高技术通讯 2013 年 第23 卷 第9 期:901 ~907

doi:10.3772/j.issn.1002-0470.2013.09.004

基于线程负载的嵌入式多核低功耗优化方法研究①

王 庆② 季振洲 朱素霞

(哈尔滨工业大学 计算机科学与技术学院 哈尔滨 150001)

摘 要 为了提高并行应用程序在片上多核系统上运行的高效能,提出了一种面向多线程负载不平衡的低功耗执行模型,并针对该模型提出了一种基于运行时信息反馈的线程执行频率控制算法,使得程序运行时可以根据并行线程的负载状况动态调节运行频率,在不影响并行程序运行性能的情况下,降低程序运行的能耗。在多核模拟器实验平台上实现了上述低功耗多线程执行模型及运行时算法。实验结果表明,这种新的低功耗多线程执行方法平均可节省 13% 的能量消耗,显著地降低了能耗和提高了程序的性能和功耗比。

关键词 负载不平衡,动态频率调节,低功耗,并行程序

0 引 言

随着集成电路物理极限的到来,仅仅通过提高处理器速度来提高计算能力已变得越来越困难。所以多核处理器通过增加处理器个数来满足不断增长的计算能力需求。片上多处理器(chip multiprocessor,CMP)是指在一个处理器芯片上集中两个或者多个内核。CMP 允许线程在多个处理器核上并行执行,从而利用线程级并行性来提高系统性能[1]。现行主流的基于共享内存的并行程序设计模型是基于锁和信号量同步的多线程模型,但使用这一模型编程容易出现共享数据的冲突和竞争,这些冲突和竞争往往难以在编程和编译阶段发现。因此,并行执行的多线程之间经常存在负载不平衡的现象,有些线程运行完分配自己的工作任务后,总是处于空闲或者忙等状态,而另外一些线程则一直处于工作状态。这样将造成系统资源的浪费。因此,为了支持并行程序在负载不平衡的情况下低功耗的高效能运行,本文结合动态电压频率调节(dynamic voltage and frequency scaling,DVFS)[2]技术,提出了一种基于片上多核负载不平衡的能耗优化执行模型,并针对该模型提出了一种基于运行时信息反馈的线程执行频率控制算法,使得运行时系统可以根据并行线程的负载不平衡性动态调节运行频率,在不影响并行程序运行性能的情况下,降低程序运行的能耗。模拟实验结果表明,所提出的低功耗多线程执行模型有效降低了并行程序的能耗,能更好地获得性能与功耗间的平衡。

1 背景和相关研究工作

在 CMP 系统结构中,大多数多线程并行程序采用共享内存结构的并行模型(比如 OpenMP[3])。该并行模型通常采用 fork-join 形式的执行方式。如图 1所示,在 fork-join 执行模式下,程序由主线程开始执行,当遇到并行区时,主线程启动(唤醒)其它工作线程并将并行区的计算分配给各个线程,在并行区结束时,工作线程终止(睡眠),继续交由主线程继续执行。通常,在并行程序中,该并行区和串行区交替出现。为了保持数据的完整和避免数据冲突,在一个并行区域结束时,所有并行执行的线程必须采取同步操作。图 1 给出了一个并行程序的结构并详细描述了一个并行区域的执行情况。

在并行程序多线程执行环境下,程序并行执行区的运行最慢的线程称为关键线程,它的运行时间为临界时间,如图 1 中所示,线程 3 是关键线程,它是最后进入障碍同步点的,其他运行比较快的线程

① 国家自然科学基金(61173024)资助项目。

② 男,1982 年生,博士生;研究方向:嵌入式多核,并行计算,计算机系统结构;联系人,E-mail:wangqing@hit.edu.cn

(收稿日期:2012-09-24)

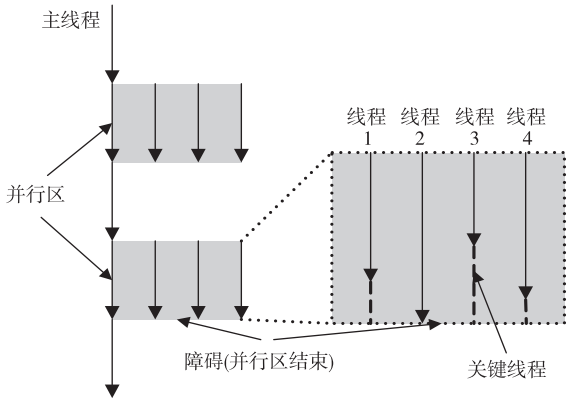

图 1 并行程序的执行结构

等待关键线程进入障碍同步。并行应用程序的多线程之间经常存在这种负载不平衡的现象,造成多核系统资源的浪费。因此,并行程序中负载不平衡的问题给如何充分利用硬件提供的并行处理能力从而使软件高效能运行带来了更多的挑战。

