维普资讯 http://www.cqvip.com

嵌入式实时系统容错集成技术的研究*)

黎忠文

(厦门大学信息科学与技术学院 厦门 361005) (电子科技大学中山学院 中山 528402)

摘 要 本文提出了一种用于嵌入式实时系统的集成检查点回卷、任务重复和DVS的容错方法。该方法支持处理器速度的在线调整,并根据系统的特点,分别插入额外的SCP或CCP点,有效使用检查点的存贮和比较功能,减少任务的执行时间,提高系统性能。通过概率原理导出了该方法任务的平均执行时间。仿真结果表明在DMR系统上,与原有的方法相比,所提出的方法明显减少了任务的平均执行时间。在此基础上,进一步提出了可适配处理器速度的算法,在减少任务执行时间的同时又节约系统能源。本文研究成果也可用于其它任务重复系统,如TMR-F、DMR-F-1和RFCS等。

关键词 设置检查点,任务重复,DMR,DVS,SCP,CCP,CSCP

Research on Fault Tolerance Integration Technology for Embedded Real-Time Systems

LI Zhong-Wen

(Information Science and Technology College, Xiamen University, Xiamen 361005)

(Zhongshan Institute, University of Electronic Science and Technology of China, Zhongshan 528402)

Abstract An integrated fault tolerance approach that provides checkpointing, task duplication and DVS (dynamic voltage scaling) for embedded systems is presented in this paper. This paper shows that, by supporting processors' speed adjustment during task execution and inserting additional SCPs or CCPs according to characteristics of systems, a significant reduction in the execution time can be achieved. With SCPs and CCPs, we can use both the comparison and storage operations in an efficient way and improve the performance of checkpointing schemes. Average execution times to complete a task for the proposed approach are obtained, using the theory of probability. Simulation results show that compared to previous method, the proposed approach significantly reduces the task execution time of DMR checkpointing scheme. Furthermore, an adaptive processors' speed algorithm, combined with the dynamic voltage scaling scheme, is put forwarded to achieve power reduction. My research results may be applied to the other task duplication systems, such as TMR-F, DMR-F-1 and RFCS, etc.

Keywords Checkpointing, Task duplication, Double modular redundancy, Dynamic voltage scaling, Store-checkpoint, Compare-checkpoint, Compare-and-store-checkpoint

1 引言

设置检查点和回卷恢复CRR(checkpointing and rollback recovery)是实时系统中常用的一种容错技术。它定期或不定期地在实时任务的正常运行过程中插入检查点,保存系统状态(关键变量和上下文环境等)。当检测到故障发生后,任务就回卷到最近一个状态相同的检查点继续运行,从而达到有效减少重复量的目的,提高实时任务在死限前的完成率。在故障条件下,如果不使用检查点技术,任务平均执行时间将随任务的有效执行时间,即不发生故障时任务所需的执行时间,呈指数增长;而当采用固定间隔的检查点时,任务平均执行时间则呈线性增长[1]。此外,当故障率超过一定值时,如果两次故障的平均间隔小于任务的有效执行时间时,如不采用检查点,任务很难完成[2]。目前检查点回卷技术已经在软件容错、软件调试器、多处理器系统和移动计算等领域得到广泛研究[3~5]。

将CRR与任务重复(task duplication)相结合还是一种很好的故障检测技术,它具有能检测出几乎任意类型故障的能力,而且开销相对比较低,特别适用于占全部故障数90%以上的瞬态故障的检测和恢复[2]。任务重复的思想是通过让同一任务在不同的处理器上运行,相互比较状态以检测故障的发生。在采用这类技术的系统中,比较、保存和每次因故障而回卷计算所损失的一部分已完成的有效计算量是系统的三大开销。显然不同的系统其保存和比较能力比例不尽相同,甚至差别很大,因此统一对待,很难较准确地分析和提高系统的性能。为此,Ziv和Bruck等人把检查点的比较和保存两种操作分开,并把只用于比较而不进行保存操作的检查点称为CCP(compare-checkpoint, CCP)点,反之则称为SCP(store-checkpoint, SCP)点,传统的检查点则命名为CSCP(compare and store checkpoint, CSCP)点,即比较存贮检查点。Ziv和Bruck根据系统的实际能力,以比较高的频率使用开销比较低的操作,理论分析和实验数据都显示该方法能够显著提高性能[6]。该研究引起了广泛的关注。文[2]用增量检查点设置(incremental checkpointing)方法对Ziv和Bruck的算法进行了改进,进一步减少比较和保存操作的开销。此外,更多的研究集中于CCP和SCP检查点间隔的优化工作[7,8]。

