

基于多核处理器的 K 线程低能耗的任务调度优化算法

王科特 王力生 廖新考

(同济大学电子与信息工程学院计算机系 上海 201804)

摘要 针对具有独立 DVFS 的多核处理器系统,提出了一种 K 线程低能耗模型的并行任务调度优化算法(Tasks Optimization based on Energy-Effectiveness Model, TO-EEM)。与传统的并行任务节能调度相比,该算法的主要目标是不仅通过降低处理器频率来减少处理器瞬时功耗,而且结合并行任务间的同步互斥所造成的线程阻塞情况,合理分配线程资源来减少线程同步时间,优化并行性能;保证任务在一定的并行加速比性能前提下,提高资源利用率,减少能耗,达到程序能耗和性能之间的折衷。文中进行了大量模拟实验,结果证明提出的任务优化模型算法节能效果明显,能有效降低处理器的功耗,并始终保持线性加速比。

关键词 多核,能耗优化模型,多线程,多任务并行,资源利用率,同步

中图分类号 TP311.11 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.2.004

K-threaded Low Energy-consuming Task Scheduling Optimization Algorithm Based on Multi-core Processors

WANG Ke-te WANG Li-sheng LIAO Xin-kao

(Department of Computer Science, College of Electronic and Information Engineering, Tongji University, Shanghai 201804, China)

Abstract Based on multi-core processor system with independent DVFS module, this paper proposed a K-threaded low-power optimal algorithm for parallel task modeling which is tasks optimization based on Energy-Effectiveness Model (TO-EEM). Compared with the traditional energy-efficient scheduling of parallel tasks, the main solution for reducing processor power consumption reduces synchronization duration between threads and optimizes parallelism performance not only by decreasing the instantaneous frequency of processors, but also rationally allocating thread resources. Regarding tasks with a certain acceptable speedup performance, we improved resource utilization and reduced energy consumption to reach a compromise between power consumption and program performance. The paper carried a lot of simulation experiments, and the result presents that the proposed task optimization scheduling model has a effective impact on reducing processor power consumption, and still maintains a linear speedup.

Keywords Multi-core, Energy-effectiveness model, Multi-thread, Multi-task parallelism, Resource Usage, Synchronization

1 引言

线程级粗粒度并行的片上多核处理器架构广泛普及,在一些领域,如数据挖掘、计算生物学和计算机图形处理,并行编程针对串行化计算密集型问题的计算等提供潜在的并行性。随着半导体工艺的发展、主频的提高与片上可集成的晶体管规模的增大,功耗和散热问题成为制约处理器发展的性能瓶颈。

在提升处理器性能的同时,如何降低功耗、解决散热问题成为近年来学术界和工业界在体系结构领域所共同关注的热点之一。对于节能调度技术的研究,包括(Dynamic Voltage Frequency Scaling, DVFS)动态电压频率调节技术和(Dynamic

Power Management, DPM) 动态功耗管理技术等硬件节能技术在多核处理器中的应用。文献[1-6]主要通过动态调节降低芯片的运行频率和电压、运行时关闭处理器来减少处理器的静态功耗,优化并行任务调度策略,提高线程同步机制效率,增加产出率,从而达到节能的目的。文献[1]设计了一种启发式处理器合并优化方法,简称 PRO (Processor Reduction Optimizing)。该方法通过合并任务数较少的任务分组,减少处理器使用数目,从而降低系统总能量开销。PRO 方法与 TDS、EAD 和 PEBD 调度结合形成了 3 种优化的调度算法 TDS-PRO、EAD-PRO 和 PEBD-PRO。总体能量开销平均减少 23.71%、21.28% 和 23.51%^[1]。文献[2]针对具有独立 DVFS 的多处理器系统,在考虑处理器状态切换开销的情况

到稿日期:2014-04-08 返修日期:2014-06-03 本文受国家 863 计划项目(2013AA040302),国家高技术研究发展计划(863 计划),高端大规模 PLC 可编程自动化系统研制及应用(2013AA040302),上海经信委重大技术装备研制专项:大型 PLC 控制器的研制及应用(ZB-ZBYZ-03-12-1067-1)资助。

王科特(1986—),男,博士生,主要研究方向为多线程并行计算、基于骨架的并行编程模式、并行编程平台;王力生(1957—),男,教授,博士生导师,主要研究方向为计算机体系结构、嵌入式系统、多线程并行计算、IP 软核、可信计算、FPGA 单片机;廖新考(1986—),男,博士生,主要研究方向为对等网络、信任模型激励机制。

下,提出一种基于帧任务模型的最优节能实时调度算法(Largest Utilization Task First based On Switching Overhead, LUF-SO)。该算法在调度过程中允许实时任务在任意处理器之间迁移,可以在离线调度中确定所有任务的执行过程和执行速度。文献[3]研究了在离散工作模式与实时约束下多核平台下的线性加速比并行实时任务的能耗最小化问题,证明各项任务都在系统中的全部核上执行时能耗最小,将该问题建模为0-1整数线性规划,提出了基于层装箱算法的节能调度算法。文献[4]针对硬实时任务,提出一种多核系统中基于Global EDF的在线节能调度算法GEDF-OLEASA(Global EDF-based on-line energy-aware scheduling algorithm),该算法利用片上DVFS和DPM,降低动态功耗和泄露功耗,达到实时约束与能耗节余之间的合理折衷。文献[5]提出片上多/众核结构中硬件支持的同步机制和细粒度同步机制及基于原语的同步机制,其效率对处理器的性能非常重要,提出并设计了粗粒度同步机制的评估标准和评估方法。文献[6]对并行计算系统引出产出率性能度量因子,建立综合系统可靠性、通信、并行化控制和成本投入要素的产出率并行加速比模型,分析总结模型中各要素影响产出率并行加速比的关键因子,并考虑能耗对系统的影响。