近年有大量并行程序负载平衡方面的研究工作。在解决应用程序负载不平衡提高性能方面,文献[4]提出一种通过任务或者数据重分配的动态负载平衡策略;Boneti[5,6]面向同时多线程(SMT)多核结构,利用硬件线程的优先级改变指令解码速度,使用重新分配硬件资源的负载平衡策略;DCP 策略[7-9]通过对片上多核的共享 cache 进行动态划分,实现多线程的负载平衡;文献[10]提出线程每条指令的平均时钟周期数(cyeles per instruction,CPI)值能准确反映线程负载状况,结合 DCP 策略,可在运行时实现单一多线程程序执行的负载平衡,提高程序性能。上述方法均只强调通过不同的动态负载平衡策略来提高程序运行性能,而且动态负载平衡策略会给系统带来更大的能量消耗,牺牲能耗来提高程序性能。本文针对并行程序出现的负载不平衡的问题,在不降低并行程序运行性能的情况下,通过运行时系统对线程进行降频操作,显著降低能耗,提高并行程序运行的高效能。

与本文工作密切相关的另一方面是基于动态电压频率调节(DVFS)的低功耗负载平衡策略。Jian[11]扩展了在并行程序中的障碍同步时的不平衡性,实现了 thrifty barrier 障碍同步技术,通过软硬结合的方式动态预测处理器障碍同步的空闲时间。Chun[12]在文献[11]的工作基础上进行了扩展,提出了一种基于 barrier 的负载不平衡的能耗优化方法,该方法是基于 barrier 障碍同步点来预测线程的空闲程度,没有考虑 barrier 内的多线程负载情况。文献[13]设计 Meeting Points 在程序运行时动态地检测线程负载状况,提出了线程延时和线程负载平衡技术进一步指导并行程序降低功耗及提高性能,但是该策略只考虑基于循环并行的应用程序(如 OpenMP),没有讨论其它并行应用的情况。文献[14]通过大量的实验分析了如何更准确地反映多线程负载情况,提出了用 TCP 来指导 Intel 线程构建模块(TBB)并行应用程序的 Work-stealing 策略,并结合 DVFS 技术实现并行程序的低功耗运行。

文献[15]从低功耗编译优化技术的角度,提出基于编译指导的动态电压调节算法,结合可降频运行的线程的计算模型和降频因子的计算,实现低功耗编译优化算法;文献[16]基于语音语法树的 DVFS 低功耗算法,通过在程序内部自动插入电压调节代码来实现电压调节。然而,通过编译指导的低功耗优化方法只能发掘程序内部的特性来降低运行功耗,但是在并行程序运行过程中,由于数据冲突或者竞争等不确定因素,编译指导的方法无法动态检测多线程的负载状况,不能够进一步指导并行应用程序降低功耗。

2 高效能优化模型

本文提出了面向多线程负载不平衡的并行多线程的高效能执行模型来实现程序运行的高效能,降低功耗,提高性能和功耗比。本节从能量消耗的角度分析动态电压/频率调节对并行程序能耗的影响,并且进一步提出面向多线程负载不平衡的低功耗执行模型。

2.1 能耗分析

当前计算机系统主要采用 CMOS 技术。CMOS 电路的功耗和电压成平方关系,和频率成线性关系,通过降低程序运行期间的电压和频率能够有效降低系统功耗。一个程序的能量消耗(E)可以通过以下公式来估计[17]:

$$E = c \times V^2 \times f \times T \tag{1}$$

其中 T 是总的执行时间,f 是时钟频率,c 是容量交换效率,V 是提供的电压。在 CMOS 电路系统中,频率 f 和电压 V 可以近似为线性关系[18]:$f = V/\beta$,β 为一个常数。因此,本文中只讨论能耗与频率的关系。程序的执行周期数为 W,执行时间 $T = W/f$。因此当系统执行频率为 f_1、执行时间为 T_1 时,能量消耗为

