DOI:10.13196/j.cims.2018.06.014

开放式数控系统的低能耗和可靠性调度算法

邓昌义[1,2],郭锐锋[1,2],段立明[1,2],王　帅[1,2],杜少华[3]
(1.中国科学院 沈阳计算技术研究所,辽宁 沈阳　110168;2.中国科学院大学,北京　100049;
3.沈阳高精数控智能技术股份有限公司,辽宁 沈阳　110168)

摘　要:为充分利用多核处理器的并行能力,同时在保证数控系统可靠性约束下实现系统低能耗。在任务级粗粒度软件流水线技术基础上,首先提出非依赖有向无环图算法将周期性依赖任务转换为基于重定时的一组独立任务,通过任务并发执行和动态电压调节技术降低系统能耗。其次,在保证系统可靠性方面,算法分为无错和容错两个阶段。在无错阶段,一个任务的多个副本同时执行,通过投票确定任务是否执行正确。当不一致时,任务进入容错模式。在容错模式下,剩余副本独占处理器以最高速度完成容错。实验结果表明,所提出的算法在保证数控系统可靠性前提下,可以降低系统能耗,相比已有算法不仅具有更高的可靠性而且能耗更低。

关键词:数控系统;多核处理器;实时操作系统;能耗;可靠性;调度
中图分类号:TP311　　**文献标识码**:A

Reliability scheduling algorithm for low power and reliability for open CNC system

DENG Changyi[1,2], *GUO Ruifeng*[1,2], *DUAN Liming*[1,2], *WANG Shuai*[1,2], *DU Shaohua*[3]
(1. Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China;
2. University of Chinese Academy of Sciences, Beijing 100049, China;
3. Shenyang Golding NC Intelligence Tech. Co., Ltd., Shenyang 110168, China)

Abstract: To make full use of parallel ability of multicore for ensuring the real-time system reliability constraints and low power consumption, a novel non-dependent directed acyclic graph algorithm was given based on coarse-grained software pipelining, which could transform a periodic dependency task graph into a set of independent tasks with retiming technique. The system energy consumption could be reduced by task parallel execution and dynamic voltage scaling. In terms of guaranteeing system reliability, the algorithm was divided into two phases: fault-free and fault-tolerant. In the fault-free phase, multiple versions of task were executed at the same time, and results of multiple versions were used for the majority voting to determine whether the task had been executed successfully. If not, the task would enter the fault-tolerant mode. In the fault-tolerant mode, the remaining copies performed the fault tolerance at the highest speed in an exclusive processor. Experimental results showed that the proposed algorithm could reduce the system energy consumption under premise of guaranteeing the reliability of system, which yielded high-reliability and lower energy consumption than the existing algorithms.

Keywords: numerical control system; multicore processor; real-time operating system; energy consumption; reliability; scheduling

0　引言

基于 PC 的开放式数控系统主要分为 NC 嵌入 PC 型、PC 嵌入 NC 型和全软件型 NC 这三种类型结构[1]。嵌入式的 PC 数控系统使得开放式数控系统能随着 PC 技术的发展而升级。开放式数控系统

收稿日期:2017-02-07;**修订日期**:2017-03-30。Received 07 Feb. 2017; accepted 30 Mar. 2017.
基金项目:国家科技重大专项资助项目(2014ZX04014-021)。**Foundation item**: Project supported by the National Science and Technology Major Project, China(No. 2014ZX04014-021)

作为一种典型的实时系统，是数控机床的核心部件。数控系统采用编程指令实时控制数控机床的运动位置，通过伺服系统内的位置闭环、速度环和电流环的调节，完成程序规定的加工任务[2]。数控系统不仅要求在截止期限内完成任务，而且要保证任务的正确执行。随着多核平台以其强大的计算能力被广泛应用在实时系统[3]（例如 AMD APU，Intel Haswell，龙芯 3B1500 和 ARM11 MPCore 等处理器），尽管多处理器系统级芯片（Multicore System-on-chip，MPSOC）的多核架构已被用于开方式数控系统中，但带来的系统能耗越来越高，高能耗带来的高温会导致系统发生故障的可能性增加，进而降低数控系统的可靠性[4]。因此，设计满足高可靠低能耗的开放式数控系统成为一个非常重要的问题。

在开放式数控系统方面，为方便快捷地开发出专用控制器，允许系统进行二次开发。文献[5]提出开放式专用数控系统的核心模型包括：功能模型、数据模型、结构模型和行为模型，该模型采用嵌入式和多 CPU 架构实现了专用数控系统的分布式操作。文献[6]提出面向开放式数控系统的一种基于预分配的空闲挪用算法，算法开始前，采用预分配算法为实时周期任务预留处理器时间，使得在调度周期任务时空闲时间尽可能提前。在调度非周期任务上，采用最大的可用空闲时间机制，实现以较小的计算和较低存储开销获得最短的非周期任务响应时间。

在降低能耗方面，动态电压调节（Dynamic Voltage Scaling，DVS）和动态电源管理（Dynamic Power Management，DPM）被应用于能耗优化。DVS 通过调整处理器的电压降低能耗，DPM 通过利用空闲时间的处理器和关闭电源，以减少静电能耗的产生[7]。多核处理器通过权衡空闲时间长短，选择利用 DVS 调整电压或者使用 DPM 关闭空闲处理器。文献[8]提出一种能耗感知调度技术，在最小化能耗的同时满足吞吐量和响应时间的要求。文献[9]提出一种能耗管理算法，以探索在单处理器系统中使用流水线和并行处理技术。但这种基于单处理器的技术不能直接应用于多核系统。

要实现对瞬态故障的容错，需要故障检测机制[10-11]。然而，常见的故障检测机制不能满足数控系统高可靠性的需要，作为一种故障检测机制，N 模块冗余不需要额外硬件检测故障，该方法使用多数投票的方式进行故障检测和容错[10]。由于所有任务不可能同时出现相同的错误结果，多数投票的方式可以实现对系统故障的检测和容错[10]。但 N 模块冗余需要消耗更多的能耗来实现容错[11]。

为解决基于嵌入式数控系统的可靠性和低能耗之间的协同调度问题，本文首先对依赖任务的有向无环图（Directed Acyclic Graph，DAG）进行建模，使其成为不具有依赖的任务，消除有向无环图任务的优先关系，然后执行协同优化算法以获得更好的能耗结果。该算法利用 N 模块冗余技术，在多个处理单元对每个任务执行相同的副本，并对其结果以投票方式输出。若多个备份任务执行的结果相同，则任务执行正确，并回收容错阶段备份任务的空闲时间，否则，执行容错阶段的剩余副本完成容错。算法充分利用多核架构优势，采用流水线并行执行任务，在任务时序约束下，实现多核处理器系统上的低能耗和高可靠的目标。

1 系统模型与能耗模型

1.1 实时任务模型

作为数控机床的控制核心，数控系统的加工任务不是独立的而是相互依赖，需要处理轨迹插补、位置控制、运动控制等周期性任务，后续任务的完成需要前一个任务的数据。因此本文研究数控系统中的实时周期性依赖任务的调度。