现有的实时系统能耗模型假定处理器的频率值为连续改变的,这不适用于实际系统。本文结合以上节能思想,针对具有独立DVFS的多处理器系统,提出一种K线程低能耗模型的并行任务优化算法(Tasks Optimization based on Energy-Effectiveness Model, TO-EEM)。在满足任务依赖的条件下,优化任务调度,尽量减少功耗消耗,并且保证一定的程序并行效率,与传统的并行任务节能调度相比,其主要目标并不是仅仅通过降低处理器频率来减少处理器瞬时功耗,而是结合任务间的同步情况,合理分配线程以减少线程同步时间,有选择地进行性能优化,保证任务在一定的并行加速比性能前提下,提高资源利用率,减少能耗,达到程序能耗和性能之间的折衷。相比文献[1]通过复制任务减少数据传输时间,但同时增加了程序运行的空间成本,本算法无额外空间开销。文献[2-4]未考虑任务间数据交换、同步锁造成的开销,而本文将线程同步所带来的开销作为模型的一个参数参与计算。

2 系统模型定义

2.1 多核处理器能耗模型

定义1 假设系统仅在休眠(sleep)与活动(active)两个状态迁移,具有 n 个同构核的片上多核处理器,处理器核 $core_i$ 以频率 f 运行时的功耗 $P_i(f)$ 为

$$P_i(f) = P_s + \lambda(P_{ind} + P_d) = P_s + \lambda(P_{ind} + C_{ef}f^m) \quad (1)$$

其中, P_s 为休眠功耗, P_{ind} 为频率无关功耗,主要来自于漏电流产生的泄露功耗。 P_d 为频率相关功耗,主要由门电路电容充放电引起的动态功耗(dynamic power)、短路瞬态电流产生的短路功耗(short-circuit power)构成。 $\lambda=0$ 时,系统处于休眠状态; $\lambda=1$ 时,系统处于活动状态。有效开关电容 C_{ef} 与动态功率指数 m 为与系统相关的常量,且 m 为大于等于2的常数。为了方便讨论,假设最大频率 $f_{max}=1$,最大电压 $V_{max}=1$,记 P_d^{max} 为最大频率相关功耗, $P_s = \alpha P_d^{max}$, $P_{ind} = \beta P_d^{max}$, α, β 为常量。

$$P_i(f) = \alpha P_d^{max} + \lambda(\beta P_d^{max} + C_{ef}f^m) \quad (2)$$

定义2 基于DVFS的多核处理器可以通过调节各核工

作频率来降低各核的功耗,频率调节范围为 $[f_{min}, f_{max}]$,设 θ_i 为 t 时刻 $core_i$ 频率调节系数, $f_i = \theta_i f_{max}$ 。因此, n 个核的处理器运行时功耗为

$$\begin{aligned} P &= \sum_{i=1}^n P_i(f_i) \\ &= \sum_{i=1}^n \{\alpha P_d^{max} + \lambda(\beta P_d^{max} + C_{ef}f_i^m)\} \\ &= \sum_{i=1}^n \{\alpha P_d^{max} + \lambda(\beta P_d^{max} + C_{ef}(\theta_i f_{max})^m)\} \end{aligned} \quad (3)$$

定义3 n 个核的处理器,在单位时钟周期内的功耗为 P ,单位时钟周期内最小功耗为 P_{min} 。

$$\begin{aligned} P &= \sum_{i=1}^n P_i(f_i) / f_i \\ &= \sum_{i=1}^n \{[\alpha P_d^{max} + \lambda(\beta P_d^{max} + C_{ef}f_i^m)]f_i^{-1}\} \\ P &= \sum_{i=1}^n \{[(\alpha P_d^{max} + \lambda\beta P_d^{max})f_i^{-1} + \lambda C_{ef}f_i^{m-1}]\} \\ P &= \sum_{i=1}^n \{[(\alpha P_d^{max} + \lambda\beta P_d^{max})(\theta_i f_{max})^{-1} + \\ &\quad \lambda C_{ef}(\theta_i f_{max})^{m-1}]\} \end{aligned}$$

且 $\exists \theta_i$,当 $\theta_i = f_{max}^{-1} m \sqrt[m]{\frac{\alpha P_d^{max} + \lambda\beta P_d^{max}}{(m-1)\lambda C_{ef}}}$,

$$P_{min} = \sum_{i=1}^n 2(m-1)\lambda C_{ef}^{2/m} + 2(\alpha P_d^{max} + \lambda\beta P_d^{max})^{(m-2)/m} \quad (4)$$