$$E = c \times V^2 \times f_1 \times T_1 = c \times \beta^2 \times f_1^3 \times T_1$$

$$= c \times \beta^2 \times f_1^2 \times W \quad (2)$$

假定在程序的一个并行执行区域,关键线程的执行频率是最大的,为 f_{max},它在到达障碍同步点的执行时间为 t。其中一个非关键线程也以最大的执行频率,执行时间为 $0.8t$,假设该非关键线程在到达障碍同步时进入睡眠状态,在这个 $0.2t$ 时间内没有能量消耗。该非关键线程的能量消耗为

$$E_{f_{max}} = c \times \beta^2 \times f_{max}^3 \times 0.8t \quad (3)$$

引入动态电压频率调整后,为了保持该非关键线程不影响整体的程序执行时间,使得该线程与关键线程同一时间到达障碍同步点,将执行频率降到 $0.8f_{max}$(电压和频率满足线性关系,只考虑频率变化),能量消耗变为

$$E_{f_{scaled}} = c \times \beta^2 \times (0.8f_{max})^3 \times t = 0.64E_{f_{max}} \quad (4)$$

在上述的假设情况下能量消耗变化中,通过对非关键线程的动态电压频率调整可以显著降低 CPU 能量消耗。

2.2　低功耗执行模型

对于大多数线程来说,同步的大部分时间都用在等待关键线程到达同步点。先到达的线程进入等待状态,等待最后一个线程到达。处于等待状态的线程都不需要进行复杂的操作,可以将其占用的处理器电压或者频率降低,从而减少能量消耗。

对上述减少能量消耗的方法进行扩展,加入动态电压频率在线调整功能:并行线程运行一段时间后,在到达障碍同步点前,动态检测每个线程的负载性能情况,根据负载状况对每个线程进行电压频率控制,使得运行比关键线程快的线程有较低的电压和频率,关键线程仍然保持最高的电压和频率。每个线程尽量保持在同一时间内到达障碍同步点,因此在保证程序性能不变的情况下(执行时间没变),通过降低其他线程的电压和频率,减少能量消耗,如图 2 所示。

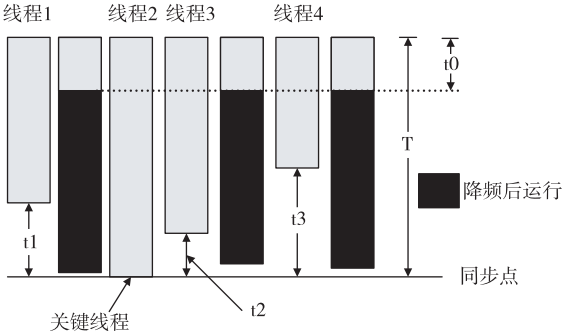

图 2　低功耗执行模型

本文给出基于程序并行区的功耗优化研究,定义并行线程降频因子 α_t 为并行线程 t 降频后的执行频率与正常的频率比值。假设程序的并行区有 4 个线程执行,如图 2 所示,线程 2 为关键线程,由于负载不平衡,线程 1、线程 3 和线程 4 分别有 $t1$、$t2$ 和 $t3$ 时间等待线程 2 进入同步点。

不采用动态电压频率调整的情况下,每个线程执行上面的过程中消耗的能量相同,为

$$E_{orig} = c \times \beta^2 \times f_{max}^3 \times T \quad (5)$$

如图 2 所示,在 $t0$ 时刻,为了讨论的方便,定义该时刻为并行程序执行过程各线程负载状况稳定但远小于 T 的某一个时刻,根据每个线程的负载情况对执行线程进行降频,使得每个线程的执行时间尽量接近关键线程,程序的执行性能保持不变。当采用这种延时降频执行模型后,由于 $t_0 > 0$,总的消耗的能量可以表示为

$$E_{delay} < c \times \beta^2 \times f_{max}^3 \times T(1 + \alpha_1^3 + \alpha_3^3 + \alpha_4^3) \quad (6)$$

