂度。Wangyuan Zhang 和 Jie Yin 等[21-22]提出同时多线程的冗余方式实现容错，并对实现过程做出优化，提高系统可靠性。

在 DMR 中，通常采用两台机器同时工作，对两台机器的执行结果比较是否一致来判断是否出错，这只能检查出错误，不能纠正错误，纠正错误只能交给其他功能模块，这往往限制 DMR 容错的

第 30 卷第 11 期 系统仿真学报 Vol. 30 No. 11
2018 年 11 月 Journal of System Simulation Nov., 2018

使用范围。在实时系统中，对 DMR 进行修改，在假设两个模块两次执行任务不能同时发生错误前提下，设计出主副版本 PB(Primary&Backup)冗余机制，选择其中一个为主模块，另一个为副模块，主副模块执行同一任务，副模块工作方式有主动方式，被动方式，重叠方式。这种方式在执行过程中需要增加额外的计算过程，而且还需要增加对结果的检测模块。Jing Weipeng 等[23]介绍了在实时多处理器系统中，为了保证每个任务在截止期之前完成，当主版本任务发生错误时，选用副版本的执行结果实现系统容错。在 TMR 里，由于三个模块同时工作，采用多数原则，不仅能够发现并纠正错误，且易于实现，大大的提高了系统的可靠性，这种特点促进了其应用的推广。J.F.Wakerly 等[24]在 20 世纪 70 年代开始提出 TMR 机制及实现过程来解决系统中瞬态错误。贾佳[25]提出一种 Rollback TMR 处理瞬态故障的方法。经过几十年的发展，现如今已广泛应用于对系统可靠性要求较高的领域[26-28]。但是传统的 TMR 的方法执行任务时，存在效率低下，功耗高的问题。

2 传统的 TMR 容错机制

传统 TMR 的容错结构与逻辑结构如图 1 所示，采用三个相同的模块执行同样的功能，表决器根据多数一致原则，选择正确性结果输出，如果其中一个模块运算过程出现错误，另外两个模块的一致结果则将屏蔽错误模块的数据，最终输出正确的结果。由于在同一个时刻或者周期内，两个模块同时出现错误是小概率事件，所以 TMR 假设在每一次表决器裁决结果时，至多只有一个模块发生错误，本文的方法在实现时也是作出如此合理性假设前提。TMR 的输出可用逻辑表达式 $X=X_1\ X_2||\ X_2\ X_3||X_1X_3$ 来定义，这是一个多数表决函数，对于 X_1，X_2，X_3 中仅发生一个错误时，则输出 X 为正确结果，因此 TMR 对任何一个模块出错都有容错能力。传统 TMR 的实现过程如算法 1 所示。

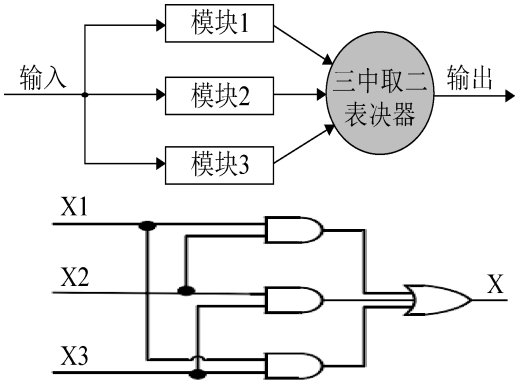


图 1 三模冗余结构与逻辑图
Fig. 1 Structure and logic of TMR

算法 1 传统三模冗余执行算法 TMRA(TMR Algorithm)

输入：3 个模块的输入端同时输入同一任务 T_i

输出：系统输出正确结果 X

Step 1: 把每个任务 T_i 同时传送至 3 个执行模块；

Step 2: 每个模块同时执行 T_i；

Step 3: 每个模块输出结果至表决器；

Step 4: 如果表决器检测出有两者一致或三者均一致，则通过，则输出 X_i 结果 Result_i；

Step 5: 如果未通过，跳转至 Step1 重新执行。

3 系统模型

3.1 系统定义

定义 1 可靠性 $R(t)$: 指一个完整的系统在一个运行状态下，在一个时间范围[0,t]内，完成期望功能的概率。当一个系统具有 M 个模块，在时间范围[0,t]结束后，有 $E(t)$个模块执行发生错误，其余模块产生符合期望的结果，则这个系统的不可靠性 $UR(t)$和可靠性 $R(t)$可以按照定义如下。

$$UR(t)=E(t)/M; \quad (1)$$

$$R(t)=(M-E(t))/M=1-E(T)/M=1-UR(t); \quad (2)$$

定义 2 失效率 $Z(t)$: 表示在某个时间范围[0,t]内，不能正常工作的模块与正常工作的模块数目之间的比值，对一个模块来说，$Z(t)$表示在时间范围[0, t]内不能正常工作的概率。根据实际统计可得失效率 $Z(t)$与时间 t 的关系如图 2 所示。