定义4 在时间段 $T, T=t_2-t_1$, n 个核的处理器功耗为 $E(T)$,且与频率调节系数 θ_i 、进程执行总时间 T 相关,满足

$$\begin{aligned} E(T) &= \int_{t_1}^{t_2} P dt \\ &= \int_{t_1}^{t_2} \sum_{i=1}^n \{[(\alpha P_d^{max} + \lambda\beta P_d^{max})f_i^{-1} + \lambda C_{ef}f_i^{m-1}]\} dt \\ &= \int_{t_1}^{t_2} \sum_{i=1}^n \{[(\alpha P_d^{max} + \lambda\beta P_d^{max})(\theta_i f_{max})^{-1} + \\ &\quad \lambda C_{ef}(\theta_i f_{max})^{m-1}]\} dt \end{aligned} \quad (5)$$

2.2 任务模型

程序通过DAG任务图描述了任务并行执行的一定逻辑顺序,每一个节点描述了一个计算任务,存在潜在的并行性。每一个节点代表一个任务,任务可以是一个算法、一个函数或者一种数学逻辑运算等。每条边表示一种任务关系约束,任务之间的关系主要有4种:串行(Serial, S)、派生并行(Fork Parallelism, P_{fork})、聚合并行(Join Parallelism, P_{join})、条件性并行(Condition Parallelism, $P_{condition}$)。任务关系如图1所示,图1(a)中S边连接两个必须串行执行的任务, T_2 依赖于 T_1 ,记 $T_2 \rightarrow T_1$;图1(b)中由 P_{fork} 边连接的子任务可以异步并行执行,且至少存在2个子任务节点, T_2, T_3 依赖于 T_1 ,记 $T_2, T_3 \rightarrow T_1$;图1(c)中 P_{join} 边指向的子任务必须等待所有父任务同步后才能开始执行, T_3 依赖于 T_1, T_2 ,记 $T_3 \rightarrow T_1, T_2$;图1(d)中Pcondition指向的任务可以有条件地派生并行执行,即有条件派生并行, T_2, T_3 分别条件依赖于 T_1 ,记 $T_2, T_3 \rightarrow PT_1$ 。

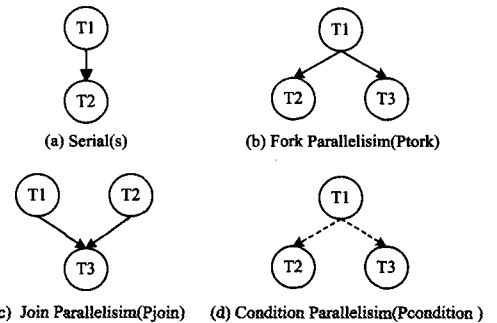


图1 任务图中任务节点并行关系

任务节点的计算分为锁内计算与锁外计算,锁内计算中的操作数在赋值期间不能被其他任务操作,记 t_c 为锁内计算的时间,锁外计算中操作数一般为私有变量或不被共享的公有变量,如图 2 所示。



图 2 任务计算

假设 1 记 t_p 为锁外可并行的计算所消耗的时间。考虑线程的同步有一定的时间消耗,假定每个任务线程同步加锁解锁时的时间消耗为 t_l ,线程上下文切换时间为 t_x ,任务调度时间片为 Δt , T_1 为单线程执行任务集的时间, T_k 为分配 k 个线程执行任务集的总时间。

假设 2 假设任务图中共有 ω 个锁, μ 为运行在锁内计算的任务线程数, k 为执行任务的线程总数, μ 满足

$$\mu = U\langle t_l, t_c, t_p, k \rangle = \lfloor [1 - t_p / (t_l + t_c + t_p)] \times k \rfloor \quad (6)$$

$E\langle \omega, \mu \rangle$ 表示 ω 个锁与 μ 个锁内计算线程相对应的锁竞争期望值。

$$E\langle \omega, \mu \rangle = \left[\frac{\sum_{i=\max(\omega-\mu, 0)}^{\omega-1} (\omega-i) C_{\omega}^i C_{\mu}^{\omega-i-1}}{C_{\omega+\mu-1}^{\omega-1}} \right] \quad (7)$$

假设 3 $P\langle k, n, \omega \rangle$ 为单位时间片内处于锁竞争的概率,与线程数 k 、CPU 核数 n 、任务锁数 ω 有关,满足公式

$$P\langle k, n, \omega \rangle = \frac{C_{k-\omega}^n}{C_k^n} \quad (8)$$

假设 4 本模型考虑线程切换的实践开销 t_x 以及锁竞争导致的线程等待时间消耗,当任务集以 k 线程调度时,总执行时间为 T_k ,且 $T_k = T\langle n, t_x, k, \theta, P \rangle$,满足

$$\begin{aligned} T\langle n, t_x, k, \theta, I, P \rangle &= T_1 / (\theta \times n) + T_1 [\Delta t \times I \times P\langle k, n, E\langle \omega, \mu \rangle \rangle + t_x] / \\ &\quad (\theta \times n \times \Delta t) \\ &= T_1 / (\theta \times n) [(\Delta t + t_x) / \Delta t + I \times P\langle k, n, E\langle \omega, \mu \rangle \rangle] \end{aligned} \quad (9)$$