因为 $0 < \alpha_t \leqslant 1$,所以优化后的能耗 $E_{delay} < 4E_{orig}$。采用线程延时降频后,程序的并行执行部分的总的能耗降低。同理扩展到有 n 个线程并行执行的情况下,降频后的能量消耗比率可以近似表示为

$$E_{delay}/E_{all} \approx (1 + \sum_{i}^{n-1} a_i^3)/n \quad (7)$$

因此,在片上多核系统中,针对多线程执行负载不平衡的状况,在保持程序性能不变的条件下,通过计算每个执行线程的降频因子,可以得出高效能优化执行后的能量消耗。

3　并行程序高效能优化的实现

在并行程序运行时对执行线程动态频率调节需要动态估计每个线程的负载情况,并且根据这些负载情况计算每个线程的降频因子。本节将给出所实现的多线程低功耗执行模型。

3.1　运行时系统设计框架

为了支持并行程序多线程高效能执行模型以及程序的运行时能耗自适应优化,本文采用了基于并行程序负载不平衡的动态检测运行时系统和具有支持 DVFS 的多核系统结构相结合的设计方案,如图3所示。动态检测线程负载不平衡的性能检测器(CPI Monitor)从底层处理器硬件计数器得到 CPI 值,运行时系统根据这些负载信息计算出每个线程的降频因子,通过线程频率调整降低程序运行能耗。

本文中的能耗优化执行模型是基于共享 cache 或者共享内存架构的片上多核处理器,运行时系统根据多线程并行执行的负载情况动态地调整线程执行频率,线程在第一次调整频率后,通过循环检测的方式实时统计线程负载情况变化,避免出现由于降低线程执行频率影响并行程序性能的情况。

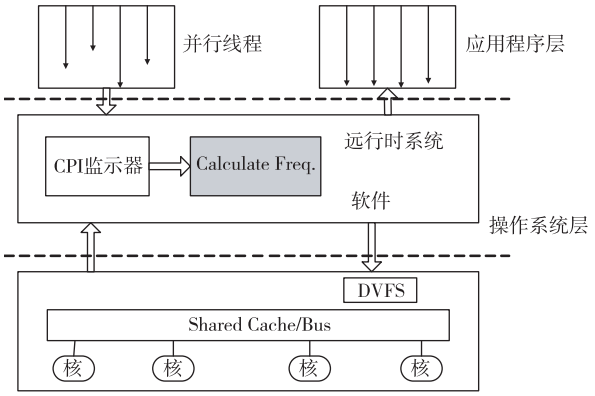

图 3 能耗优化的运行时系统设计

运行时系统在 Solaris 系统中进行了实现,为了能够正确对处理器核进行降频操作,需要对程序并行执行的线程和处理器核进行绑定(CPU affinity)。每个线程在并行程序执行过程中绑定到指定的处理器核上,使得执行线程实现对每个处理器实际运行频率的控制。

3.2 能耗优化算法

在程序并行执行的初始阶段,需要设置并行执行的每个线程的初始运行频率,在程序运行时根据线程的负载情况对每个线程进行降频操作,满足程序完成的临界时间不变。但是进行动态电压频率调节会导致线程执行时间随频率变化而改变,某个线程可能会出现超过临界时间而使得程序的并行性能下降。文献[10]研究了基于线程 CPI 值的 cache 动态划分,提出 CPI 能够很好地反映线程负载。本文基于该文献的部分工作,使用 CPI 来评估线程的负载情况。如果线程的 CPI 值最大,说明该线程负载状况最差,即是关键线程,运行快的线程具有小的 CPI 值。因此通过每个线程 CPI 与最大 CPI 值的比较估计线程降低的频率,采用动态性能检测方法得到每个线程的降频因子。

能耗优化算法根据在程序运行过程中对每个线程的性能参数 CPI 来计算各个线程的运行频率。图 4 中的算法 EnergyAwareOptimization 在程序的并行初始阶段为每个线程分配最大的执行频率 f_{max}。然后在并行程序的性能检查点,统计每个线程的性能参数 CPI_i,并根据这些参数对那些非关键线程进行降频操作。这一过程一直重复到检查点结束为止。在每个检查点,算法能够识别多线程负载的不平衡,控制可降频运行的线程并计算每个线程的降频因子。

```
EnergyAwareOptimization
1.  Initialization;
    Init Thr;//线程负载状况的阈值.
    init_freq( f_max );
2.  Point:At the end of each checkpoint(interval);
    /* 记录每个线程 CPI 值, CPI_i */
    Record_CPI( CPI_i );
    /* 根据每个线程的 CPI 值,设置线程执行的频率 */
3.    if ( CPI_max / CPI_min > Thr ){      //负载状况判断
4.        Reset_freq( CPI_i ,f_max );}
5.      else goto Point;
6.  goto Point;
7.  end
```