电源的DVS(Dynamic voltage scaling)技术是当前一种很受欢迎的系统在线调节工作电压的技术,特别是在航空、深海探测等能源补充困难的环境更显重要。现代嵌入式微处理器大都具备DVS能力,比如Intel公司采用的speedstep技

*)基金项目:福建省2003年青年科技人才创新基金(2003J020)、福建省2004年自然科学基金(A0410004)、厦门大学新世纪优秀人才支持计划基金,厦门大学院士引进基金。黎忠文 博士后,副教授,研究领域:实时系统高安全和高可靠技术。

维普资讯 http://www.cqvip.com

术[9]。由于系统工作电压的降低或提升与处理器的运行速度紧密关联，利用处理器 DVS 功能的研究较多地集中于调度算法[10]。尽管 DVS 和容错技术在嵌入式实时系统中有着重要意义，然而长期以来，把两者集成在一起的研究却很少[11,12]。文[11]提出了可变电压的情况下动态调整检查点间间隔的算法。文[12]分析了仅一个故障条件下电源管理与容错的开销关系。

本文提出了一种集成 CCR、任务重用和电源 DVS 技术的一种提高系统性能的方法。在此基础上，进一步提出了速度可适配算法，在提高任务完成率的条件下节能。目前尚没有查到相似研究。

2 系统描述

为简化说明，本文以 DMR(double modular redundancy)重复系统为例，所提出的方法同样也适用于其它任务重复系统，如 TMR-F、DMR-F-1 和 RFCS[13]等。实时任务 τ 用三元组 $\langle N,D,T\rangle$ 表示，其中 T 为 τ 两个实例间最小间隔时间；N 是 τ 在处理器速度为 1 且无故障情况下，所需的最坏执行时间。若处理器的速度为 f，则 τ 无故障条件下的执行时间为 $C=\lceil\frac{N}{f}\rceil$；$D$ 为 τ 的死限。设把 τ 等分为 m 个间隔，每个间隔末设置一个 CSCP 点。DMR 环境下，τ 同时在两个处理器上执行，在检查点处两个处理器进行状态比较。如果相同，则认为两个处理器均没有发生故障，于是在该检查点保存处理器状态，并继续向前执行任务；否则，两处理器回卷到前一个相邻的检查点，继续执行任务。

为提高系统的性能，针对通信开销(或存贮开销)为主的系统，用 SCP(或 CCP)点把每个 CSCP 间隔等分 n 个子间隔。于是 τ 被等分为 nm 个间隔，每间隔无故障执行时间为 $E=\lceil\frac{C}{n\times m}\rceil$。SCP 和 CCP 点的工作原理如图 1 和 2 所示。

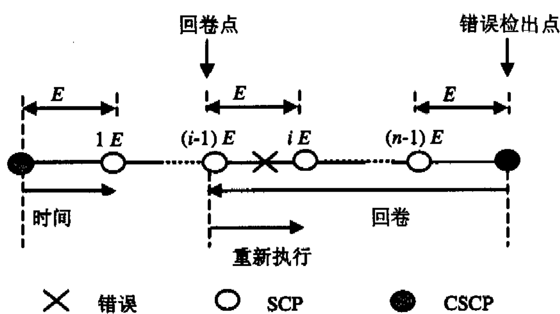


图 1　SCP 检查点工作原理

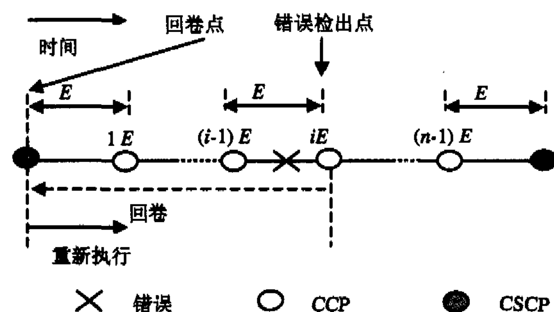


图 2　CCP 检查点工作原理

假设两处理器的故障分布相互独立，且均服从于错误到达率为 λ 的 Poisson 分布。每个处理器有两个工作速度 f_1 和 $f_2(f_1<f_2)$，它们之间的切换开销忽略不计。此外，设在比较和保存检查点状态的操作过程中不发生错误。下文在上述系统及假设条件的基础上，研究容错集成方案。