2.3 研究的问题

假设在具有 DVFS 和 DPM 功能的同构多核处理器上,对于一般多任务进程,考虑任务集中所有的同步开销及线程上下文切换开销对处理器功耗的影响。本文研究的目标是,保证一定的并行性能前提下,根据多线程同步的机率与多线程的切换频率,调节处理器频率与线程数的分配,从而达到降低进程的总功耗的目的。

3 算法思想

对于周期性与非周期性任务构成的一般多任务进程,处理器若始终保持 $\theta_{critical}$ 的频率运行任务集,虽然减少了进程的总执行时间,但因为长时间维持在较高的处理器频率上,以功耗消耗的角度并不能得到最优解。考虑多线程同步机制及线程切换的开销,增加线程的数量、提高处理器运行频率,在一定程度上虽然减少了进程执行时间,但仍然提高了进程的总功耗。减少进程的总功耗即是减少进程每个时间片阶段的

阶段功耗。为减少阶段功耗,需考虑阶段时间内由于锁竞争导致的时间开销,本文考虑在每个 ΔT 周期时间到达时,计算即将运行的任务的数量及锁共享数量,调节线程分配数量,重新最优地计算处理器速度调节因子,实现动态电压/频率调节,最大程度减少进程执行时间,同时尽可能减缓处理器功耗的上升,达到降低进程总体功耗的目的。

3.1 K 线程功耗模型

在每个 ΔT 周期时间内,对于任务集 τ , $\tau_{ij} = 1$ 表示任务 τ_i 的工作频率调节因子恒定为 θ_j ,且 $i \in (1, N]$, $\theta_j \in (0, 1]$ 。将式(6)一式(9)代入式(5)得到关于进程在 T_k 时间段内的功耗 $E(T_k)$,得到方程组(11)。假设进程的任务其依赖关系由 DAG 任务图构成,任务中 t_l, t_c, t_p, t_x 与处理器频率相关,且 $t_l \propto f^{-1}$, $t_c \propto f^{-1}$, $t_p \propto f^{-1}$, $t_x \propto f^{-1}$ 。其中, $\alpha, \beta, \lambda, P_d^{\max}, C_{ef}, n, m, I, \omega, \mu, \zeta, \sigma, \tau, \nu$ 均为先知常量参数,化简后,功耗 $E(T_k) = W(\theta, k)$,为关于线程数 k 与 $core_i$ 频率调节系数 θ_i 的二元曲面函数,加速比 $Speedup = T_1 / T_k$ 。考虑一般任务集的计算量为一定值,任务集执行总时间 T_k 随处理器频率 f 升高而减少,与分配的线程数 k 成反比,并且与任务集中锁的数量 ω 相关。令

$$E(T_k) = W(\theta, k)$$

$$\begin{cases} E(T_k) = W(\theta, k) = \int_0^{T_k} \sum_{i=1}^n \{ [(\alpha P_d^{\max} + \lambda \beta P_d^{\max}) (\theta_i f_{\max})^{-1} + \lambda C_{ef} (\theta_i f_{\max})^{m-1}] \} dt \\ T_k = T\langle n, t_x, k, \theta, I, P \rangle = T_1 / (\theta \times n) [(\Delta t + t_x) / \Delta t + I \times P\langle k, n, E\langle \omega, \mu \rangle \rangle] \\ E\langle \omega, \mu \rangle = \left[\frac{\sum_{i=\max(\omega-\mu, 0)}^{\omega-1} (\omega-i) C_{\omega}^i C_{\mu}^{\omega-i-1}}{C_{\omega+\mu-1}^{\omega-1}} \right], P\langle k, n, \omega \rangle = \frac{C_{k-\omega}^n}{C_k^n} \\ \mu = U\langle t_l, t_c, t_p, k \rangle = \lfloor [1 - t_p / (t_l + t_c + t_p)] \times k \rfloor \\ t_l = \sigma(\theta_i f_{\max})^{-1}, t_c = \zeta(\theta_i f_{\max})^{-1}, t_p = \tau(\theta_i f_{\max})^{-1}, \\ t_x = \nu(\theta_i f_{\max})^{-1} \end{cases} \quad (11)$$

K 线程功耗模型是以多任务进程为衡量对象的低功耗评估模型,其关系模型由 Matlab 仿真得出,如图 3 所示。通过该模型发现,仅仅降低处理器频率可以降低处理器瞬时功耗,在此过程中相应也增加了进程的总执行时间 T_k ,变相增加了整个进程的总功耗;而通过增加处理器频率来减少进程总执行时间,也可能导致进程总功耗的增加。因此,在任务数量大于处理器核数的情况下,多任务进程的总功耗不仅与处理器频率有关,也与分配的线程数量及任务间锁共享数量相关。对于瞬时功耗的极小值所对应的处理器频率调节系数 $\theta_{critical}$ 及线程数量 $k_{critical}$,在 ΔT 周期时间范围内,存在 (θ_i, k_i) ,满足 $\theta_{critical} < \theta_i < 1$,且 $W(\theta_i, k_i) < W(\theta_{critical}, k_{critical})$,即进程在该 ΔT 周期时间内的功耗存在极小值。