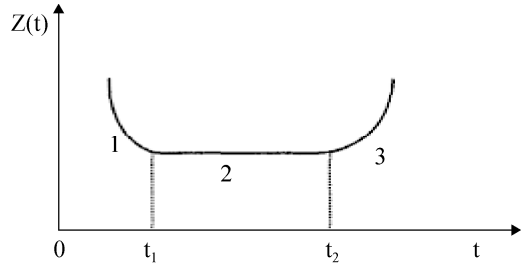


图 2 模块失效率与时间的对应关系
Fig. 2 Relationship between module failure rate and time

在图 2 中，分为三段，其中第一段与第三段失效率较高而且不稳定，第二段失效率稳定且较低，通常使用的模块应该避开第一段与第三段这两个阶段，所以本文在使用前，应该先对模块进行测试，使其工作状态处于图 2 中的第二段过程，否则更换模块，重新检测，这能够提高系统的可靠性，使得系统工作在预期范围内，在这第二阶段时期元件的失效率近似为较低的一个稳定常数，故令 $Z(t)=\lambda$，通常假设系统的平均发生故障时间为 MTBF (Mean Time Between Failures)，则 λ= 1/MTBF，通常 λ 数量级为 10^{-6} 次/小时。

根据失效率的定义有:

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

由此可得：

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

通过推导公式表明系统可靠性与模块失效率成指数关系，可以据此来判断系统的可靠性，从而能够为进一步设计提供基础。

3.2 系统竞争工作机制

异构多核采用竞争机制[29]的系统结构如图 3 所示，这里主要是为介绍本系统 TMR 的机制的方便，选用三个核。在实际的芯片中应该多于三个核，设置其中三个核处于这种工作模式。每一个核都能够单独执行任务，可以相互通信，在设定的每个时间段，每个核通过总线都可以把执行结果广播给其他核，也可以接收其他核的广播结果。

在本文提出的面向异构多核系统的竞争的容错机制中，执行状态如图 4 所示，由于执行任务的不同，在每一个 T 时刻结束时，每个核的执行任务的速度因而会有所不同，每隔时间 T，同步一次，选取执行任务较快的节点作为同步标准，而不需要等三个节点都执行完才同步，这优化了以往执行快的核等待落后核的效率低下的缺点。

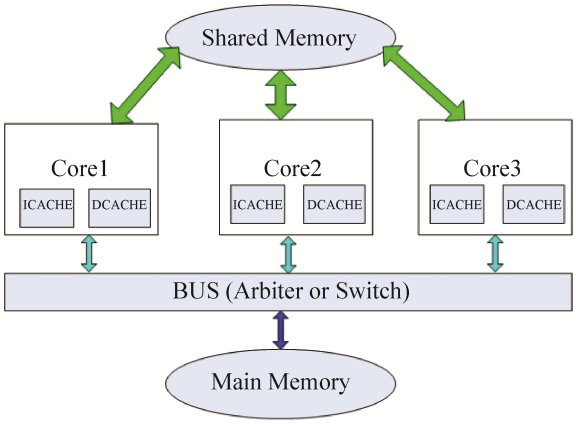


图 3 本文的异构多核结构图
Fig. 3 Heterogeneous multicore structure

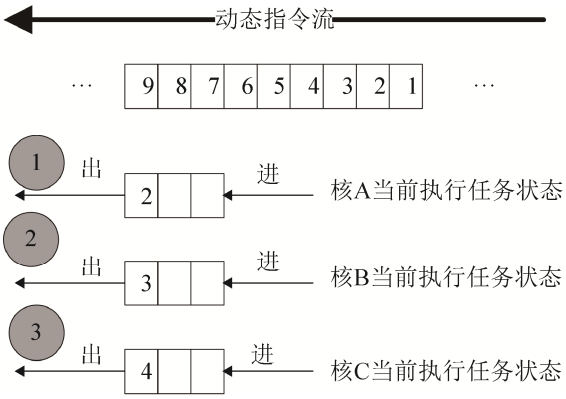


图 4 各核间竞争思想状态图
Fig. 4 Competition mechanism between cores

采用的设计思路如下：当前系统执行到 T 时刻时，到了同步的时刻 T，此时各个核执行的状态如图 4 所示，由于每个核的当前执行任务的速度不一致，假如核 C 的内部结构与当前执行的任务属性最匹配，所以执行最快，内核 C 会把执行结果保存到相关寄存器或者存储单元中，其他落后的两个核则抛弃尚未执行完的任务，而接收 C 核的执行结果，所以在当前时刻采用 C 核的任务作为下一次任务开始的起点，而不再等待较慢的 A，B 核执行完之后再开始，这样当任务进入到下一阶段

https://dc-china-simulation.researchcommons.org/journal/vol30/iss11/20
DOI: 10.16182/j.issn1004731x.joss.201811020