3 基于 DVS 的检查点方案

实时系统的故障率有严格的要求，在安全域内一般为 10^{-6} 个/小时，有的高达 10^{-9} 个/小时，甚至更高。因此有些研究直接限制任务的一个实例最多只发生一个或两个错误[12,14]。本文以第一个错误为分界线，从系统开工到第一个故障被检测出，并回卷到正确的检查点为止，这段时间令系统以速度 f_1 运行，此后系统提高速度以 f_2 运行。

令 T_s 为存贮处理器状态所需的时间；T_{cp} 为比较处理器状态所需的时间；T_r 为处理器回卷到相应匹配检查点所需的时间。设 ρ 为两处理器在检查点时间间隔内均不发生故障的概率，处理器的速度为 f，显然 $\rho(f)=e^{-2\lambda N/nmf}$。相邻 CSCP 点间不发生故障的概率为 $\rho^n(f)$。又设 δ 为发生第一个故障之前的最后一个检查点间隔，即在间隔 $1,2,\cdots,\delta$ 之中无故障发生，而在间隔 $\delta+1$ 发生故障。φ 表示在检测出故障之前，状态相同的 CSCP 个数，则 $\varphi=\lfloor\frac{\delta}{n}\rfloor$。显然 δ 和 φ 均服从参数为 $\rho(f)$ 和 $\rho^n(f)$ 的几何分布。令 $t_l=\lceil\frac{N}{n\times m}\rceil$。

3.1 插入 SCP 点

本节计算在上述系统环境下插入 SCP 点后，任务 τ 的平均执行时间。

任务的执行时间 T_{SCP} 分为两段，分别用 $T_{SCP1}(f_1)$ 和 $T_{SCP2}(f_2)$ 表示。$T_{SCP1}(f_1)$ 代表从任务开工到检测出第一个错误并回卷到正确的检查点的时间，这段时间内处理器的速度为 f_1；此后处理器以速度 f_2 运行，直到完成任务为止，这段时间用 $T_{SCP2}(f_2)$ 表示。见图 3 所示。

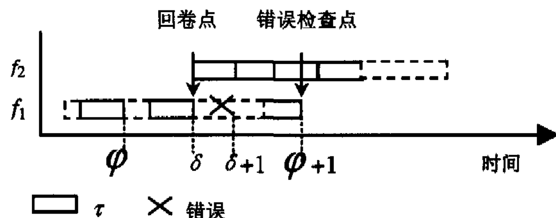


图 3　任务执行情况

3.1.1 T_{SCP1} 平均值的计算

要计算 T_{SCP1} 就必须要求出这段时间内，τ 的有效执行进度 A_{SCP1}，以及为完成该有效进度所花 CPU 时间 D_{SCP1}。

在 SCP 方式下，第一个错误发生在第 $\delta+1$ 个间隔，但在第 $\varphi+1$ 个 CSCP 检查点被检测出来，然后任务回卷到第 δ 个间隔重新执行，这时 τ 的有效执行进度(以间隔计算)为 δ。因此 τ 的平均有效执行进度如(1)式所示。

$$\overline{A}_{SCP1}(f_1)=\overline{\delta}(f_1)=0\times(1-\rho(f_1))+1\times\rho(f_1)(1-\rho(f_1))+2\times\rho^2(f_1)(1-\rho(f_1))+\cdots+k\times\rho^k(f_1)(1-\rho(f_1))+\cdots=\frac{\rho(f_1)}{1-\rho(f_1)} \quad (1)$$

D_{SCP1} 值为从 τ 开工执行到 τ 的第 $\varphi+1$ 个检查点所花时间与回卷至第 δ 个检查点的时间之和，其值见(2)式所示。其中 r_1 代表检出故障后，为找到最后一个状态相同的 SCP 点所需的比较次数。D_{SCP1} 的平均值为(3)式所示，其中 $\overline{r_1}$ 是 r_1 的

维普资讯 http://www.cqvip.com

均值。

$$D_{SCP1}(f_1)=\frac{1}{f_1}[(\varphi(f_1)+1)\times(n\times(t_l+t_s)+t_{cp})+r_1\times t_{cp}] \quad (2)$$