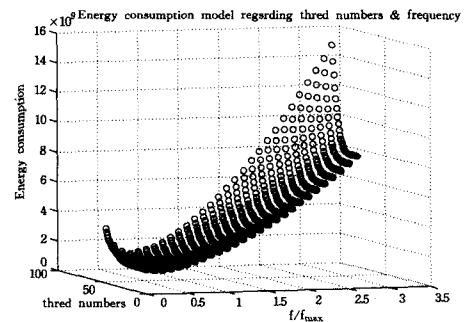


图 3 K 线程功耗模型

3.2 k 线程低功耗任务调度算法

令线程数 $k \in [1, n]$, 频率调节系数 $\theta \in (0, 1]$, 过程如下:

a. 创建一个线程池, 初始化线程池最大线程数为 $MaxPoolSize$, 创建一个守护线程, 用于计算后续最小功耗调节参数与分配线程数量。初始化任务初始化 $Holdqueue$ 队列。

b. 根据任务图 DAG 中的任务依赖顺序, 调用 $Task_Allocate$ 函数, 将任务 t 压入 $Holdqueue$ 队列, 等待与线程的绑定。线程调度器每个 CPU 时间片周期执行 $Task_Scheduler$, 若线程池未满, 该函数创建一个新的线程, 并绑定任务至线程上执行; 否则, 该任务在 $HoldQueue$ 队列里继续等待线程资源。

c. 每间隔周期 ΔT 时间, 函数 $KThread_TOEEM$ 对 DAG 任务图下个周期的任务组做一次并行性检测, 如图 4 所示。

d. 假设待测周期的任务组中共有 ω 个锁, μ 为运行在锁内计算的线程数, k 为执行任务的线程总数, 瞬时最小功耗调节参数为 $\theta_{critical}$, 及此时的线程数量为 $k_{critical}$ 。统计待测周期内将要执行的锁内计算的线程数 μ 及分配的线程总数 k , 根据式(7), 计算 ω 个锁与 μ 个锁内计算线程相对应的锁竞争期望值 $E(\omega, \mu)$ 。

e. 当期望值 E 变化率 R_E 较上次变化幅度超过指定范围 (假定 20%) 时, 根据 k 线程功耗模型, 式(7)计算功耗的极小值所对应的处理器频率调节系数 θ_{act} , k_{act} , 满足 $\theta_{critical} < \theta_{act} < 1$, 且 $W(\theta_{act}, k_{act}) < W(\theta_{critical}, k_{critical})$, 即该阶段总功耗为最低功耗, 记作 $W(\theta_{act}, k_{act})$, 由函数 $KThread_OptW$ 实现。任务运行状态如图 5 所示。

f. 根据 θ_{act} , 调节处理器的执行频率, 为下个周期执行的任务分配 k_{act} 个线程, $MaxPoolSize = k_{act} + 1$ 。

g. 等待下一个周期 ΔT 时间后, 重复执行步骤 b, 直至所有任务完成。

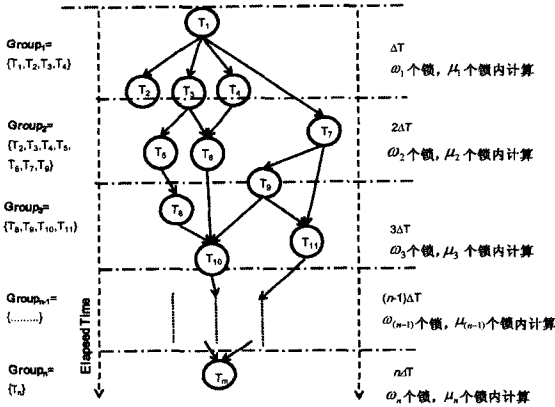


图 4 周期性任务并行性检测

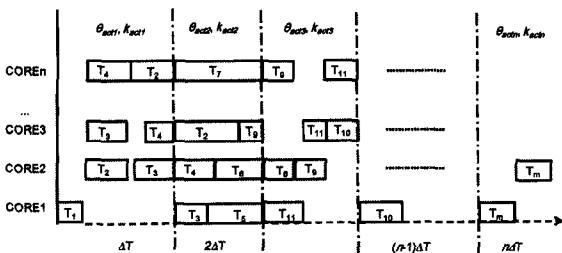


图 5 处理器能耗优化的任务调度策略

3.3 算法设计

算法设计如表 1 所列。

表 1 Design of KThread

FUNCTION KThread_Task_Allocate(Task taski)
Holdqueue, push(taski)
END-FUNCTION
FUNCTION KThread_INIT(int N)
MaxThreadPoolSize=N
Holdqueue, Init()
END-FUNCTION
FUNCTION KThread_Task_Scheduler
WHILE(!ThreadPool, IsFull)
KThread_Start((Task)Holdqueue, Pop())
END-FUNCTION
FUNCTION KThread_Start(Task _t)
((Thread)new Thread(_t)).run()
END-FUNCTION
FUNCTION KThread_TOEEM
WHILE(1)
SLEEP((T)
KThread_OPTW()
END-FUNCTION
FUNCTION KThread_OPTW()
CALCULATE W(theta_act, k_act) //根据步骤 d,e 计算相应的 theta_act, k_act
::SETPARAMETER(theta_act, k_act) //修改 CPU 频率及线程池最大线程数
END-FUNCTION