定义 1 DAG 表示一组周期性轨迹插补任务的依赖关系，是数据流图（DFG）的一种特殊情况。DAG 可以表示为 $G=\langle V,E_d,\rho,CM\rangle$，是节点加权和边加权的有向图[12]，其中 V 是节点集，每个节点表示迭代中的任务，E_d 是连接两个节点的有向边集合，ρ 是有向边集合 E 的映射函数，表示两个相邻节点之间的延迟，$CM(u)(u\in V)$ 是用于计算节点 u 的时间函数。

定义 2 DAG 的循环周期指计算无延迟有向边所组成的最长路径需要的时间。同一次循环的相关性使用无延迟边表示，不同次循环间的相关性用延迟边表示。带延迟边 $E_d(u\rightarrow v)$，$G(E)$ 表示节点 V 依赖第 i 次循环产生的数据，而节点 u 的依赖数据在第 $(i-G(E_d))$ 次循环产生。每个节点上的数字表示任务执行需要的时间，“|”表示节点间需要 1 次延迟，“‖”表示需要 2 次延迟。

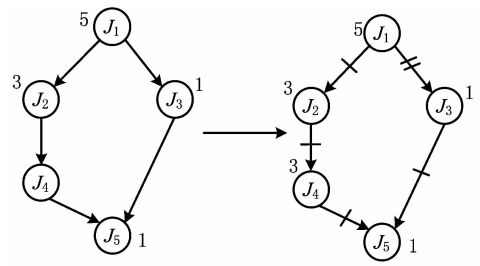
定义 3 一个任务集合有 n 个可并发执行的 DAG 任务 $\{G_1,G_2,G_3,\cdots,G_n\}$，系统中存在 M 个可用的并行处理器。每个 DAG 任务都包含若干个子

任务集 $\sigma_k=\{J_1,J_2,J_3,\cdots,J_k\}$,$J_i=(C_i,T_i)$,$i=1,2,\cdots,k$,同一任务的多个子任务具有一定的时序依赖关系,执行存在先后关系。

任务在每个执行周期内,必须在截至时限 D 要求下执行结束。n 为周期任务的数目,T_i 为任务的周期,任务的周期都等于其截至期限,C_i 为任务在处理器最大速度下的执行时间,即最坏情况下的执行时间。表 1 描述了一个周期任务流的执行时间和对应的能耗。图 1 是基于表 1 的 DAG 模型实例,其中每个节点代表一个任务,两个节点之间的边表示任务优先级关系。

表 1　数控系统周期任务流的时间和对应能耗实例

任务	高频		低频	
	t/mS	E/mJ	t/mS	E/mJ
J_1	5	20	10	5
J_2	3	12	6	3
J_3	1	4	2	1
J_4	3	12	6	3
J_5	1	4	2	1



a DAG模型实例初始状态　　b 重定时技术后的DAG模型实例

图1　数控系统周期任务流DAG模型实例

重定时技术(retiming)利用平均分配 DFG 的延迟实现最小的循环周期[13]。在 DAG 中将节点 V 的每条输入边的延迟通过节点 V 在其每条边上进行标记,通过仅改变节点初始分配值的方式,保持节点间的相关性,而初始分配可以通过装入来实现[13]。实际上,每一次的重定时等同于一次软件流水操作。

1.2　能耗模型和可靠性模型

假设处理器速度频率或速度可以连续调节。当任务 J_i 在速度 S_i 下执行时,任务实际执行时间为 C_i/S_i。对于整个任务集合对应的速度可以表示为 $S=\{S_1,S_2,S_3,\cdots,S_n\}$,这里 S_i 是任务 J_i 运行时对应的速度。本文采用文献[14]的功耗模型,系统的功耗

$$P=P_s+h(P_{ind}+P_{dep})=P_s+h(P_{ind}+C_{cf}\cdot S^m)。\tag{1}$$

式中:P_s 表示系统静态功耗,当整个系统处于关闭状态时 $P_s=0$,静态功耗主要包括维持系统主存和 I/O 设备在睡眠状态下的功耗;P_{ind} 表示和处理器频率无关的功耗;P_{dep} 是依赖于处理器频率的动态功耗,包括 CPU 功耗,以及其他依赖于处理器速度产生的功耗;C_{ef} 表示系统有效的负载电容;m 表示动态指数($2\leqslant m\leqslant 3$);h 表示系统状态,当系统处于活动状态时 $h=1$,当系统处于睡眠状态时 $h=0$。当系统处于活跃状态,一个任务在$\lceil N/2\rceil$个备份下的能耗表示为:

$$E(f_i)=\left\lceil\frac{N}{2}\right\rceil\times(P_{s,i}+h\times(P_{ind,i}+P_{dep,i}))\times\frac{c_i}{f_i}+E_{tr}+E_{com}。\tag{2}$$

式中:E_{tr} 表示处理器状态转换开销,E_{com} 表示处理器间通信能耗开销。

定义 4　任务的可靠性即任务被成功执行的概率。瞬时任务出错概率符合泊松分布[14]。对于支持动态电压调节的实时系统,由瞬时错误引起的任务平均出错率和处理器调整频率 f(对应电压 V)之间的关系,可以表示为:

$$\lambda(f_i)=\lambda_0 10^{\frac{d\cdot(1-f_1)}{1-f_{\min}}}。\tag{3}$$

式中 λ_0 表示在频率 $f_{\max}$ 最大情况下任务平均出错率,当 f 取最大值时 $g(f_{\max}=1)$。低频率或者低电压将导致由瞬时错误引起的任务错误率增加。由于 $f<f_{\max}$,得 $g(f)>1$,$d(d>0)$是一个常量,表示任务出错对 DVS 的敏感度。即当处理器处于最低的频率工作时,具有最高的失效率 $\lambda_{\max}=\lambda_0\cdot 10^d$。

定义 5　系统的可靠性为所有任务成功正确执行的概率。第 i 个任务按最坏时间 C_i 执行,在处理器频率 f_i 下的正确执行的概率为[15]:

$$R(f_i)=\mathrm{e}^{-\lambda(f_i)\cdot\frac{c_i}{f_i}}。\tag{4}$$

定理 1　$R^j(J_1,\cdots,J_n)$表示系统的可靠性,即每一个任务成功执行的可能性。若任务出错,使用空闲时间恢复第 j 个出错的任务。规定 R_g 表示系统要保证的稳定性。当整个系统存在 j 个出错任务,此时可靠性为:

$$R^j(J_1,\cdots,J_n)=R_1(f_1)\cdot R^j\cdot(J_2,\cdots,J_n)+(1-R_1(f_1))\cdot R_1(f_{\max}\cdot R^{j-1}\cdot(J_2,\cdots,J_n))。\tag{5}$$

证明　当整个系统不存在任务,此时可靠性为

$R^0(J_2,\cdots,J_n)=\prod_{i=1}^{n}R_i(f_i)$。相反，实例任务 J_i 的的任务出错率 poF 为：$1-R(f)$。当系统中出现一个任务需要容错则可靠性为：

$$R^1(J_1,\cdots,J_n)=R_1(f_1)\cdot R^1\cdot(J_2,\cdots,J_n)+(1-R_1(f_1))\cdot R_1(f_{\max})\cdot R^0\cdot(J_2,\cdots,J_n)。\tag{6}$$