图 4 能耗优化算法

为了更准确地计算每个线程的降频因子,函数 Reset_freq 根据每个线程的负载情况计算线程的降频因子。本文提出的低功耗执行模型是假定线程的执行频率可以在一个极大值 f_{max} 和一个极小值 f_{min} 之间连续变化,实际的处理器只需根据线程的负载情况(CPI)设定合适的电压频率层次。线程的降频因子计算方法如下:

(1)获取每个线程的 CPI_i 值,得到 CPI_{max}。

(2)计算每个线程的降频后的执行频率 f_i:$f_i = \frac{CPI_i}{CPI_{max}} \times f_{max}$。

(3)根据处理器预先设定的电压频率档位,选择合适的电压频率层次,使得每个线程执行的频率为最接近 f_i。

4 实验结果及分析

本节将给出本文所实现的程序并行区能耗优化的原型,并结合在其上所开发的能耗优化控制运行时系统来验证能耗优化的有效性。本文提出的高效能优化执行模型在我们更改过的 Simics/GEMS[19] 模拟器上进行了实验验证。Simics 是一款高性能的系统模拟器,它提供了一个受控制的、确定性的完全

虚拟环境模拟平台,它可以模拟单处理机和多处理机系统。为了能够更准确地分析程序并行执行性能和能耗,结合 Wattch 功耗模型对 GEMS 的时序模块进行了扩展。实验平台模拟了一个基于共享存储的片上多核处理器,多核结构具有二级共享 cache。其中指令和数据一级 cache 大小为 16KB,访问延迟为 2 拍,二级共享 cache 大小为 1MB,访问延迟为 12 拍。表 1 详细列出了相关配置参数。

表 1　模拟的片上多核系统的参数

模拟参数	Default Value
处理器核	4
指令集	Sparc v8
缓存行	32 bytes
一级指令缓存	16KB, 4-way set associative, 2-cycle hit time
一级数据缓存	16KB, 4-way set associative, 2-cycle hit time
二级 cache	Shared,1MB,12-cycle hit time
置换策略	Strict LRU

借鉴文献[20]的方法,本文在 Simics 模拟器中加入了 Wattch[21] 功耗模型,对于程序在实际运行中,统计线程在其所属的时钟域内各周期的活动情况来计算能耗。实验中模拟多核处理器的执行频率设置从 100MHz 到 1000MHz,每 100MHz 为一档。需要说明的是,在模拟器给出的模拟低功耗优化执行的实验验证过程中,没有考虑电压以及频率变换带来的额外开销,但作为低功耗执行优化方法的研究,本文重点关注的是在程序运行时中,根据线程负载情况对线程降频优化后功耗的变化和性能变化之间的关系。

实验采用的应用程序实例来自 SPLASH－2[22] 基准测试程序,如表 2 所示。本文在模拟的实验平台上统计所选用的实验测试程序不平衡情况,表 2 给出了实验用例程序的多线程负载不平衡比率(Imbalance Percentage,IP)[23]。它反映了程序中的并行线程负载性能,比率越高的说明程序的负载不平衡性越严重。不平衡比率 IP 的定义如下(其中 N 代表线程数):

$$IP = 100 \times \frac{\text{最长执行时间} - \text{平均执行时间}}{\text{最长执行时间}} \times \frac{N}{N-1} \tag{8}$$

表 2　实验用例的测试程序

测试集	测试大小	不平衡比率(%)
LU	1024 * 1024matrix, 32 * 32 blocks	19.53%
FFT	2 * *24 data points	24.27%
Cholesky	tk29.0	18.54%
Radix	262144 integers	16.59%
Ocean	514 * 514	7.21%
Water-nsquared	4096 molecules, 8time steps	13.31%
Water-spatial	4096 molecules, 8 time steps	9.67%
FMM	16384 particles, 8 time steps	27.45%
Barnes	16384 particles, 8 time steps	21.58%
Volrend	Head.den	56.48%
Radiosity	-ae 5000.0 -room -en 0.05 -bf 0.01	5.14%