第 30 卷第 11 期 系统仿真学报 Vol. 30 No. 11
2018 年 11 月 Journal of System Simulation Nov., 2018

时，三个内核又从相同状态开始执行，能够充分发挥不同的内核的特点，与任务相匹配的内核又会再次领先，然后再执行同步，直到所有任务执行完成，因此系统执行任务的性能可以得到明显的提升。

在这种方案的执行过程中，设置的同步时间 T 需要根据实际来确定，如果过大，造成与当前任务较为匹配的内核会遥遥领先，从而失去竞争机制的作用，如果过小，会产生同步次数过多，效率低下。与当前任务不匹配的内核，由于接收了较快的核执行结果，而直接终止当前尚未执行完成的任务，这在一定程度上也降低了系统的功耗。由于内核间传递数据的延迟对性能影响较大，因此各个核之间结构的设计需要尽可能减少总线间传递的延迟，这属于细粒度结构设计的范围，不在本文讨论的内容中，因此不做详述。系统竞争机制的执行过程如算法 2 所示。

算法 2 竞争机制的算法 CA(Competitive algorithm)

输入：系统 3 个模块输入动态任务流中的任务 Ti

输出：系统输出结果 Result_i

Step 1: 设置预期定时检测同步时间 T；

Step 2: 如果执行时间到达预期值 T：

Step 2.1: 分别保存每个内核的执行结果；

Step 2.2: 比较每个核的执行结果，保留当前最快的执行结果；

Step 2.3: 把最快的执行结果同步到其他核，作为下一阶段执行的初始阶段，并输出当前结果 Result_i；

Step 3: 如果执行时间未达到预期值 T，则每个核继续执行当前任务直到达到预期值 T；

Step 4: 跳转到 Step 2 继续执行。

4 系统实现过程

在大数据时代，各个领域大量的应用程序具有一定的容错性。应用程序的容错性是指，即使该应用程序中的某些计算不是以 100%的准确率执行的，最后的输出仍然在可接受的范围内。这样的应用程序存在于许多领域中，如数字信号处理，图像、音频和视频处理，无线传输，网页搜索，数据分析等[2]。在一个应用程序中，由于不同数据和控制流的属性，不同的任务本身具有不同程度的容错能力，但是也存在一些对错误敏感的控制流任务，当被执行时，如果不采用容错措施提高可靠性，系统可能会出现严重错误，一旦出错，整个应用程序的执行都会发生错误，甚至导致系统崩溃，因此本文提出了一个可执行的方案，在任务容错与系统性能之间做一个平衡，在保证系统具有容错功能的基础上，实现高效的执行应用程序时，使得系统具有更高的性能，更低的功耗。

定义 3 系统所要解决的计算任务可以定义为有向无环图 DAG，可以表示为 DAG=(V，E，A，T)，其中 V={v_0，v_1，…，v_n}表示子任务的集合，E={e_{ij}}表示边的集合，A={a_0,a_1,…，a_n}表示任务是否具有容错属性，a_i={0,1}，其中 a_i=1 表示任务具有容错属性(resilient task)，a_i=0 表示任务对错误敏感(sensitive task)，不具备容错属性。T={t_0，t_1，…，t_n}，t_i 表示每个任务执行的可靠性阈值 *Reliability_threshold*，作为任务执行是否满足的依据。具体计算任务流图如图 5 所示。

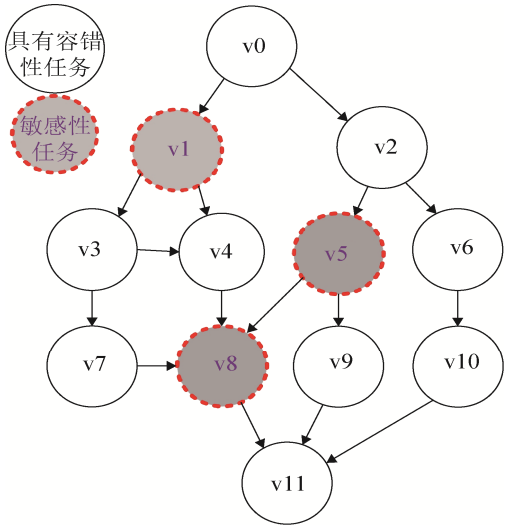


图 5 系统任务流图
Fig. 5 System task flow

首先按公式(4)中检测系统节点的可靠性 R(t)，

如果高于预定标准，然后对于自身具有容错性任务来说，采用竞争性机制来执行，在每一个固定检测时间 T 结束时，以最快的执行结果作为输出，检查执行结果是否达到可靠性阈值，如果没有达到既定要求，采用 TMR 的方案再次执行这个任务；对于自身不具有容错功能的控制流的关键性任务来说，直接采用 TMR 机制来执行，这样既达到了容错目的，同时也充分利用异构多核的特点，提高了计算任务的执行效率。具体实现过程如算法 3 所示。