3.4 计算复杂度分析

假定进程任务数量为 m , k 线程功耗模型每周期计算执行频率 θ_{act} 和分配线程数 k_{act} 的时间复杂度为常数, 记为 $O(1)$ 。若任务存储于顺序表, 每次更新 θ_{act} 和 k_{act} 的时间复杂度为 $O(1) + O(m)$; 若任务已二叉树方式存储, 则每次更新 θ_{act} 和 k_{act} 的时间复杂度为 $O(1) + O(\log_2 m)$ 。因此 k 线程功耗模型在最坏情况下的时间复杂度为 $O(\log_2 m)$ 。由于每周期最坏情况下任务组含有 m 个任务需要计算, 因此在最坏情况下, 空间复杂度为 $O(m)$ 。

3.5 处理器功耗消耗分析

根据模型假设, 每个周期的阶段功耗等于 $W(\theta_{act}, k_{act}) \times \Delta T$, 处理器的总功耗等于每个周期的阶段功耗之和, $W_{total} = \sum_{i=1}^n W(\theta_{act}, k_{act}) \times \Delta T$ 。如图 6 所示, 多任务进程在执行过程中, 优化后的处理器频率在某些时间段略低于优化前的频率, 进程总执行时间虽然相比优化前稍长, 但总能耗却低于优化前的进程。

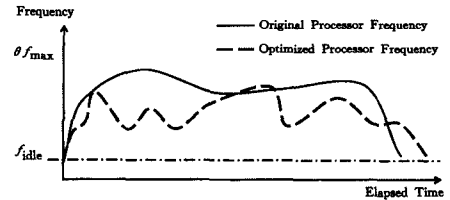


图 6 处理器能耗优化前后的频率

4 性能评估

本模型算法的主要任务是在多核处理器平台上、多项式时间内确定进程在下个周期内的最优工作频率及线程分配方案, 最大限度降低该进程在处理器上的能耗。为验证模型的正确性、有效性及扩展性, 本文在不同多核处理器平台上进行了一系列的模拟实验来验证 k 线程功耗模型算法。

4.1 实验描述

Divide & Conquer 模式是针对大型数据分而治之处理的算法框架。使用该框架需要实例化 divide 方法来划分数据块,conquer 方法合并计算不同数据块。实验使用 MergeSort 并行设计模式实例化了数据规模为 100M 个整型数据集的归并排序算法,其串行伪代码如表 2 所列。

表 2 Parallel MergeSort

```
template<class T>
inline void parallel_merge_sort_r(T* src,int l,int r,T* dst,bool srcToDst=
true) {
    if (r == l && srcToDst) {
        dst[l] = src[l]; return;
    }
    KThread_Task_Allocate([&]{parallel_merge_sort_r(src,l,(r+l)/2,dst,
!srcToDst);},
    [&]{parallel_merge_sort_r(src,(r+l)/2+1,r,dst,!srcToDst);});
    if(srcToDst)merge_parallel_L5(src,l,(r+l)/2,(r+l)/2+1,r,dst,l);
    else merge_parallel_L5(dst,l,(r+l)/2,(r+l)/2+1,r,src,l);
}
inline void parallel_merge_sort(T* src,int l,int r,T* dst,bool srcToDst=
true){
    if (r<l) return;
    parallel_merge_sort_r(src,l,r,dst,srcToDst);
}
```

实验测试平台分别为不同核数的多核 CPU,分别是双核 4 物理线程的 Intel core i5 3230M 2.6GHz、4 核 8 物理线程的 Intel core i7 Q740 1.73GHz 和 6 核 12 物理线程的 Intel Xeon E5-2620 2.00GHz。按照处理器功耗与频率的理论依据,处理器的能耗与离散频率对应关系如表 3 所列,简化后将频率调节系数设置为 0.2、0.4、0.6、0.8、1.0,引入此频率调节系数是为了方便计算 CPU 的即时理论功率。

表 3 离散频率下的多核 CPU 的功耗

序号	频率调节系数 θ		i5 3230M		i7 Q740		Xeon E5-2620	
	频率 MHz	功率 mW	频率 MHz	功率 mW	频率 MHz	功率 mW	频率 MHz	功率 mW
1	0.2	640	7000	586	9000	500	19000	
2	0.4	1280	8400	1172	10800	1000	22800	
3	0.6	1920	12600	1758	16200	1500	34200	
4	0.8	2560	22000	2344	28800	2000	60800	
5	1.0	3200	35000	2930	45000	2500	95000	

4.2 实验结果

MergeSort 程序在不同的多核处理器上分别通过裸线程及 KThread 调度算法优化后计算抓取到的功耗,如图 7—图 9 所示。根据实验数据显示,优化后的程序的 CPU 功耗分别在 i5 3230 处理器上减少 15%,在 i7 Q740 处理器上减少 25.8%,在 Xeon E5-2620 处理器上减少 23.1%。优化后的程序由于频率降低,总运行时间有所增加,相对应的加速比如图 10 所示,与优化前的加速比相比,加速度稍有减小,在不同核数的处理器上加速度较原有加速度分别减少 10.4%、6.6%、12.1%,但仍然接近线性增长。