推广到一般模型，当有 j 个任务需要容错，此时系统的可靠性为式(5)。式中加号前半部分表示任务 J_1 成功执行，余下的 j 个备份任务可以给后续任务使用；加号的后半部分表示 J_1 执行失败，后续任务需要 $j-1$ 个备份任务容错。

问题描述 在保证系统可靠性前提下找到最优的处理器频率，使得总能耗 E 最小：

$$\min:E(f_i)=\left\lceil\frac{N}{2}\right\rceil\times(P_{s,i}+h\times(P_{ind,i}+P_{dep,i}))\times\frac{c_i}{f_i}+E_{tr}+E_{com}。\tag{7}$$

$$\text{s.t.}\quad R^j(J_1,\cdots,J_n)>R_g;\tag{8}$$

$$\sum_{i=1}^{n}C_i/f_i\leqslant D'。\tag{9}$$

式中 $D'=D-BJ$，其中 BJ 是容错任务所需总空闲时间，D 是最坏截止期限。

2 低能耗与可靠性协同优化调度算法

2.1 非依赖有向无环图算法

非依赖有向无环图算法(Non-Dependent Directed Acyclic Graph Algorithm, DPDAGA)用于将周期性 DAG 变换成仅具有数据交互的非依赖性任务。根据重定时的定义，为了将周期性相关的 DAG 变换为一组独立任务，DPDAGA 在初始 DAG 的每个边上添加一个延迟。同时，避免在每个边上添加不必要的冗余延迟，任务非依赖算法为每个节点采用最小重定时值的办法实现消除冗余。基于该规则，式(10)用于计算每个节点的重定时值。

$$r(v)=\begin{cases}\max\{r[v],r[u]+1\}, & \text{若 } v \text{ 是 } u \text{ 的父节点；}\\ 0, & \text{若 } v \text{ 是叶子节点。}\end{cases}\tag{10}$$

式中，$r[v]$和 $r[u]$表示节点 v 和节点 u 的重定时值。DPDAGA 算法中将每个节点的初始重定时值初始化为 0。找出所有叶节点并将它们放入队列 Q 中。Q 的当前尾部元素存储在名为 e_value 的变量中。计算以广度优先的方式进行。当判断节点 u 的父节点 v 是当前队列 Q 的尾部元素，则不需要将节点 v 放入 Q，因为它会产生不必要的冗余计算。所以 e_value 变量是非常有用的。在获得每个节点的重定时值之后，生成仅具有独立任务关系的重定时任务图 Gr。算法的时间复杂度为 $O(n)$，主要来自对节点的出队入队操作。

步骤 1 DPDAGA 算法以周期性依赖任务图 G 为输入，对 G 各个节点进行遍历，并赋值 $r(u)\leftarrow 0$；

步骤 2 遍历 G 的所有节点，若其中一个节点 u 是叶子节点，则将 u 加入队列 Q，若 u 是尾部元素，则 e_value$\leftarrow u$；

步骤 3 若每个 u 的父节点 v 存在，则根据式(10)计算：$r(v)\leftarrow\max\{r(v),r(u)+1\}$，若 e_value $\neq$v，则将 u 加入队列 Q，e_value$\leftarrow v$；

步骤 4 重复以上遍历，直到 G 遍历完。输出每个任务重定时值 r 和它的任务图 Gr

2.2 低能耗与可靠性协同优化调度算法

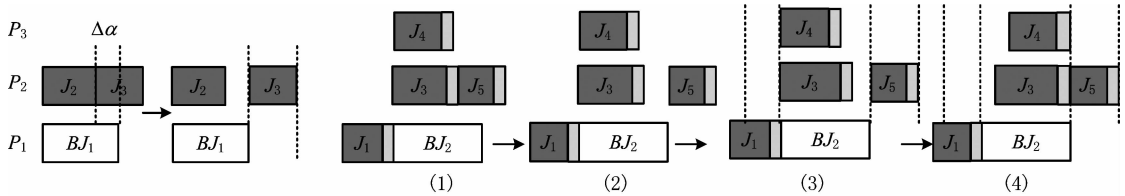
为了保证数控系统的可靠性，在多核平台上结合 DVS 技术采用具有低能耗的 N 模块冗余技术实现系统的可靠性和低能耗。为此，提出了低能耗与可靠性协同优化调度算法(Co-optimal Scheduling Algorithm for Low Power and Reliability on Multicore, CSALPRM)。所提出的 N 模块冗余算法分为两个阶段：

(1)无错阶段 系统正常运行时，每个任务的$\lceil N/2\rceil$个备份并行执行。对于每个任务，将比较$\lceil N/2\rceil$个任务的结果。若$\lceil N/2\rceil$个任务投票结果相同，则没有故障发生，任务成功执行。当投票结果不相同时，系统临时切换到容错阶段。

(2)容错阶段 系统执行剩余备份任务部分，剩余的副本以独占模式执行完成投票。由于任务的$\lceil N/2\rceil$副本已经在无错阶段中被执行，因此在容错阶段中，执行相同任务的剩余$\lfloor N/2\rfloor$副本，一个任务执行完整投票过程至少需要 2 个任务备份。

任务容错模式下，如图 2a 每个处理器最多可以和其他处理器上的一个任务并行执行。而图 2b(1)中，J_2 在处理器 P_1 执行期间和处理器 P_2 上的 J_3，J_5 与 J_2 发生了重叠并行，当任务在容错模式执行完 J_2 后，再返回到无错模式下执行，多核任务重叠将引起多处理器间的任务不能同步。因此，需要对容错阶段任务并行执行这种方式做出调整。如图

2b 中所示,J_2 和任务 J_3 在空间上发生重叠,移动 J_3 及其后续任务,直到 J_2 和 J_3 之间没有重叠,移动的时间量为 $\Delta\alpha$,即 J_2 任务的完成时间和 J_3 任务的开始执行时间之差。基于该规则,在容错模式下实现出错任务独占处理器,同时空间上和其他处理器最多实现一个任务并行执行。如图 2b,经过 4 步实现容错模式下独占处理器。



a 多任务并执行过程　　b 任务独占处理器执行过程

图2　容错模式下任务独占处理器过程

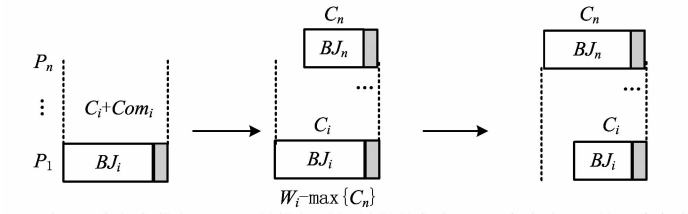
当任务成功执行,容错模式将丢弃备份任务,释放空闲时间以降低能耗。如图 3 所示,容错阶段可以被回收的空闲时间有 3 种情况:

(1)情况Ⅰ(图 3a)　若除了 J_i 外没有任务,则从调度表中丢弃 J_i 时,释放的空闲是 C_i+Com_i,其中 C_i 是 J_i 最坏情况下的执行时间,Com_i 是比较结果(多数投票)或保存结果所需的能耗开销。

(2)情况Ⅱ(图 3b)　若 J_i 是执行时间最长的任务,释放的空闲时间:$C_i-\max\{C_n\}$,则从调度中丢弃 J_i 之后,被回收空闲时间为:$C_i-\max\{C_n\}+Com_i$。

(3)情况Ⅲ(图 3c)　若存在大于 J_i 的一个任务 J_n,则在从调度中移除 J_i 之后,将无空闲时间。因此空闲时间计算公式表示为:

$$\theta=\begin{cases}C_i+Com_i, & \text{仅当 } J_i \text{ 存在;}\\ C_i-\max\{C_n\}, & C_i\geqslant\max\{C_n\};\\ 0, & C_i<C_n。\end{cases}\quad(11)$$



a 只有J_i一个任务执行　b J_i是执行时间最长的任务　c 存在大于J_i的一个任务J_n

图3　回收容错阶段空闲时间

算法的时间复杂度是 $O(N\times M)$,主要来自函数 Tolerance_task(J_i)对调度序列任务中出错任务的查找,以及出错任务 J_i 独占处理器移动其他处理器重叠任务所需要的操作。其中 N 是调度序列 SQ 的任务数,M 是处理器数目。CSALPRM 算法如下:

输入:独立重定向图 Gr;D(任务截止期限);

M(处理器个数);每个任务设置⌈$N/2$⌉个任务备份数

输出:调度序列和系统能耗

1. 初始化调度序列 SQ=∅和当前调度时间长度 L←0.
2. 每个任务分配初始电压,按照任务处理时间降序排列.
3. 每个任务分配 N 个并行拷贝
4. 根据初始任务序列,将⌈N/2⌉个并行拷贝分配给不同的处理器上,获得一个新的执行序列和执行时间长度
5. If(L≤D),
6. 　　则新的 SQ 序列是可调度的,break;
7. 　Else
8. 　　转向步骤 10
9. End if
10. While(True)
11. 　If(L>D)
12. 　　选择调度序列中执行任务时间最长的任务 J_L,计算通信开销,增加该任务的电压,获得新调度序列和当前调度时间长度 L
13. 　Else
14. 　　选择调度序列中执行任务时间最短的任务 J_s,计算通信开销,降低执行该任务的电压,增加任务执行时间 Δt
15. 　If(L+Δt≤D)
16. 　　更新调度序列 SQ 和当前调度时间长度 L=L+Δt
17. 　Else
18. 　　获得新调度序列 SQ,break
19. 　EndIf
20. 　EndIf
21. EndWhile
22. 任务执行阶段,在不同处理器上同时得到一个任务 J_i 的 N 个不同备份任务的执行结果
23. 　If(RT(J_i)==RT(J'_i))
24. 　　J_i 执行正确,break
25. 　　根据式(11)回收容错阶段 BJ_i 备份任务的空闲时间
26. 　else
27. 　　无错模式中断,标记中断位置 IR
28. 　　启动容错模式,出错任务独占处理器,调用 Tolerance_task(J_i)
29. 　If SQ=∅&Slack_time>t。
30. 　　处理器进入睡眠模式

31. Else
32. 保持当前处理器速度运行
33. EndIF
34. EndIF

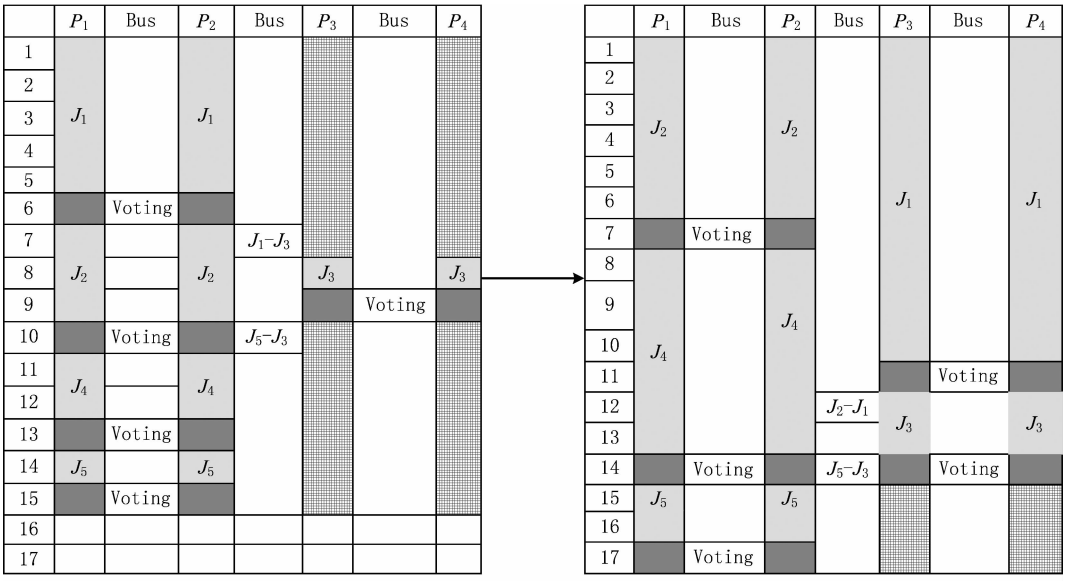
Function Tolerance_task(J_i)

1. For(J_i in SQ)
2. For(i=0;i<M;i++)
3. If 处理器上的备份任务 BJ_i,有其他处理器上多个任务与其并行重叠执行比如 J_{i+1} 和 J_{i+2}
4. Then
5. $\Delta\alpha$=BJ_i 的完成时间−J_{i+2}的开始执行时间(J_{i+2}是其中最长执行时间任务)
6. J_{i+2}以及其处理器上的后续任务向右移动 $\Delta\alpha$
7. End if
8. Endfor
9. Endfor
10. 处理器以最大速度独占处理器执行备份任务 BJ_i,得到执行结果 $RT(BJ_i)$
11. Return $RT(BJ_i)$and IR