算法 3 考虑容错的面向异构多核的任务执行算法 TEAHFT (Task Execution Algorithm for Heterogeneous multicore Considering Fault Tolerant)

输入：任务流 V

输出：系统每一个任务 V_i 的执行结果

Step 1：初始化任务可靠性阈值 t_i，各内核的可靠度；

Step 2：按照公式 4 检测每个内核节点的可靠度，如果达不到预期值，则说明内核模块故障多，更换内核模块再重新检测；

Step 3：把当前任务按照 DAG 方法划分为任务流(v_0,v_1,…，v_n)，包含不具有容错的敏感任务和具有容错功能的弹性任务；

Step 4：执行任务流 v_i；

Step 5：如果 v_i 为敏感性任务，则执行三模冗余调度算法 TMRA(v_i)；

Step 6：如果 v_i 为弹性任务，则执行竞争机制算法 CA(v_i)；

Step 6.1：如果执行结果未达到可靠性阈值，则对当前任务 vi 以 TMRA(vi)方式重新执行；

Step 6.2：如果执行结果达到可靠性要求，则保存当前任务 vi 执行结果 result_i；

Step 7：同步结果 result_i 到另外两个内核，作为下一任务的工作初始状态；

Step 8：如果任务流没有执行结束，则跳转到 Step 4 继续执行；否则结束，输出结果 result_i。

5 系统仿真与分析

5.1 仿真实验平台与方法

参考文献[30]方法，选用并修改了 simplescalar 模拟器[31]作为仿真实验平台。Simplescalar 模拟器是由 Intel 公司计算机研究室于 1996 年设计实现，源代码公开的体系结构模拟器[32-33]。在功能级上实现执行驱动、解释执行，在行为级上实现流水线模拟，提供了超标量处理器结构中指令乱序执行的模拟，还提供了包括编译器，Benchmark，测试工具，流水线跟踪器等。目前支持 ARM，X86，PISA，Alpha 等指令集，由于 simplescalar 的优良特性，现已成为研究计算机体系结构的重要工具。

选用 PISA，ARM1 和 ARM2(这里 ARM1 与 ARM2 采用相同的指令集，不同性能配置)组成异构平台，仿真模拟器实现总体框架如图 6 所示。采用系统级硬件描述语言 SystemC 作为开发工具，向 C++添加类库，引入并发、定时事件和硬件数据类型，仿真内核，同时定义软硬件成分，可以对硬件进行建模描述，内核间通过共享存储单元进行通信并实现同步功能，采用 SimOutorder 进行功能仿真，选用 SPEC2000，MediaBench 和排序算法作为测试用例。其中对于 SPEC2000 的整型应用测试，取平均值作为比较对象；对于 MediaBench 应用，选用自适应差分脉冲编码调制 G.721 为测试对象；对于排序算法，设计的规模为 500 个数字。选用传统 TMR，DMR 和 PB 容错算法作为性能对比分析。

我们把各个测试用例执行时间进行标准化处理，以本文提出的算法执行完成的时间作为基准，在 TMR，DMR 和 PB 方法下执行的时间分别与基准进行比较，分别计算相对执行时间，从而得出同一种测试用例在不同方法中执行的效率。

DOI: 10.16182/j.issn1004731x.joss.201811020

第 30 卷第 11 期 系统仿真学报 Vol. 30 No. 11
2018 年 11 月 Journal of System Simulation Nov., 2018