$$\overline{D}_{SCP1}(f_1)=\frac{1}{f_1}[(\overline{\varphi}(f_1)+1)\times(n\times(t_l+t_s)+t_{cp})+\overline{r_1}\times t_{cp}]$$
$$=\frac{1}{f_1}\{[0\times(1-\rho(f_1)^n)+1\times\rho^n(f_1)(1-\rho^n(f_1))+2\times\rho^{2n}(f_1)(1-\rho^n(f_1))+\cdots+k\times\rho^{kn}(f_1)(1-\rho^n(f_1))+\cdots+1]\times(n\times(t_l+t_s)+t_{cp})+\overline{r_1}\times t_{cp}\}$$
$$=\frac{1}{f_1}[(\frac{\rho^n(f_1)}{1-\rho^n(f_1)}+1)\times(n\times(t_l+t_s)+t_{cp})+\overline{r_1}\times t_{cp}]$$
$$=\frac{1}{f_1}\times\frac{1}{1-\rho^n(f_1)}\times(n\times(t_l+t_s)+t_{cp})+\overline{r_1}\times t_{cp}]$$
$$=\frac{1}{f_1}\times[\frac{n\times(t_l+t_s)+t_{cp}}{1-\rho^n(f_1)}+\overline{r_1}\times t_{cp}] \quad (3)$$

(4)式是 T_{SCP1} 的平均值。

$$\overline{T}_{SCP1}=\overline{A}_{SCP1}(f_1)\times\frac{\overline{D}_{SCP1}(f_1)}{\overline{A}_{SCP1}(f_1)}=\overline{D}_{SCP1}(f_1) \quad (4)$$

3.1.2 T_{SCP2}平均值的计算

根据 Poisson 分布的平稳增量性质，可知：$P\{N(\Delta t)\geq 2\}=O(\Delta t)$，即在间隔时间 Δt 充分小时，系统连续多次错误的可能性为 Δt 的高阶无穷小。因此假设两次错误之间的间隔是相互独立的。τ 以速度 f_2 执行时，可以把它看成任务的另一个实例，只不过其计算长度为 $nm-\overline{A}_{SCP1}(f_1)$ 个间隔而已。

与 T_{SCP1} 类似，要计算 T_{SCP2} 就必须要求出这段时间 τ 的有效执行进度 $A_{SCP2}(f_2)$，以及为完成该有效进度所花时间 $D_{SCP2}(f_2)$。与 3.1.1 类似，$A_{SCP2}(f_2)$ 和 $D_{SCP2}(f_2)$ 的平均值分别见(5)式和(6)式所示，其中 $r_2(\overline{r_2})$ 与 $r_1(\overline{r_1})$ 意义相同。

$$\overline{A}_{SCP2}(f_2)=\overline{\delta}(f_2)=\frac{\rho(f_2)}{1-\rho(f_2)} \quad (5)$$

$$\overline{D}_{SCP2}(f_2)=\frac{1}{f_2}[(\overline{\varphi}(f_2)+1)\times(n\times(t_l+t_s)+t_{cp})+\overline{r_2}\times t_{cp}]$$
$$=\frac{1}{f_2}\times[\frac{n\times(t_l+t_s)+t_{cp}}{1-\rho^n(f_2)}+\overline{r_2}\times t_{cp}] \quad (6)$$

(7)式是 T_{SCP2} 的平均值。

$$\overline{T}_{SCP2}=(nm-\overline{A}_{SCP1}(f_1))\times\frac{\overline{D}_{SCP2}(f_2)}{\overline{A}_{SCP2}(f_2)} \quad (7)$$

3.1.3 T_{SCP}平均值的计算

T_{SCP} 为 T_{SCP1} 与 T_{SCP2} 之和，其平均值见(8)式所示。

$$\overline{T}_{SCP}=\overline{T}_{SCP1}(f_1)+\overline{T}_{SCP2}(f_2)$$
$$=\overline{D}_{SCP1}(f_1)+(n\times m-\overline{A}_{SCP1}(f_1))\times\frac{\overline{D}_{SCP2}(f_2)}{\overline{A}_{SCP2}(f_2)} \quad (8)$$

3.2 插入 CCP 点

本节计算在上述系统环境下插入 CCP 点，任务 τ 的平均执行时间。与 3.1 节一样，任务的执行时间 T_{CCP} 也分为两段，分别用 $T_{CCP1}(f_1)$ 和 $T_{CCP2}(f_2)$ 表示。$T_{CCP1}(f_1)$ 代表从任务开工到检测出第一个错误并回卷到正确的检查点的时间，这段时间内处理器的速度为 f_1；此后处理器以速度 f_2 运行，直到完成任务为止，这段时间用 $T_{CCP2}(f_2)$ 表示。

3.2.1 T_{CCP1}平均值的计算

要计算 T_{CCP1} 就必须要求出这段时间内，τ 的有效执行进度 $A_{CCP1}(f_1)$，以及为完成该有效进度所花时间 $D_{CCP1}(f_1)$。