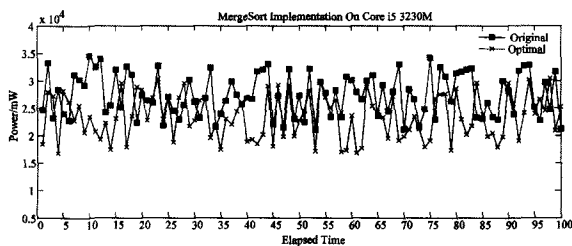


图 7 MergeSort 在 i5 3230M 处理器的功耗比较

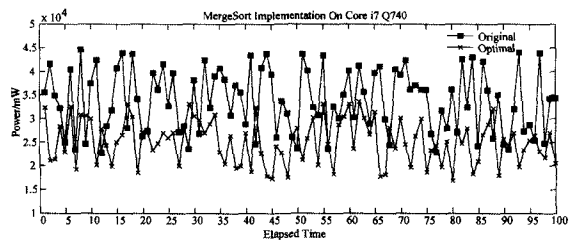


图 8 MergeSort 在 i7 Q740 处理器的功耗比较

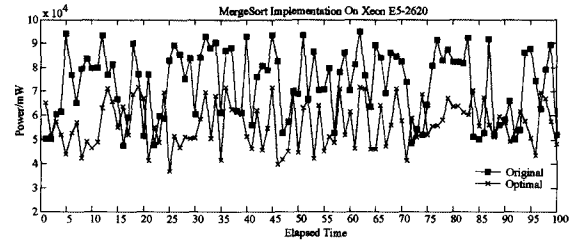


图 9 MergeSort 在 Xeon E5-2620 处理器的功耗比较

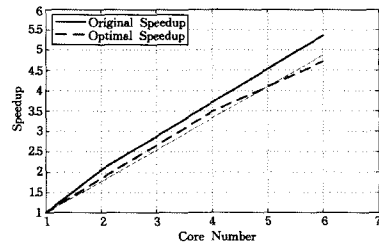


图 10 K 线程调度算法优化后的加速比

结束语 本文提出的 K 线程低能耗模型的并行任务调度优化算法,应用于多核环境下基于裸线程的性能优化方案,与传统的并行任务节能调度相比,主要目标不仅通过降低处理器频率来减少处理器瞬时功耗,并且结合同步互斥所造成的线程阻塞情况,合理分配线程资源,以减少线程同步时间,优化并行性能。理论建模及实验数据证明,并行程序通过降低频率,减少线程同步阻塞造成的 CPU 竞争,能够减少 15%~23.1% 的能耗,优化后的加速比虽有 6.6%~12% 的减少,但仍呈线性增长,达到程序能耗和性能之间的折衷。下一步将进一步研究功耗与程序期望性能的关系,建立更理想更完善的多任务低能耗模型。

参考文献

[1] 李新,贾智平,鞠雷,等.一种面向同构集群系统的并行任务节能调度优化方法[J].计算机学报,2012,35(3):591-602

[2] 张东松,吴飞,陈芳园,等.开敏敏感的多处理器最优节能实时调度算法[J].计算机学报,2012,35(6):1297-1312

[3] 林宇晗,孔繁鑫,徐惠婷,等.线性加速比并行实时任务的节能研究[J].计算机学报,2013,36(2):384-393

[4] 张东松,吴彤,陈芳园,等.多核系统中基于 Global+EDF 的在线节能实时调度算法[J].软件学报,2012,23(4):996-1009

[5] 徐卫志,宋凤龙,刘志勇,等.众核处理器片上同步机制和评估方法研究[J].计算机学报,2010,33(10):1777-1787

[6] 王之元.产出率并行加速比模型[J].计算机工程,2011,37(5):10-12

[7] Bansal N,Kimbrel T,Prush K. Speed scaling to manage energy and temperature[J]. Journal of the ACM,2007,54(1):1-39

[8] Wierman A, Andrew L L H, Tang Ao. Power-aware speed scaling in processor sharing systems: Optimality and robustness