3 能耗与可靠性实例分析

3.1 能耗实例分析

本节考虑调度一个有依赖关系的应用程序流,对其进行能耗优化。采用表 1 的任务实例,假设有 4 个处理器,每个处理器都支持动态电压调节。功耗为:$P=C_{ef}\cdot S^2$[14],不失一般性,假设 $C_{ef}=1$ nF,电压和频率对应关系为(2 V,1 GHz)和(1 V,0.5 GHz)。每个任务的执行时间和对应的高电压低电压能耗如表 1 所示。为了更好地阐明问题。假设处理器睡眠模式下能耗为 0.1 W,通过总线的读写通信能耗为 0.5W,每个任务之间的通信时间是 1 ms,所有任务完成的最坏截止时间是 17 ms。如图 4a 所示,该调度过程采用每个任务 2 个备份策略,所有处理器执行任务使用高电压。执行完毕后,对比结果,若相同则该任务正确执行。此时执行图 4b 的任务,总能耗为:$E=2\times20\times5+2\times12\times3+2\times12\times3+2\times4\times1+2\times0.5\times4\times1+2\times0.5\times1+2\times0.5\times1+2\times4\times1+2\times0.1\times7+2\times0.1\times5=368.4$ mJ。



a 无优化调度DAG任务集　　b 优化后调度DAG任务集

图4 使用软件流水并行技术和DVS技术调度DAG任务

图 4b 是利用 DVS 和 DPM 算法以及软件并行技术。首先根据非依赖算法将图 1a 转化为图 1b。之后使用 CSALPRM 算法重新调度任务,调度结果如图 4b 所示。为了尽可能降低系统能耗,该算法使用软件并行技术,则数据依赖迭代产生的空闲时间可以被回收,用于降低处理器速度。但是需要考虑处理器内部通信开销,如 J_1 与 J_2 和 J_3 与 J_5 之间的通信开销。基于此 DVS 和 DPM 回收空闲时间降低系统能耗。所有任务在 15 ms 调度后,在处理器 P_3 和 P_4 剩下空闲时间,由于空闲时间不够长,不足以补偿关闭或转换处理器状态开销,本文所提出的算法将分配与任务 J_3 相同的电压。整个能耗消耗为:$E=2\times6\times3+2\times6\times2+2\times2\times1+2\times5\times10+2\times1\times2+2\times1\times3+2\times0.5\times3+2\times0.5\times1+2\times0.5\times2=180$ mJ。CSALPRM 算法调度任务比按正常顺序调度任务节约 51.14%能耗。可见,经

过并行优化的算法能够大幅降低系统能耗。以上实例只考虑任务不出错情形下的能耗,而真实的执行过程任务会出现错误,此时调度实例见 3.2 节。

3.2　可靠性实例分析

本文提出的 N 模块冗余技术在任务出错时的工作流程如图 5 所示。当任务发生某些故障时所提算法如图 5a 所示进行容错。假设任务 J_2 出现故障,当比较 J_2 的结果时,它们不匹配,因此系统暂时切换到容错阶段。在容错阶段中,J_2 的备份任务 BJ_2 调度执行。然后对 J_2 的 2 个副本 J_2' 和 BJ_2 的结果进行多数投票以容错故障。在容错阶段,BJ_2 独占处理器阶段时不执行 J_3,因为在检测 J_2 中的故障之前它已经成功完成,因此不需要执行。当在容错阶段执行 BJ_2 时,J_4 并行执行,其结果保存在内存中,以便用于以后的投票。在执行 BJ_2 之后,系统切换回无错阶段,并从其被中断的点继续执行调度。在切换无错阶段之后,考虑关于任务 J_4 的两种可能的执行情况:

(1)如果在无错阶段(如图 5b)中执行 J_4 期间一个备份发生故障,当对其进行投票时,系统不需要切换到容错阶段,因为作为 2 个副本,一个副本的结果在无错阶段中被获得,另一个副本的结果已经存在于内部存储器中。

(2)如果在无错阶段(如图 5a)中执行 J_4 期间没有发生故障,则不再需要在先前的容错阶段中预先执行 J_4 的备份。

如果 BJ_4 的预先执行出现故障。在这种情况下,当系统在无错阶段中执行 J_4 时,如果没有故障发生(如图 5c),则将 BJ_4 的预先执行的结果丢弃,因此不影响最后的结果。然而,如果在无错阶段中 J_4 的执行也出现故障(如图 5d),由于预先执行的存储值也有故障,三个备份全部出错,此时系统不能容错该故障,调度失败。事实上,系统可以容错能力和备份任务数成正比,在 2 个备份中可以容错任何一个任务出错,在 3 个任务中可以实现任意 2 个错误容错。一般来说,系统可以容错每个任务的 $\lceil N/2 \rceil$ 个故障[16]。在容错阶段中任务的预先执行(例如,图 5 中的 BJ_4)不但可以提高可靠性,而且不会对系统能耗产生显著影响,因为:

1)在容错阶段并行执行,预先执行不会施加任何时间开销。例如,在图 5 中可以看出,当系统必须在容错阶段中执行 BJ_2 时,与其并行的 BJ_4 预先执行。由于 BJ_2 在独占处理器期间相对其他处理器在同一时间具有最长的执行时间,这意味着 BJ_4 的预先执行会延长 BJ_2。事实上,如果没有使用预先执行,在容错阶段期间将浪费处理器资源,因此使用事先执行有助于为容错阶段保留相对较少的空闲时间,从而更多的空闲可用于能耗管理。

2)如果 J_4 在后续的无错阶段没有发生错误,在容错阶段的预先执行可能变得无用,但是它对平均能耗的影响可忽略不计。这是因为仅当在无错阶段中发生故障时才预先执行。

从可靠性的角度来看,这种机制增强了系统的可靠性。从平均能耗的角度分析,考虑图 5 中的 J_2 和 J_4。假设任务执行出现故障的概率为 10^{-4},J_2 和 J_4 的能耗均为 12 mJ。当发生故障时,系统只在无错阶段执行 J_2,消耗 $2\times12 = 24$ mJ。如果在无错阶段中执行 J_2 期间发生故障,系统将在容错阶段执行 BJ_2 和 BJ_4,消耗 $(12+12)= 24$ mJ。因此,执行 J_2 和 J_4 的平均能耗是 $(1-10^{-4})24+10^{-4}(24+24) = 24.002\ 8$ mJ,这和没有故障发生(24 mJ)时的能耗非常接近,平均能耗与无故障能耗相差小于 0.01%。因此预先执行对能耗影响极小,可忽略不计。

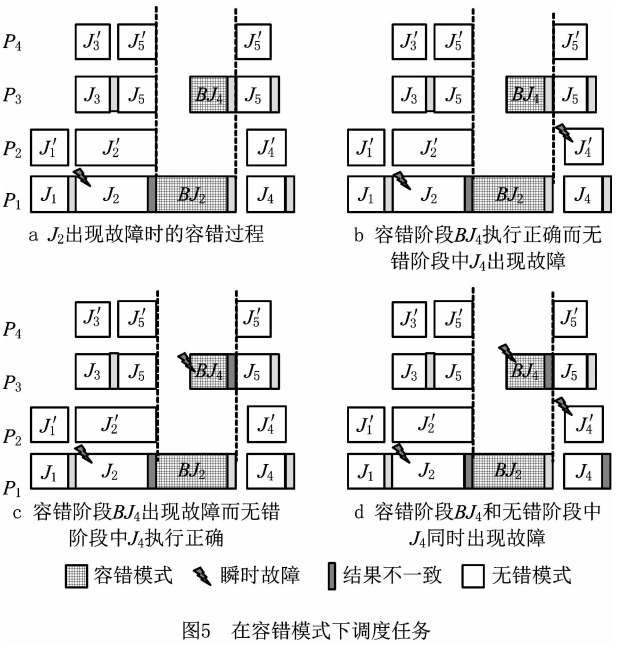


图5　在容错模式下调度任务

3.3　任务集的可调度性分析

定理 2 给出了任务集是否可调度的判断方法。

定理 2　故障情况下,若 σ_k 是在处理机 P_k 上的任务集,σ_k 最大执行时间 $\Delta C_{\max}$ 由式(12)计算,δ_k 的容错截止期 FD 的由式(14)计算。若 σ_k 满足 $\Delta C_{\max} \leqslant FD$,则任务集可调度 σ_k;否则 σ_k 不可被调度。

$$\Delta C_{\max}(J_{\max})=\sum_{J\sigma_k}\lceil \Delta C_{\max}/T_i\rceil\times C_i+\sum_{J\sigma_k}(\varphi\times\Delta C_{\max})\times D_i; \tag{12}$$