令 $A_{CCP1}(f_1)$ 代表第一个错误被检测出来，任务回卷到相应检查点时，任务的有效执行进度(以间隔表示)。当第一个

$$\overline{A}_{CCP1}(f_1)=n\times\overline{\varphi}(f_1)=\frac{n\times p^n(f_1)}{1-p^n(f_1)} \quad (9)$$

错误在第 $\delta+1$ 个 CCP 点被检测出来后，任务被回卷到第 φ 个 CSCP 点，并从此开始执行。τ 的有效执行进度为 $n\varphi$。因此 A_{CCP1} 的平均值(9)式所示。

为完成上述有效进度所花时间 $D_{CCP1}(f_1)$ 是执行到 τ 的第 $\delta+1$ 个检查点的时间与回卷到第 φ 个 CSCP 检查点所花时间之和。(10)和(11)式分别是 $D_{CCP1}(f_1)$ 及其均值。(12)式是 $T_{CCP1}(f_1)$ 的平均值。

$$D_{CCP1}(f_1)=\frac{1}{f_1}[(\delta(f_1)+1)\times(t_l+t_{cp})+\varphi(f_1)\times t_s+t_r] \quad (10)$$

$$\overline{D}_{CCP2}(f_1)=\frac{1}{f_1}[(\overline{\delta}+1)\times(t_l+t_{cp})+\overline{\varphi}(f_1)\times t_s+t_r]$$
$$=\frac{1}{f_1}[\frac{t_l+t_{cp}}{1-p(f_1)}+\frac{p^n}{1-p^n(f_1)}\times t_s+t_r] \quad (11)$$

$$\overline{T}_{CCP1}(f_1)=\overline{A}_{CCP1}(f_1)\times\frac{\overline{D}_{CCP1}(f_1)}{\overline{A}_{CCP1}(f_1)}$$
$$=\overline{D}_{CCP1}(f_1) \quad (12)$$

3.2.2 T_{CCP2}平均值的计算

3.1.2 小节类似，τ 以速度 f_2 执行时，可以把它看成任务的另一个实例，只不过其计算长度为 $nm-\overline{A}_{CCP1}(f_1)$ 个间隔而已。当回卷到第 φ 个 CSCP 点后，两处理器均以 f_2 速度执行任务 τ。与 T_{CCP1} 类似，要计算 T_{CCP2} 就必须要求出这段时间 τ 的有效执行进度 $A_{CCP2}(f_2)$，和为完成该有效进度所花时间 $D_{CCP2}(f_2)$。A_{CCP2} 的平均值为(13)式，为完成上述有效进度系统所花时间 $D_{CCP2}(f_2)$ 的平均值为(14)式。式(15)是 $T_{CCP2}(f_2)$ 的平均值。

$$\overline{A}_{CCP2}(f_2)=n\times\overline{\varphi}(f_2)=\frac{n\times p^n(f_2)}{1-p^n(f_2)} \quad (13)$$

$$\overline{D}_{CCP2}(f_2)=\frac{1}{f_2}[(\overline{\delta}(f_2)+1)\times(t_l+t_{cp})+\overline{\varphi}(f_2)\times t_s+t_r]$$
$$=\frac{1}{f_2}[\frac{t_l+t_{cp}}{1-p(t_2)}+\frac{p^n(f_2)}{1-p^n(f_2)}\times t_s+t_r] \quad (14)$$

$$\overline{T}_{CCP2}=(nm-\overline{A}_{CCP1}(f_1))\times\frac{\overline{D}_{CCP2}(f_2)}{\overline{A}_{CCP2}(f_2)} \quad (15)$$

3.2.3 T_{CCP}平均值的计算

T_{CCP} 为 T_{CCP1} 与 T_{CCP2} 之和，其平均值见(16)式所示。

$$\overline{T}_{CSCP}=\overline{T}_{CCP1}(f_1)+\overline{T}_{CCP2}(f_2)$$
$$=\overline{D}_{CCP1}(f_1)+(n\times m-\overline{A}_{CCP1}(f_1))\times\frac{\overline{D}_{CCP2}(f_2)}{\overline{A}_{CCP2}(f_2)} \quad (16)$$