- [J]. Performance Evaluation, 2012(69); 601-622
- [9] Burd T D, Brodersen R W. Energy efficient cmos microprocessor design[C]//Proc. of The HICSS Conference, Jan. 1995
- [10] Benoit A, Cole M, Gilmore S, et al. Flexible Skeletal Programming with eSkel[J]. Submitted to EuroPar 2005, Lisbon, Portugal, 2005, 3648; 761-770
- [11] MacDonald S, Anvik J, Bromling S, et al. From patterns to frameworks to parallel programs[J]. Parallel Computing, 2002 (28); 1663-1683
- [12] Bromling S, MacDonald S, Anvik J, et al. Pattern-based Parallel Programming[C]//Proceedings of the International Conference on Parallel Processing (ICPP'02), 2002
- [13] Macdonald S. Deferring Design Pattern Decisions and Automating Structural Pattern Changes using a Design-Pattern-Based Programming System[J]. ACM Transactions on Programming Languages and Systems, 2007; 1-45
- [14] Yzelman A N, Bisseling R H. An object-oriented bulk synchronous parallel library for multicore programming[M]//Concurrency and Computation: Practice and Experience Concurrency Computat., Pract. Exper. 2012, 24(5); 533-553
- [15] Yu Ce, Xu Zhen. ParaModel: A Visual Modeling and Code Skeleton Generation System for Programming Parallel Applications [J]. ACM SIGPLAN Notices, 2008, 43(4)
- [16] Benoit A, Cole M. Two Fundamental Concepts in Skeletal Parallel Programming[C]//ICCS 2005, LNCS 3515, 2005; 764-771
- [17] Matsuzaki K, Emoto K. Lessons from implementing the biCG-Stab method with SkeTo library[C]//International Conference on Functional Programming, 2010; 15-24
- [18] Benoit A, Cole M, Hillston J, et al. Using eSkel to implement the multiple baseline stereo application [OL]. <http://homepages.inf.ac.uk/mic.pubs/parco05.pdf>
- [19] Siu S, Singh A. Design patterns for parallel computing using a network of processors[C]//Sixth IEEE International Symposium on High Performance Distributed Computing, Oregon, August 1997; 293-304
- [20] Romero P, Cox R, du Boulay B, et al. A survey of external representations employed in object-oriented programming environments[J]. Journal of visual languages and Computing, 2003 (14); 387-419
- [21] Vanneschi M. The programming model of ASSIST, an environment for parallel and distributed portable applications[J]. Parallel Computing, 2002, 28 (12); 1709-1732
- [22] Aldinucci M, Coppola M, Danelutto M, et al. High level grid programming with ASSIST[J]. Computational Methods in Science and Technology, 2006, 12(1); 21-32
- [23] Sun Jun-qing, Semiconductor M, Clara S. An Effective Execution Time Approximation Method for Parallel Computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23 (11); 1045-9219
- [24] Madduri K, Su J, Williams S, et al. Optimization of Parallel Particle-to-Grid Interpolation on Leading Multicore Platforms[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(10); 1045-9219
- [25] Gonz'alez-V'elez H, Leyton M. A survey of algorithmic skeleton frameworks high-level structured parallel programming enablers [J]. Softw. Pract. Exper, 2010, 40; 1135-1160
- [26] Góes L F W, Ioannou N, Xekalakis P, et al. Autotuning Skeleton-Driven Optimizations for Transactional Worklist Applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(12); 1045-9219
- [27] Scalosub G, Marbach P, Liebeherr J. Buffer Management for Aggregated Streaming Data with Packet Dependencies[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(3); 1045-9219
- [28] Nugteren C, Custers P, Corporaal H. Algorithmic Species: A Classification of Affine Loop Nests for Parallel Programming [J]. ACM Transactions on Architecture and Code Optimization, 2013, 40(4)
- [29] S'aez S, Salvadory I. Exploiting Parallelism in Multi-View Systems using UML Activity Diagrams and OpenMP[C]//2010 Workshops on Database and Expert Systems Applications, 2010
- [30] 王科特, 王力生. 信号实时采集的最佳并行线程数的研究[J]. 计算机应用, 2011, 31(10); 2593-2596
- [31] Peng Li-zhi, Yang Bo. A parallel evolving algorithm for flexible neural tree[J]. Parallel Computing, 2011, 37; 653-666
- [32] Ovatmana T, Weigertb T. Exploring implicit parallelism in class diagrams[J]. The Journal of Systems and Software, 2011, 84; 821-834
- [33] Yang Chao-tung, Huang Chih-lin. Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters[J]. Computer Physics Communications, 2011, 182; 266-269
- [34] Redondo J L, García I, Ortigosa P M. Parallel evolutionary algorithms based on shared memory programming approaches [J]. J Supercomput, 2011, 58; 270-279
- [35] Sui Yang-yi, Lin Jun, Zhang Xiao-tuo. An Automated Refactoring Tool for Dataflow Visual Programming Language[J]. ACM SIGPLAN Notices, 2008, 43(4)
- [36] Whitleya K N, Novick L R. Evidence in favor of visual representation for the dataflow paradigm; An experiment testing LabVIEW's comprehensibility[J]. Int. J. Human-Computer Studies, 2006, 64; 281-303
- [37] Tekinerdogan B, Aksit M. Impact of Evolution of Concerns in the Model-Driven Architecture Design Approach[J]. Electronic Notes in Theoretical Computer Science, 2007, 163; 45-64
- [38] Jazayeri M, Oberleitner J. Predicting Incompatibility of Transformations in Model-driven Development[J]. Electronic Notes in Theoretical Computer Science, 2005, 127; 129-137
- [39] Daniluk A. Visual modeling for scientific software architecture design. A practical approach[J]. Computer Physics Communications, 2012, 183; 213-230
- [40] Hloupisa G, Stavrakas I. WSN Open Source Development Platform: Application to Green Learning[J]. Procedia Engineering, 2011, 25; 1049-1052
- [41] Jordan H F, Alagband G. Fundamentals of Parallel Processing [M]. Chi Li-hua, Liu Jie, ed. Beijing: Tsinghua University Press, Oct. 2004
- [42] Herlihy M, Shavit N. The Art of Mutiprocessor Programming [M]. Jing Hai, Hu Kan, ed. Beijing : China Machine Press, Aug. 2009
- [43] Culler D E, Singh J P, Gupta A. Parallel Computer Architecture: A Hardware/Software Approach(Second Edition)[M]. Li Xiaoming, et al, ed. Beijing; China Machine Press, Oct. 2002
- [44] Giacaman N, Sinnen O. Parallel Task for Parallelising Object-Oriented Desktop Applications[J]. Int J Parallel Prog, 2013(41); 621-681
- [45] Vianna E, Comarela G. Analytical Performance Models for MapReduce Workloads[J]. Int J Parallel Prog, 2013(41); 495-525