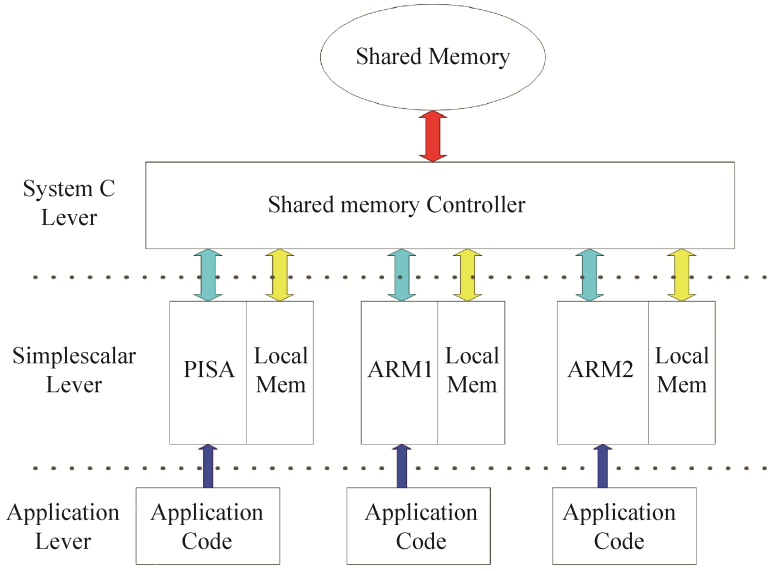


图 6 异构多核仿真模拟结构图

Fig. 6 Structure of heterogeneous multicore

5.2 仿真结果与分析

5.2.1 执行性能结果与分析

图 7 显示了对于 SPEC2000 整型应用，G.721 应用以及规模为 500 的排序应用分别按照 TMR，DMR，PB 以及本文提出的 TEAHFT 方法执行时的性能差异，在这里我们执行 SPEC2000 整型应用时，选取平均值，PB 采用目前在同构系统中容错调度方法性能最高的 TPFTRM[34]执行本文中应用程序测试用例。

从图 7 中可以看到，在执行 SPEC2000 整型应用时，TEAHFT 方法比 TMR，DMR，PB 方法性能分别提高了 22.5%，18.0%，15.3%，平均提高了 18.6%；在执行 G.721 应用时，TEAHFT 方法比 TMR，DMR，PB 方法性能分别提高了 20%，16%，13.8%，平均提高了 16.6%；在执行规模为 500 的排序时，TEAHFT 方法比 TMR, DMR, PB 方法性能分别提高了 15.3%，13.0%，11.5%，平均提高了 13.3%。

为了说明在发生错误时，本文提出的方法的优越性，分别在执行 SPEC2000 整型应用，G.721 与规模为 500 的排序应用时，以一定的概率随机对存储空间中数据修改，以便模拟瞬态故障，在注入不同个数的错误后，分别执行每一种应用，发现 DMR 不具有容错功能，因为在 DMR 方法中，如果不增加结果判断模块，DMR 只能发现错误而不具有容错功能。因此只对比 TMR，PB 与 TEAHFT 方法，这三种方法均具有相近的容错功能，但是不同方法的执行效率有所不同。

通过图 8，可发现，在执行 SPEC2000 整型应用分别注入 100，1000，5000 个错误时，TEAHFT 比 TMR 性能分别提高了 17.4%，13%，5.6%；TEAHFT 比 PB 分别提高了 15.3，12.2%，4.8%；在执行 G.721 应用注入 100，1 000，5 000 个错误，TEAHFT 比 TMR 方法性能分别提高了 15.2%，10.7%，6.5%；TEAHFT 比 PB 方法性能分别提高了 13.8%，10%，5.7%；在执行规模为 500 的排序注入 100，1 000，5 000 个错误时，TEAHFT 比 TMR 方法性能分别提高了 12.3%，9.9%，6.5%；TEAHFT 比 PB 方法性能分别提高了 10.7%，8.3%，4.8%。

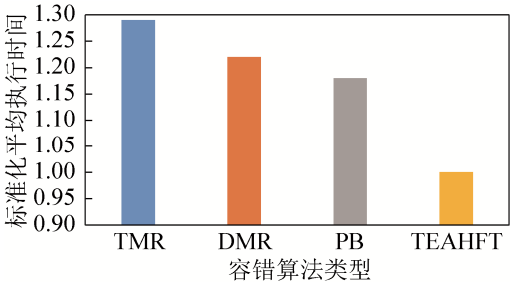
通过数据可以发现，随着注入错误数的增加，TEAHFT 方法的性能优势越来越小，这主要由于错误数的增多，执行应用时需要做容错的任务部分所占比例增多，做竞争任务的部分所占比例减小，所以 TEAHFT 的优势在减弱，而当注入错误数达到 5 000 时，TEAHFT 执行应用的优势缩小至 4.8%。

这种情况在实际应用中很少出现，当执行一个应用程序有如此多的错误时，系统运行中肯定出现了不正常的问题，需要我们去检查相关硬件模块问题。另外可以发现，在执行排序应用时，TEAHFT 的性能优势比其他应用要小，这主要由于在排序中循环操作所占比例较高，这同种类型的操作很难充分发挥异构性的优势作用。