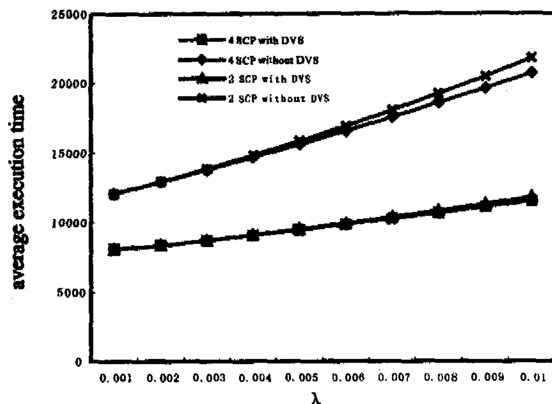
4 性能比较

各参数取与文[2]和[6]相同的值，即在插入 SCP 点方案中，$t_s=5\times10^{-5}$，$t_{cp}=5\times10^{-4}$，$t_r=10^{-5}$，$N=1000$，采用折半查找；而在插入 CCP 方案中，$t_s=5\times10^{-4}$，$t_{cp}=5\times10^{-5}$，$t_r=10^{-4}$。下面比较不支持 DVS 的方案(记为 without DVS)和本方案即支持 DVS(记为 with DVS)下任务的平均执行时间和能源的消耗。设 without DVS 方案中处理器速度 $f_1=1$，with DVS 方案中处理器速度为 $f_1=1$ 和 $f_2=1.5$。这里执行任务所消耗能源的计算采用文[12]的方法，即与任务的执行时间和处理器速度平方成正比，本文取比例常数为 1。

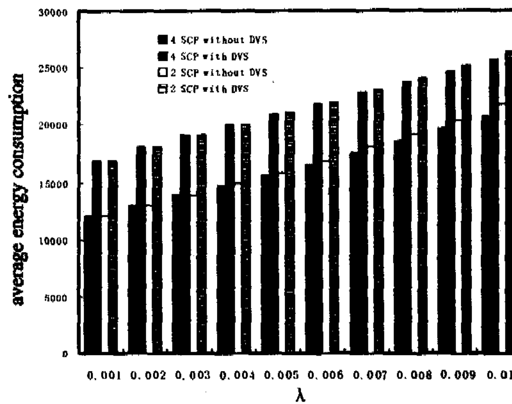
维普资讯 http://www.cqvip.com

图4表示当系统的存贮开销小于比较开销时，我们在CSCP点间插入SCP检查点时，without DVS和with DVS方案下，系统的性能比较。我们画出了插入2和4个SCP的比较结果，从图4(a)可以看出with DVS方案明显地减少了任务的平均执行时间，但其能源消耗却增加了。具体而言，4SCP with DVS和2SCP with DVS方案分别与相应的without DVS方案的平均执行时间之比为59.83%和59.04%；而4SCP without DVS和2SCP without DVS方案分别与相应的with DVS方案的平均耗能之比为75.78%和76.73%。

图5表示当系统的比较开销小于存贮开销时，我们在CSCP点间插入CCP检查点时，with DVS和without DVS方案下系统的性能比较。我们可以得到与图4相似的结论。具体而言，4CCP with DVS和2CCP with DVS方案分别与相应的without DVS方案的平均执行时间之比为59.80%和57.74%；但4CCP without DVS和2CCP without DVS方案分别与相应的with DVS方案的平均耗能之比为75.08%和76.44%。

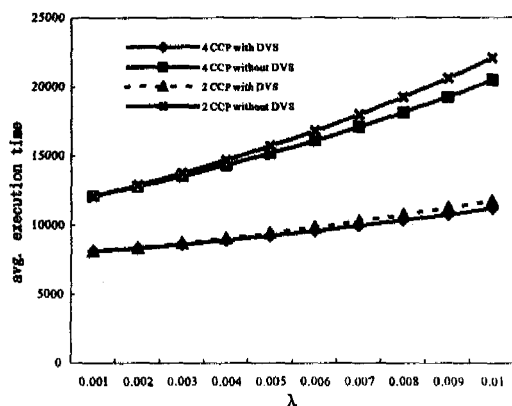


(a)平均执行时间

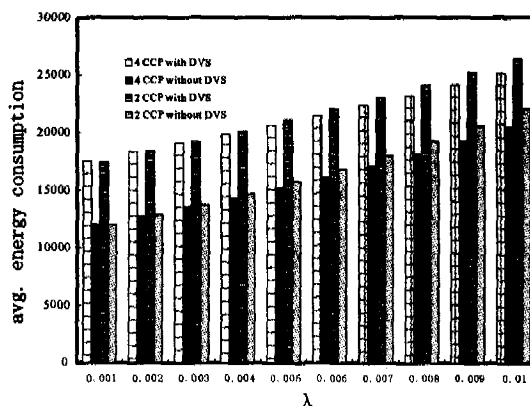


(b)平均能量消耗

图4　支持和不支持DVS性能的对比(SCP)