从表 2 中的数据可以看出,SPLASH－2 并行基准测试程序在实际运行中存在多线程负载不平衡性,其中 Volrend 的不平衡比率达到了 56.48%。因此,通过实现高效能低功耗的优化执行模型可以避免由于应用程序的负载不平衡导致的系统资源的浪费,节省程序运行中的能量消耗。

图 5 给出了高效能低功耗优化执行后的测试结果,所有的结果相对于没有能耗优化的结果进行了归一化,Orin 是不进行优化的结果,Opt 是能耗优化的结果。从结果可以看出,本文提出的面对多线程负载不平衡的能耗优化执行模型产生了 5% ~22% 的能量节省,volrend 取得了最好的节能效果,达到了 22%。

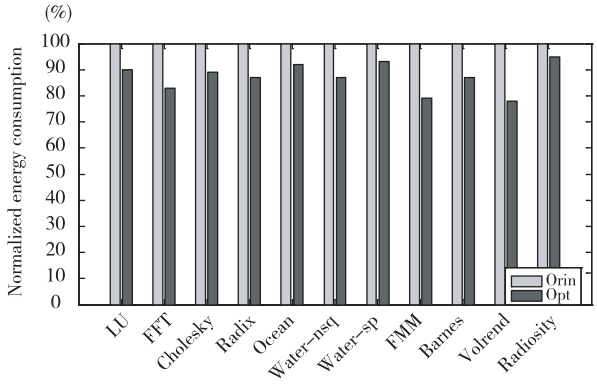

图 5　能耗优化的实验结果对比

图 6 是进行能耗优化后对程序运行性能的影响,从图中结果可以看出,实现的低功耗执行模型对实际程序运行造成的性能影响很小,平均只有2.2%的性能损失。

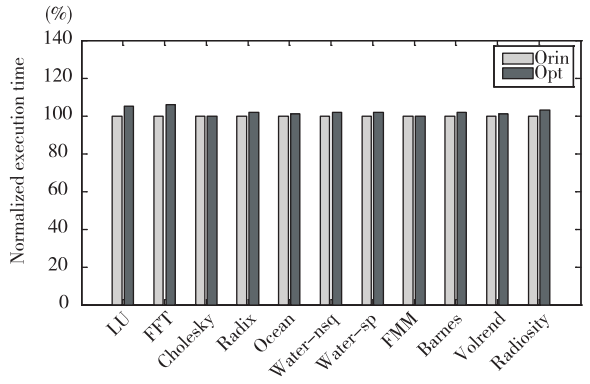

图 6 程序性能的实验结果对比

5 结 论

在片上多核系统中,并行程序中的多线程负载平衡状况直接关系到程序运行性能和系统的能耗。为了并行应用程序在片上多核系统上的高效能的运行,本文研究和探讨了在面向共享内存的并行应用程序中,针对多线程负载不平衡的问题,提出了一种基于片上多核负载不平衡的高效能优化执行模型,并针对该模型提出了一种基于运行时信息反馈的线程执行频率控制算法,使得程序运行时可以根据并行线程的负载状况动态调节线程运行频率,在不影响并行程序运行性能的情况下,降低程序运行的能耗,提高了并行程序的高效能。

实验结果表明,在实际的并行应用程序中,通过在程序运行时动态调整多线程执行频率,实现并行程序高效能优化执行,可以显著降低程序的能量消耗,提高程序的性能和功耗比。本文提出的低功耗高效能多线程执行模型在只造成 2.2% 的性能损失的情况下,最大可节省 22% 的能量消耗,平均可节省 13% 的能量消耗。

参考文献

[1] Chauhry S, Caprioli P, Yip S, et al. High-performance throughput compuing. *IEEE Micro*, 2005, 25(3): 32-45

[2] Chandrakasan A, Sheng S, Brodersen R. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 1992, 27(4): 473-484

[3] Dagum L, Menon R. OpenMP: an industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 1998, 5(1): 46-55

[4] Schloegel K, Karypis G, Kumar V. Parallel multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of 6th International Euro-Par Conference on Parallel Processing, Munich, Germany, 2000. 296-310

[5] Boneti C, Gioiosa R, Cazorla F J, et al. Balancing HPC applications through smart allocation of resources in multithreaded processors. In: Proceedings of the IEEE 22th International Symposium on Parallel and Distributed Processing, Miami, USA, 2008. 1-12