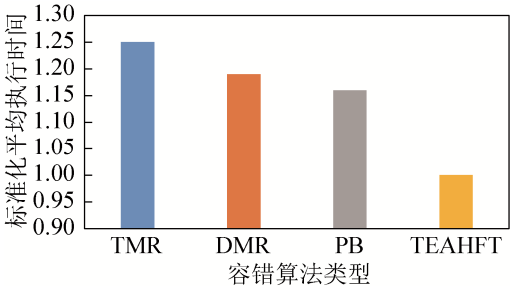
5.2.2 执行功耗分析

Wattch[35]是在 simplescalar 基础上的实现的开放源代码的计算机系统的性能/功耗分析工具，参考[30,36]的方法，分别计算对 TMR，PB 方法与 TEAHFT 执行任务的功耗，未注入错误时，TEAHFT 功耗仅相当于 TMR 的 60%，这主要由于对于传统应用，在 TEAHFT 算法里执行竞争部分时，只要领先的内核执行完任务，其他落后的内核就自动停止，这将减少系统整体功耗。分别注入错误后，执行三种应用时，以 TEAHFT 方法的功耗为参考标准，得到相互间功耗关系，如图 9 所示。

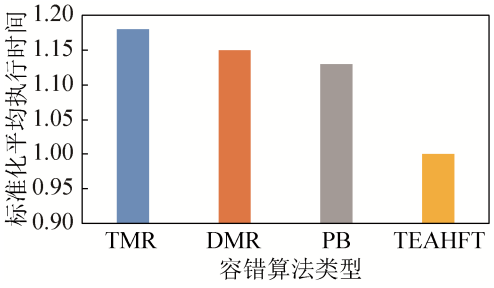
注入 100 个错误时，TEAHFT 比 TMR，PB 方法功耗分别优化了 28.1%，16.1%。由此可见，随着注入错误数的增加，TEAHFT 的功耗也在提高，这主要由于，错误增多，做三模冗余运算部分增多，这导致 TEAHFT 的功耗也在增加，优势在缩小。而发生 5000 个错误是小概率事件，因此本文的算法具有较好的功耗优势。



(a) 执行 SPEC2000 整型时性能对比

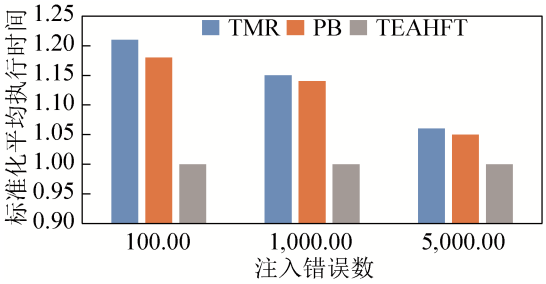


(b) 执行 G.721 时性能对比

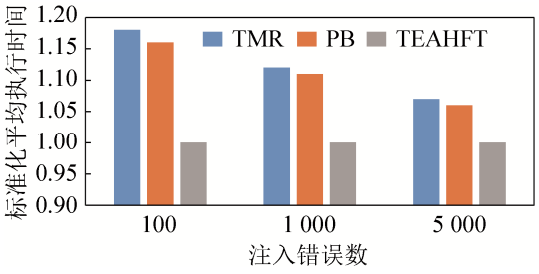


(c) 执行排序时性能对比

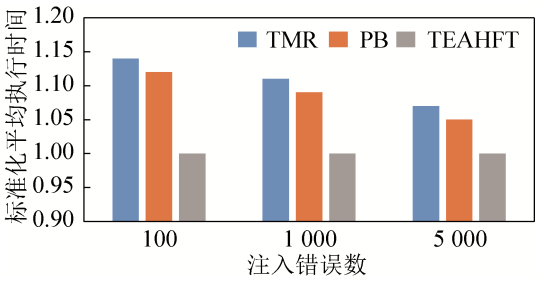
图 7 执行不同应用时的各个算法性能对比
Fig. 7 Performance comparison of algorithms



(a) 注入错误后，执行 SPEC2000 整型时性能对比



(b) 注入错误后，执行 G.721 时性能对比



(c) 注入错误后，执行排序应用时时性能对比

图 8 注入错误后，各个算法性能对比
Fig. 8 Performance comparison after injecting errors

https://dc-china-simulation.researchcommons.org/journal/vol30/iss11/20
DOI: 10.16182/j.issn1004731x.joss.201811020

第 30 卷第 11 期 系统仿真学报 Vol. 30 No. 11
2018 年 11 月 Journal of System Simulation Nov., 2018