(a)平均执行时间



(b)平均能量消耗

图5　支持和不支持DVS性能的对比(CCP)

5　可适配的速度切换算法

实时系统，特别是硬实时系统，有着严格的时间限制，关系着系统的成败。本文以上部分，以减少任务的执行时间为目的，尽可能提高实时任务在死限前的完成率，不以节能为首选考虑条件。本节研究节能问题。设T_{est}和R_N分别代表设置检查点方案中，任务τ在故障和无故障时所需要的执行时间(处理器速度为1的情况下)；R_d是任务离死限的时间。λT_{est}是时间段T_{est}内系统潜在的故障数。插入SCP和CCP点，处理器速度为f时，T_{est}分别为：

$$T_{est}(f,SCP)=\frac{1}{f}\{R_N+\lambda T_{est}(f)[n(t_l+t_s)+t_{cp})+\overline{rt_{cp}}]\}$$

$$T_{est}(f,CCP)=\frac{1}{f}\{R_N+\lambda T_{est}(f)[n(t_l+t_{cp})+t_r)]\}$$

等式右边第一项为无故障情况下任务τ的执行时间，等式第二项是恢复λT_{est}个故障的时间。完成τ需满足$T_{est}\leqslant D$。速度可适配算法如adapspeed过程所示。

```
Procedure adapspeed(f1, f2, D, N, λ, model)
1. model=CCP or SCP;
2. if model=SCP RN=m[n(tl+ts)+tcp]; else RN=m[n(tl+tcp)+ts];
3. Rd=D;
4. IF (Test(f1, model)<=Rd) f=f1; else f=f2;
5. While (RN>0)do{
6.   If(RN>Rd)break;
7.   Case1: During normal execution, do{
8.          resume execution;
9.          if find fault then{
10.           roll back and restore status;
11.           Update RN, Rd;
12.           go to 13}}
13. Case 2: Upon fault occurrence, do{
14.   IF(Test(f1, model)<=Rd) f=f1; elsef=f2;
15.         Resume execution;}}
```

算法第4行判断若能以速度f_1完成任务，则处理器速度

维普资讯 http://www.cqvip.com

设置为 f_1，否则设置为 f_2；第 7 至 12 行代表在无错情况下，以选定的速度执行任务，当发现错误后回卷任务并更新 R_N 和 R_d 的值，然后转到第 13 行；第 13 至 15 行代表出错后判断若能以速度 f_1 完成任务，则处理器速度设置为 f_1，否则设置为 f_2。该算法在完成任务的条件下尽可能让处理器低速工作，从而节约能量。

总结 以 DMR(double modular redundancy)系统为例，从插入额外 CCP 和 SCP 检查点的角度，本文提出了一种集成 CCR、任务重用和电源 DVS 技术的一种提高系统性能的方法。在此基础上，进一步提出了速度可适配算法，在提高任务完成率的条件下，节约系统的能量。本文提出的方法同样也适用于其它任务重复系统，如 TMR-F、DMR-F-1 和 RFCS 等。仿真结果表明与原有的方法相比，所提出的方法明显减少了任务的执行时间。下一步将改进本算法，使之适用于多任务环境。

参考文献

1 Duda A. The effects of checkpointing on program execution time, Information processing Letter, 1983, 16(6): 221～229
2 李凯原，杨孝宗. 提高用任务重复的检查点方案的性能. 电子学报，2000，28(5)：33～35
3 周双娥，袁由光，熊兵周，等. 基于任务复制的处理器预分配算法. 计算机学报，2004，27(2)：216～223
4 杨金民，张大方. 一种基于虚拟对象的进程检查点实现方法. 系统仿真学报，2004，16(6)：1354～1357
5 Li Guo-Hui, Wang Hong-Ya. A novel min-process checkpoinging scheme for mobile computing systems. Journal of systems architecture, 2005, 51: 45～61
6 刘建，汪东升，沈美明，郑纬民. 一种基于检查点的并行程序调试器的设计与实现. 计算机研究与发展，2002，39(12)：1580～1586
7 Ziv A, Bruck J. Performance Optimization of Checkpointing Schemes with Task Duplication. IEEE Trans. Computers, 1997, 46(12): 1381～1386
8 Sayori N, Satoshi F, Naohiro I. Optimal checkpointing intervals of three error detection schemes by a double modular redundancy. Mathematical and computer modeling, 2003, 38: 1357～1363
9 Kimura M, Yasui K, Nakagawa T. Optimal checkpointing interval of a communication system with rollback recovery. Mathematical and computer modeling, 2003, 38: 1303～1311
10 Intel Corp. SpeedStep. http://developer. intel. com/mobile/, PentiumIII, 2003
11 Demers Y F, Shenker S. A scheduling model for reduced CPU Energy. Proc. IEEE Ann. Foundations of Computer Science, 1995. 375～382
12 Zhang Y, Chakrabarty. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems, In: Proc. of the design, automation and test in Europe Conference and Exhibition, 2003
13 Melhem R, Mosse D, Elnozahy E. The interplay of power management and fault recovery in real-time systems. IEEE Tran. On computers, 2004, 53(2): 217～231
14 Ziv A, Bruck J. Analysis of checkpointing schemes with task duplication. IEEE Trans. Computer, 1998, 37(2): 222～227
15 George M A, Burns A. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. IEEE Transactions on computers, 2003, 52(10): 1332～1346