[6] Boneti C, Gioiosa R, Cazorla F J, et al. A dynamic scheduler for balancing HPC applications。In: Proceedings of the ACM/IEEE Conference on High Performance Computing, Austin, USA, 2008. 1-12

[7] Suh G E, Devadas S, Rudolph L. A new memory monitoring scheme for memory-aware scheduling and partitioning. In: Proceedings of the 8th International Symposium on High-Performance Computer Architecture, California, USA, 2002. 117-128

[8] Kim S, Chandra D, Solihin Y. Fair cache sharing and partitioning in a chip multiprocessor architecture. In: Proceedings of 13th International Conference on Parallel Architecture and Compilation Techniques, France, 2004. 111-122

[9] Qureshi M K, Patt Y N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches。In: Proceedings of 39th Annual IEEE/ACM International Symposium on Microarchitecture, Florida, USA, 2006. 423-432

[10] Muralidhara S P, Kandemir M, Raghavan P. Intra-application cache partitioning. In: Proceedings of 24th IEEE International Symposium on Parallel and Distributed Processing, Atlanta, USA, 2010. 1-12

[11] Li J, Martinez J, Huang M. The thrifty barrier: Energy-efficient synchronization in shared-memory multiprocessors. In: Proceedings of the 10th International Symposium on High Performance Computer Architecture, Madrid, Spain, 2004. 14-23

[12] Liu C, Sivasubramaniam A, Kandemir M, et al. Exploiting barriers to optimize power consumption of CMPs. In: Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, 2005. 5a-5a

[13] Cai Q, González J, Rakvic R, et al. Meeting points: using thread criticality to adapt multicore hardware to parallel regions. In: Proceedings of the 17th International Confer-

ence on Parallel Architecture and Compilation Techniques, Ontario, Canada, 2008. 240-249

[14] Abhishek B, Margaret M. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors. In: Proceedings of 36th International Symposium on Computer Architecture, Texas, USA, 2009. 290-301

[15] 赵荣彩,唐志敏,张兆庆等. 低功耗多线程编译优化技术. 软件学报,2002,13(6):1123-1129

[16] 易会战,陈娟,杨学军等. 基于语法树的实时动态电压调节低功耗算法. 软件学报,2005,16(10):1726-1734

[17] Hopfield J J. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 1985, 52(1): 141-152

[18] 黄春,易会战,杨学军. 面向 OpenMP 的能耗优化技术. 计算机工程与科学,2008,30(2):151-155

[19] Martin M. Multifacet's general execution-driven multiprocessor simulator (GEMS) Toolset. *SIGARCH Computer Architecture News*, 2005, 33(4): 92-99。

[20] Chen J W, Dubois M, Stenstrom P. SimWattch and learn. *Potentials, IEEE*, 2009, 28(1): 17-23

[21] Brooks D, Tiwari V, Martonosi, M. Wattch: a framework for architectural-level power analysis and optimizations. In: Proceedings of the 27th Annual International Symposium on Computer Architecture, British Columbia, Canada, 2000. 83-94

[22] Woo S C, Ohara M, Torrie E, et al. The SPLASH-2 programs: Characterization and methodological considerations. In: Proceedings of the 22nd International Symposium on Computer Architecture, Santa, Margherita Ligure, Italy, 1995. 24-36

[23] Rose L D, Homer B, Johnson D. Detecting application load imbalance on high end massively parallel systems. In: Proceedings of 13th International Euro-Par Conference on Parallel Processing, 2007. 150-159

Study of a low power optimization method based on multithreaded load imbalance for embedded multicore

Wang Qing, Ji Zhenzhou, Zhu Suxia

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

Abstract

In order to improve the energy efficiency of parallel application programs on chip multiprocessor systems, a high-performance low-power execution model is proposed for load imbalance in multi-threaded programs. Then, according to the model, an algorithm for control of parallel threads' execution frequency based on the run-time feedback scheme is presented to realize dynamical scaling of the execution frequency based on the load imbalance information of parallel threads to decrease energy consumption without affecting the performance of the parallel programs. The model and the algorithm were implemented on a multicore simulator. The experimental results show that the proposed model can save on average 13% of energy consumption, significantly reduce the energy consumption and improve the performance and power ratio.

Key words: load imbalance, dynamic frequency scaling, low power, parallel programs