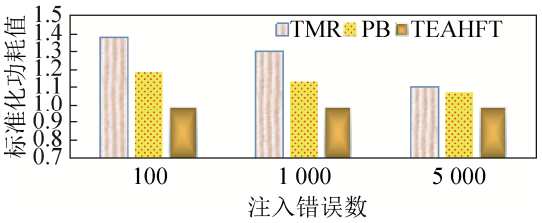


图 9 注入错误时，TMR 与 TEAHFT 功耗对比
Fig. 9 Power comparison after injecting errors

6 结论

TMR 是解决处理器的瞬态故障的一种重要的方法，但是传统的 TMR 存在性能低下，功耗较高，由于异构系统在面对不同类型任务能表现出不同的较高的执行效率，利用异构特点，本文提出了一种新的方法 TEAHFT。该方法引入了竞争机制的思想，面对不同应用，不同的内核呈现出不同的执行效率，对于敏感性任务，执行 TMR 机制，对于有容错功能的任务，利用竞争机制完成任务的执行，这种方法从总体来说，满足了容错需求，但是提高了任务执行效率，同时降低了系统功耗。

本文背景是在冯诺依曼体系结构下，考虑容错的同时，提高单线程的执行效率和降低系统功耗，因为根据 Amdahl 定理，串行应用的性能决定了程序的加速比，本文的方法可以较好应用于并行计算容错之中，例如分布式系统与云计算的容错等领域。下一步我们将结合不同内核体系结构特点，将待执行的任务进行划分，分配至不同内核执行，实现系统具有容错功能的同时，进一步提高系统执行效率并降低功耗。

参考文献:

[1] Jian Chen, Lizy K John. Efficient Program Scheduling for Heterogeneous Multicore Processors[C]. 46th ACM/IEEE Design Automation Conference, San Francisco, CA, USA: IEEE, 2009: 927-930.

[2] 易娟. 面向多核处理器系统的可靠性与能耗优化调度研究[D]. 重庆: 重庆大学, 2016.
Yi Juan. Multiprocessor Systems Scheduling for the Optimization of Reliability and Energy Consumption[D]. Chongqing: Chongqing University, 2016.

[3] 傅忠传, 陈红松, 崔刚, 等. 处理器容错技术研究与展望[J]. 计算机研究与发展, 2007, 44(1): 154-160.
Fu Zhongchuan, Chen Hongsong, Cui Gang, et al. Processor Fault-Tolerance Technology Research and Prospect[J]. Journal of Computer Research and Development, 2007, 44(1): 154-160.

[4] J Karlsson, P Liden, P Dahlgren, et al. Using Heavy-ion Radiation to Validate Faulthandling Mechanisms[J]. IEEE Micro (S0272-1732), 1994, 14(1): 8-23.

[5] S Mukherjee. Architecture Design for Soft Errors[M]. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 2008.

[6] A Ienis. Building Dependable Systems: How to Keep Up with Complexity[C]. International Conference on Fault-tolerant Computing, 1995: 4-14.

[7] 陆阳, 王强, 张本宏, 等. 计算机容错技术研究[J]. 计算机工程, 2010, 36(13): 230-235.
LU Yang, WANG Qiang, ZHANG Ben-hong, et al. Research on Fault-tolerant Technology for Computer System[J]. Computer Engineering, 2010, 36(13): 230-235.

[8] Xun Jian, Rakesh Kumar. ECC Parity: A Technique for Efficient Memory Error Resilience for Multi- Channel Memory Systems[C]. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New Orleans, LA, USA, 2014: 1035-1046.

[9] John M Bird, Michael K Peters, Travis Z Fullem, et al. Neutron Induced Single Event Upset (SEU) Testing of Commercial Memory Devices with Embedded Error Correction Codes (ECC)[C]. IEEE Radiation Effects Data Workshop(REDW), New Orleans, LA, USA: IEEE, 2017: 17-21.

[10] Omer Subasi, Osman Unsal, Jesus Labarta, et al. CRC-Based Memory Reliability for Task-Parallel HPC Application[C]. International Parallel and Distributed Processing Symposium (IPDPS), 2016: 1101-1112.

[11] Evgeny Tsimbalo, Xenofon Fafoutis, Robert J Piechocki. CRC Error Correction in IoT Applications [J]. IEEE Transactions on Industrial Informatics (S1551-3203), 2017, 13(1): 361-369.

[12] Yu-Peng Hu, Nong Xiao, Xiao-Fan Liu. An Elastic Error Correction Code Technique for NAND Flash-Based Consumer Electronic Devices [J]. IEEE Transactions on Consumer Electronics (S0098-3063), 2013, 59(1): 1-8.

[13] Anjana Balachandran, Nandeesh Veeranna, Benjamin, et al. On Time Redundancy of Fault Tolerant C-Based MPSoCs[C]. IEEE Computer Society Annual

Symposium on VLSI, Pittsburgh, PA, USA, 2016: 631-636.