此时，

$$\varphi=\begin{cases}\left\lceil\dfrac{t}{T_1}\right\rceil, & t\leqslant BJ_i;\\ \left\lceil\dfrac{t-BJ_1}{T_1}\right\rceil+1, & t>BJ_i。\end{cases} \tag{13}$$

$$\mathrm{FD}(J_{\max})=\begin{cases}T_{\max}, & \text{无错阶段;}\\ BJ_{\max}, & \text{容错阶段。}\end{cases} \tag{14}$$

证明 由于任务的优先级是由高到低进行分配的，所以 $J_{\max}$是当前分配的任务，且 $\delta_k-\{J_{\max}\}$已经是可调度的，只需保证 $J_{\max}$的 C_i 小于其截止期即可，$J_{\max}$是 δ_k 中执行最长的任务且其 C_i 应等于在时间 $\Delta C_{\max}(J_{\max})$内 δ_k 所需要的计算量。在出现处理机故障时，δ_k 上的任务仅有容错阶段任务：

(1)任务在无错阶段每个周期 T_i 内都执行一次，根据式(13)，在时间 t 内共需执行$\lceil t/T_i\rceil$次，当任务在处理机 P_k 上，所需的计算时间量为$\lceil t/T_i\rceil\times C_i$，这部分时间为式(12)等号后的第一部分。

(2)在容错阶段，第一个请求必须在恢复时间 BJ_i 内完成，而后续请求的周期都为 T_i，因此若 $t\leqslant BJ_i$，则仅第一个请求在时间$[0,t]$被执行，即 $\varphi=\lceil t/T_i\rceil$，否则在时间$[0,BJ_i]$内请求执行一次，在$[BJ_i,t]$内请求执行$\lceil(t-BJ_i)/T_i\rceil$，此时根据式(13)，$\varphi=\lceil t-BJ_i/T_i\rceil+1$。

综上所述，可由式(12)计算得到 $J_{\max}$的 C_i。对于 $J_{\max}$的容错截止期，$J_{\max}$为在无错阶段的任务时，$\mathrm{FD}(J_{\max})$为 $J_{\max}$的周期 $T_{\max}$。当 $J_{\max}$为容错阶段的任务时，由于 $T_{\max}>BJ_{\max}$，只需保证 $J_{\max}$的第一个任务的 C_i 小于 $BJ_{\max}$，则 $J_{\max}$在以后的周期中均可调度，故此时 $\mathrm{FD}(J_{\max})$为 $BJ_{\max}$。显然若 $\Delta C_{\max}\leqslant\mathrm{FD}(J_{\max})$，则任务集 δ_k 可调度。证毕.

4 实验结果与分析

本文实验采用沈阳蓝天数控 GJ400，系统处理器采用支持动态电压调节的低功耗 64 位龙芯 3B—1 500 处理器，该处理器允许调整的电压范围为：1.0—1.3 V，对应可变的处理器频率为 1.0 GHz～1.5 GHz[17]。蓝天数控 GJ400 的核心功能模块如图 6 所示，其中运动控制器和 PLC 控制器负责进给驱动控制、主轴驱动控制及开关量控制，实现任务严格的实时性要求。任务管理器作为整个系统的管理中心，可以实现用户界面的操作信息，并为运动控制器与 PLC 控制器传递处理后的任务信息。在实验中，修改操作系统内核，以便在创建线程之前，系统调用可以将获得的任务信息发送到内核中。当任务调用 Pthread_create()创建线程时，它会依次调用一系列系统调用函数，包括 Sys_clone()，Do_fork()和 Copy_process()函数。要调度新创建的线程，系统调度程序使用 Sched_fork()函数。修改后，Copy_process()和 Sched_fork()函数将收集到的任务执行信息反馈到本文的能耗模型和可靠性模型，用于分析所提出算法的能耗和可靠性。

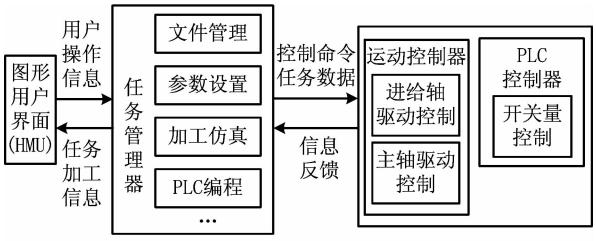


图6 基于龙芯CPU的蓝天数控系统功能模块图

数控系统每执行一次轨迹插补任务的周期为 6 ms，每次插补计算的结果通过命令寄存器分若干次传送到伺服控制单元以控制机床各个坐标轴的位移，位置控制任务的周期为 3 ms；每执行一次运动控制处理任务的时间为 15 ms[6]。为了比较 CSALPRM 算法的效果，实验使用合成应用任务图。为此，使用任务图生成器(Task Graphs for Free，TGFF)[17]。TGFF 基准套件包含合成任务图，每个任务的周期按照数控系统的实际周期产生，任务类型包括插补任务，加减速任务和运动控制任务。当处理器长时间没有任务调度时便进入睡眠模式并仅消耗空闲能耗。将上述 TGFF 生产的任务进行系统可靠性试验，其中可靠性通过式(6)计算任务出错率 $POF=(1-R(f_i))$。使用式(3)的参数 $\lambda_0=10^{-6}$个故障/s 和 $d=3$ 对故障率进行建模[14]。因此，对应于最大和最小电压，故障率在 10^{-6}个故障/s 至 10^{-3}个故障/s 之间变化。为了考虑故障检测机制对能耗和可靠性的影响，使用文献中实现的两种类型的软件故障检测机制进行检测：

1)重度故障检测机制(称为＋HFT) 具有高故障检测开销和相对高的故障覆盖。对于这种情况，假设系统使用基于代码和数据冗余、一致性检查和控制流检查等多种故障检测机制[10-11]，为不同故障类型实现高故障覆盖。