(上接第 272 页)

一些准则，在该算法中以树着规则(Tree Coloring Rules)的形式给出，图 3 是应用树着色规则的一个实例，具体的规则如下：

• 如果树的一个节点被选为线程，那么它的祖先节点和后继节点便不能作为线程。这是为了保证所选出来的线程之间不存在重叠的部分。

• 如果树的一个节点被决定不能作为线程，那么它的后继节点中不存在后继节点的，就必须作为线程。这是为了保证所选出来的线程能覆盖整个控制流图。

• 当所有的节点都被决定了是作为线程还是不作为线程后，该过程结束。

采用上述算法对 spec95int 程序进行线程划分，实验结果表明同规模的单芯片多处理器比超标量结构性能提高 50%～80%。

算法设计者研究中发现上述算法的分析和划分方法，决定了体积过小的线程是难以避免的，为了解决这一问题，在目标结构的硬件实现中，增加了线程动态扩展机制，即将多个小线程合并成为一个大线程的监测和控制机制。最终实验表明采用这一优化措施比不采用这一措施又有 8.5%～16%的性能提高。

从整体上看，上述算法对于 3.1 节中所提到的线程划分的关键问题都给出了相应的解决措施，并针对自身难以解决的问题，借助硬件技术进行了再优化。但该算法仍存在一些不足之处，在建立线程的描述方法时，没有考虑程序的 profile 信息，这使线程划分难以定位在程序执行频率最高的部分进行划分和优化；在结构分析和线程的选择过程中没有将线程间的数据相关性作为原则之一，而线程之间的相关性直接影响到线程前瞻的执行效率；最后在划分过程中没有很好的借助编译技术，影响了算法的复杂度和最终效果。

小结 微电子技术的进步使单芯片多线程处理成为一种有潜力的微处理器结构，线程划分是串行程序在其上面高效执行的必要途径，线程级前瞻执行可以大大降低线程划分的复杂度。综合现有的线程划分方法，本文提出了线程自动划分必须解决的关键问题，并结合一个典型算法进行了具体实现分析。通过分析本文认为线程划分方法还存在不成熟和需要进一步研究的方面，主要包括：

• 影响线程划分质量的因素很多，怎样才能综合考虑这些因素？

• 编译技术长于分析，如何将更多快捷的编译技术，如指令的调度策略与线程划分过程相结合？

• 多线程执行微处理器结构各异，是否可以设计一个通用的线程划分方法，通过简单的参数调整就可以适用于各种系统？

参考文献

1 Hammond L, Nayfeh B A, Olukotun K. A Single-Chip Multiprocessor. IEEE Computer Special Issue on "Billion-Transistor Processors", Sept. 1997. 79～85
2 Tsai J Y, Huang J, Amlo C, Lilja D J, Yew P C. The Superthreaded Processor Architecture
3 Tullsen D M, Eggers S J, Levy H M. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In: Proc. of the 22nd Int'l Symposium on Computer Architecture(ISCA), 1995. 392～403
4 Barli N D, Mine H, Sakai S, Tanaka H. Thread Partitioning Method for Speculative Multithreading Architectures. 修士论文东京大学大学院工学系研究科, Feb. 2002
5 Marcuello P, González A. Thread-Spawning Schemes for Speculative Multithreading. HPCA, Feb. 2002. 55～64
6 陈火旺，刘春林，谭庆平，赵克佳，刘越. 程序设计语言编译原理. 国防工业出版社，2000
7 Stallman R M. Using the GNU Compiler Collection for GCC3.3 Dec. 2002