[14] Dmitry Burlyaev, Pascal Fradet, Alain Girault. Time-Redundancy Transformations for Adaptive Fault-Tolerant Ciruits[C]. NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2015: 1-8.

[15] Tuo Li, Muhammad Shafique, Jude Angelo Ambrose, et al. Fine-Grained Checkpoint Recovery for Application-Specific Instruction-Set Processors[J]. IEEE Transactions on Computers(S0018-9340), 2017, 66(4): 647-660.

[16] Nosayba El-Sayed, Bianca Schroeder. Understanding Practical Tradeoffs in HPC Checkpoint-Scheduling Policies[J]. IEEE Transactions on Dependable and Secure Computing (S1545-5971), 2018, 15(2): 336-350.

[17] Pan Zheng, Qi Zheng, Zhankui Zeng. The Signal Integrity Design and Simulation of Triple Modular Redundant (TMR) Computer[C]. IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), Ningbo, China, 2017: 758-762.

[18] Julen Gomez-Cornejo, Aitzol Zuloaga, Uli Kretzs chmar, et al. Fast context reloading lockstep approach for SEUs mitigation in a FPGA soft core processor [C]. 39th Annual Conference of the IEEE Industrial Electronics Society. Vienna, Austria, 2013: 2261-2266.

[19] Luca Sterpone, Luca Boragno. Analysis of Radiation-Induced Cross Domain Errors in TMR Aarchitectures on SRAM-Based FPGAs[C]. 2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS), Thessaloniki, Greece, 2017: 174-179.

[20] Jyothish Soman, Timothy M Jones. High performance fault tolerance through predictive instruction re-execution [C]. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Cambridge, UK, 2017: 1-4.

[21] Wangyuan Zhang, Xin Fu, Tao Li, et al. An Analysis of Microarchitecture Vulnerability to Soft Errors on Simultaneous Multithreaded Architectures[C]. IEEE International Symposium on Performance Analysis of Systems & Software, San Jose, CA, USA, 2007:169-178.

[22] Jie Yin, Jianhui Jiang. Design and analysis of an asynchronous checkpoint-based redundant multithreading architecture[C]. IEEE World Automation Congress, Puerto Vallarta, Mexico, 2012: 519-523.

[23] Jing Wei-peng, Liu Ya-qiu, Wu Qu. Fault-Tolerant Task Scheduling in Multiprocessor Systems Based on Primary-backup Scheme[C]. 3rd International Symposium on Systems and Control in Aeronautics and Astronautics, Harbin, China, 2010: 670-675.

[24] J F Wakerly. Transient Failures in Triple Modular Redundancy Systems with Sequential Modules [J]. IEEE Transactions on Computers (S0018-9340), 1975, 24(5): 570-573.

[25] 贾佳, 杨学军, 李志凌. 一种基于冗余线程的 GPU 多副本容错技术[J]. 计算机研究与发展, 2013, 50(7): 1551-1562.
Jia Jia, Yang Xuejun, Li Zhiling. A Redundancy-Multi thread-Based Multiple GPU Copies Fault-Tolerance Technique[J]. Journal of Computer Research and Development, 2013, 50(7): 1551-1562.

[26] Xiaoxuan She, N Li. Reducing Critical Configuration Bits via Partial TMR for SEU Mitigation in FPGAs[J]. IEEE Transactions on Nuclear Science (S0018-9499), 2017, 64(10): 2626-2632.

[27] Zhuoran Zhao, Dimitris Agiakatsikas, Nguyen T H Nguyen. Fine-Grained Module-Based Error Recovery in FPGA-Based TMR Systems[C]. International Conference on Field-Programmable Technology (FPT), Xi'an, China, 2016: 101-108.

[28] A J Sanchez-Clemente, L Entrena, M García-Valderas. Partial TMR in FPGAs Using Approximate Logic Circuits[J]. IEEE Transactions on Nuclear Science (S0018- 9499), 2016, 63(4): 2233-2240.

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

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

[31] Simplescalar LLC. Simplescalar LLC to serve and project [EB/OL]. [2018-05-30]. http://www.simplescalar.com.

[32] Chunho Lee, M Potkonjak, W H Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems[C]. Proceedings of 30th Annual International Symposium on Micro-architecture, Research Triangle Park, NC, USA,1997: 330-335.

[33] T Austin, E Larson, E Dan. SimpleScalar: An Infrastructure for Computer System Modeling[J]. Computer (S0018-9162), 2002, 35(2): 59-67.

(下转第 4235 页)

DOI: 10.16182/j.issn1004731x.joss.201811020