2)轻故障检测机制(称为－LFT) 具有相对低

的故障检测开销以及低故障覆盖。对于这种情况假设系统使用较少的机制来减少检测覆盖,以降低故障检测开销的成本[18]。

4.1　能耗和可靠性分析

表 2 和图 7 分别显示了在八核处理器上执行每种类型任务的实验结果。从图中可知:CSALPRM 不仅比＋HFT 更节能(平均 15.18%),还具有更少的任务出错率,即 CSALPRM 更可靠。与－LFT 相比,CSALPRM 虽然提供相对较少的节能(平均 10.13%),但 CSALPRM 可靠性更好。这是因为 CSALPRM 算法采用并行流水线技术,充分利用了多核处理器的优势,而且在保证可靠性上采用 N 模块冗余,相对＋HFT 和－LFT 的错误检测机制更具优势,不但能够最大限度保证系统可靠性,而且在系统无错时回收多余的空闲时间降低能耗。

表 2　不同任务下三种算法的出错率对比

任务实例	任务出错率 POF		
	＋HFI	－LFT	CSALPRM
插补任务	$10^{-4.35}$	$10^{-2.35}$	$10^{-8.65}$
加减速任务	$10^{-4.69}$	$10^{-2.4}$	$10^{-8.52}$
运动控制任务	$10^{-4.15}$	$10^{-2.65}$	$10^{-8.41}$

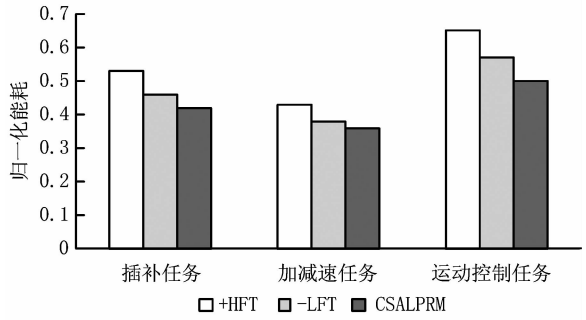


图7　不同任务下三种算法的能耗对比

4.2　处理器个数对能耗和可靠性的影响

从图 8 可以看出,在任务数不变的情况下,随着处理器个数的增加,3 个算法的能耗均降低,这是因为处理器个数多,处理器处于空闲的机率增大,用于 DVS 调节处理器速度的空闲时间就多,因而消耗的能耗少,但 CSALPRM 算法的能耗始终最小,这是因为 CSALPRM 算法将周期性依赖任务转化为独立任务,可以实现更高层级的任务并行。并且充分利用了处理器的空闲时间用于降低能耗,同时对于长时间处于空闲的处理器采用 DPM 技术使其进入睡眠模式,进一步降低能耗。从图 9 可以看出,随着处理器个数的增加,任务出错率降低,系统可靠性增加,这是因为处理器个数越多,处理器用于容错阶段的时间就越多,从而增强了系统的可靠性。但所提出的 CSALPRM 算法,始终保持了较低的出错率,说明所提出的算法使用 N 模块冗余的优越性。

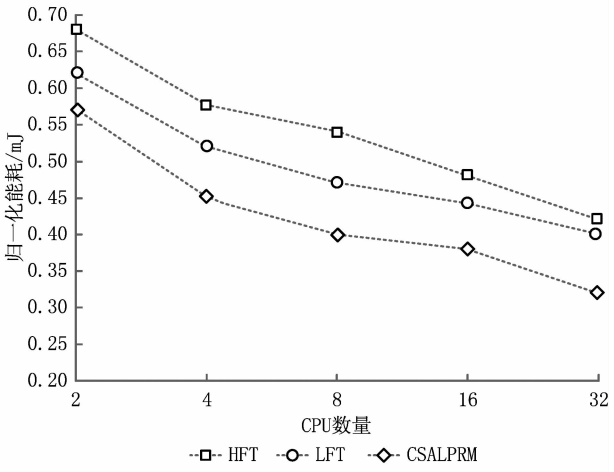


图8　不同处理器数对能耗的影响

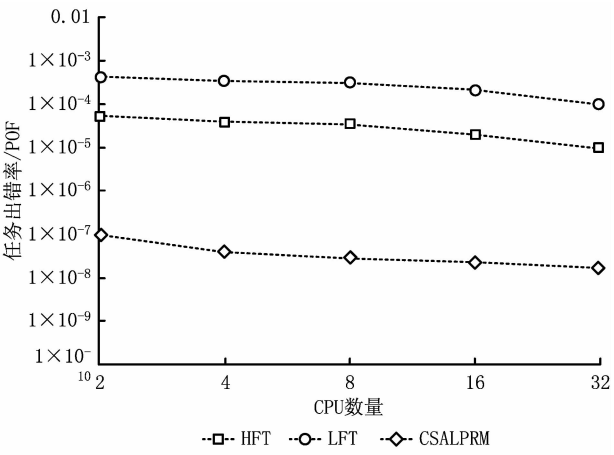


图9 不同处理器数对可靠性的影响

5　结束语

本文面向开放式数控系统研究基于国产 CPU 数控系统的低能耗和可靠性,提出一种两阶段调度算法来解决数控系统在多处理器上调度具有严格时序约束的周期性依赖任务,同时考虑了转换开销和通信开销。实现国产处理器下数控系统的高可靠性与低能耗的协同优化。提出基于粗粒度任务级软件流水线技术的 DPDAGA 算法,用于将定期依赖任务转换为基于重新定时技术的一组独立任务,该算法可以充分利用多核处理器的优势,将任务调度并行化,增强系统调度效率。考虑数控系统中任务加工过程可能出现的错误,基于 DPDAGA 算法基础

上，提出了低能耗与可靠性协同优化调度算法CSALPRM。在CSALPRM中首先采用N模块冗余技术，实现系统的高可靠性，同一任务执行不同副本，并将执行结果进行多数投票，决定任务是否需要容错。其次使用DVS和DPM技术管理容错阶段所产生的空闲时间，并在运行时为任务分配动态空闲。该算法实现了数控系统的低能耗和可靠性之间的协同，增强了数控系统的整体可靠性，并且可以在支持DVS的多核处理器上使用，而不需要增加额外的硬件。下一步基于开放式数控系统，研究数控系统在多核处理器上的的混合任务低功耗调度算法。

参考文献：

[1] ZHOU Zude, LONG Yihong, LIU Quan. Embedded based network numerical control technology and system[J]. Chinese Journal of Mechanical Engineering, 2007, 43(5): 1-7(in Chinese). [周祖德，龙毅宏，刘 泉. 嵌入式网络数控技术与系统[J]. 机械工程学报，2007，43(5)：1-7.]

[2] ZHOU Gang, WU Yijie, PAN Xiaohong. Software modules real time scheduling framework in CNC system[J]. Journal of Mechanical Engineering, 2009, 45(1): 162-166. [周 刚，邬义杰，潘晓弘. 数控系统软件模块实时调度方法[J]. 机械工程学报，2009，45(1)：162-166.]

[3] ANDERSON J H, BARUAH S K. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms[C]//Proceedings of International Conference on Distributed Computing Systems. Washington D. C., USA: IEEE, 2004: 428-435.

[4] ZHU D, MELHEM R, MOSSÉ D. The effects of energy management on reliability in real-time embedded systems[C]// Proceedings of the IEEE/ACM International Conference on Computer Aided Design. Washington D. C., USA: IEEE, 2004: 35-40.

[5] LIU Ning, WANG Gao. Development of open architecture special NC system based on embedded multi-CPUs structure [J]. Computer Integrated Manufacturing Systems, 2012, 18(8): 1854-1860(in Chinese). [柳 宁，王 高. 嵌入式多CPU架构的开放式专用数控系统的研发[J]. 计算机集成制造系统，2012，18(8)：1854-1860.]

[6] LIU Xian, GUO Ruifeng. Preallocation-based hybrid task scheduling algorithm in numerical control system[J]. Computer Integrated Manufacturing Systems, 2015, 21(6): 1529-1535(in Chinese). [刘 娴，郭锐锋. 数控系统中基于预分配的混合任务调度算法[J]. 计算机集成制造系统，2015，21(6)：1529-1535.]

[7] XU R, MELHEM R, MOSS D. Energy-aware scheduling for streaming applications on chip multiprocessors[C]. Proceedings of the IEEE International Real-Time Systems Symposium. Washington, D. C., USA: IEEE 2007: 25-38.

[8] KIM N S, KGIL T, BOWMAN K, et al. Total power-optimal pipelining and parallel processing under process variations in nanometer technology[C]//Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Washington, D. C., USA: IEEE, 2005: 535-540.

[9] SHAO Z, WANG M, CHEN Y, et al. Real-time dynamic voltage loop scheduling for multi-core embedded systems[J]. IEEE Transactions on Circuits & Systems II Express Briefs, 2007, 54(5): 445-449.

[10] AVIZIENS A. Fault-tolerant systems[M]. San Francisco, Cal., USA: Morgan Kaufmann Publishers Inc., 2007.

[11] MILLER T, SURAPANENI N, TEODORESOU R. Flexible error protection for energy efficient reliable architectures [C]// Proceedings of the International Symposium on Computer Architecture and High PERFORMANCE Computing. Washington, D. C., USA: IEEE, 2010: 1-8.

[12] XU Rongbin, LIU Xin, YANG Zhuangzhuang, et al. Directed acyclic graph real-time scheduling method based on deadline of task execution[J]. Computer Integrated Manufacturing Systems, 2016, 22(2): 455-464(in Chinese). [许荣斌，刘鑫，杨壮壮，等. 基于任务执行截止期限的有向无环图实时调度方法[J]. 计算机集成制造系统，2016，22(2)：455-464.]

[13] DICK R P, RHODES D L, WOLF W. TGFF task graphs for free[C]// Proceedings of the 6th International Workshop on Hardware-Software Codesign. Washington, D. C., USA: IEEE, 1998: 97-101.

[14] EJLALI A, AL-HASHIMI B M, ELES P. Low-energy standby-sparing for hard real-time systems[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(3): 329-342.

[15] ZHAO B, AYDIN H, ZHU D. Generalized reliability-oriented energy management for real-time embedded applications [C]// Proceedings of the Design Automation Conference. Washington, D. C., USA: IEEE, 2011: 381-386.

[16] SALEHI M, EJLALI A, AL-HASHIMI B M. Two-phase low-energy N-modular redundancy for hard real-time multi-core systems[J]. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(5): 1497-1510.

[17] WANG G, LI W, HEI X. Energy-aware real-time scheduling on Heterogeneous Multi-Processor[C]//Proceedings of the 49th Annual Cenference on Information Sciences and Systems. Washington D. C., USA: IEEE, 2015: 1-7.

[18] MISHRA R, RASTOGI N, ZHU D, et al. Energy aware scheduling for distributed real-time systems[C]//Proceedmgs of the Parallel and Distributed Processing Symposium. Washington D. C., USA: IEEE, 2003: 21-22.

作者简介：

邓昌义(1989－)，男，河南周口人，博士研究生，研究方向：实时系统，移动计算等，E-mail：dengchangyi@sict.ac.cn；

郭锐锋(1968－)，男，辽宁沈阳人，研究员，博士生导师，研究方向：实时系统，数控系统等；

段立明(1978－)，男，辽宁大连人，博士研究生，研究方向：实时系统，智能监控等；

王 帅(1992－)，女，辽宁沈阳人，博士研究生，研究方向：虚拟现实与可视化，计算机辅助设计与图形学等；

杜少华(1982－)，男，辽宁沈阳人，副研究员，研究方向：实时系统，数控系统；

REFERENCES

[1] Z. Z. L. Y. L. Quan, "Embedded-based network numerical control technology and system [j]," *Chinese Journal of Mechanical Engineering*, vol. 5, 2007.

[2] Z. G. W. Y. P. Xiaohong, "Software modules real time scheduling framework in cnc system [j]," *Chinese Journal of Mechanical Engineering*, vol. 1, 2009.

[3] J. H. Anderson and S. K. Baruah, "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms," in *24th International Conference on Distributed Computing Systems, 2004. Proceedings.* IEEE, 2004, pp. 428–435.

[4] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.* IEEE, 2004, pp. 35–40.

[5] L. Ning and W. Gao, "Development of open architecture special nc system based on embedded multi-cpus structure," *Computer Integrated Manufacturing Systems*, no. 8, p. 26, 2012.

[6] F. Zhao, Y. Hong, D. Yu, Y. Yang, Q. Zhang, and H. Yi, "A hybrid algorithm based on particle swarm optimization and simulated annealing to holon task allocation for holonic manufacturing system," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 9-10, pp. 1021–1032, 2007.

[7] R. Xu, R. Melhem, and D. Moss, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, 2007, pp. 25–38.

[8] N. S. Kim, T. Kgil, K. Bowman, V. De, and T. Mudge, "Total power-optimal pipelining and parallel processing under process variations in nanometer technology," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.* IEEE, 2005, pp. 535–540.

[9] Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L. T. Yang, and E. H.-M. Sha, "Real-time dynamic voltage loop scheduling for multi-core embedded systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445–449, 2007.

[10] A. Aviziens, "Fault-tolerant systems," *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1304–1312, 1976.

[11] T. Miller, N. Surapaneni, and R. Teodorescu, "Flexible error protection for energy efficient reliable architectures," in *2010 22nd International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2010, pp. 1–8.

[12] A. Saifullah, J. Li, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," *Real-Time Systems*, vol. 49, no. 4, pp. 404–435, 2013.

[13] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98)*. IEEE, 1998, pp. 97–101.

[14] A. Ejlali, B. M. Al-Hashimi, and P. Eles, "Low-energy standby-sparing for hard real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 3, pp. 329–342, 2012.

[15] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 381–386.

[16] M. Salehi, A. Ejlali, and B. M. Al-Hashimi, "Two-phase low-energy n-modular redundancy for hard real-time multi-core systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1497–1510, 2015.

[17] G. Wang, W. Li, and X. Hei, "Energy-aware real-time scheduling on heterogeneous multi-processor," in *2015 49th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2015, pp. 1–7.

[18] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy aware scheduling for distributed real-time systems," in *Proceedings International Parallel and Distributed Processing Symposium*. IEEE, 2003, pp. 9–pp.