温度感知的 Linux 多核调度算法研究

吴国伟,徐子川,姚 琳
WU Guo-wei,XU Zi-chuan,YAO Lin

大连理工大学 软件学院,辽宁 大连 116621
School of Software,Dalian University of Technology,Dalian,Liaoning 116621,China
E-mail:wgwdut@dlut.edu.cn

WU Guo-wei,XU Zi-chuan,YAO Lin.Research of thermal aware multi-core Linux scheduling algorithm.Computer Engineering and Applications,2009,45(24):72-76.

Abstract: Temperature rise of the multi-core processor is harmful for the processor's performance.DTM(Dynamic Thermal Management) mechanism is a hardware level thermal management method with a great loss of processor's performance.Thermal dynamic management in the operating system level called thermal aware scheduling is proposed.It can get the value of performance counter online and then calculate the core's temperature which can be used to decide which core to run a new process.Also,a heuristic method called MST is proposed.Based on ATMI,the simulation results show that the algorithm can create a uniform power density map than the algorithm without thermal awareness,and reduce the number of process migration by using MST heuristic.

Key words: thermal-aware;multi-core;Linux scheduling algorithm;dynamic priority;thread migration

摘 要: 多核处理器温度升高会影响芯片的稳定性和性能的发挥,硬件层面的 DTM(Dynamic Thermal Management)方法以牺牲处理器性能为代价来降低功耗,提出了在一种软件层面的温度感知调度算法,它可以在线实时获取处理器性能计数器的值并计算各个执行核温度,根据各执行核的温度状况在各个核上合理分配进程,给出了温度感知的启发式方法。基于 ATMI 温度仿真器的仿真表明,温度感知调度算法较无温度感知的算法可以创建更均匀的功率密度图,且带 MST 启发式方法的温度感知调度算法能明显减少进程的迁移次数。

关键词: 温度感知;多核;Linux 调度算法;动态优先级;进程迁移
DOI: 10.3778/j.issn.1002-8331.2009.24.023 **文章编号:** 1002-8331(2009)24-0072-05 **文献标识码:** A **中图分类号:** TP316.89

1 引言

随着纳米技术的发展, 处理器中晶体管的集成度越来越高。功率密度以及温度成为影响处理器性能和寿命[1-2]的主要原因。多核技术成为当今高端处理器的主要发展方向,温度问题也越来越明显。处理器温度的提高影响了芯片的稳定性和性能的发挥,处理器打包、散热技术虽然很大程度上缓解了这个问题,但是这些技术产生的降温能力和成本不成正比[3],于是有了硬件层次的 DTM 和 Clock Gating 技术来协助降低处理器温度,当处理器的温度超过阈值时 DTM 降低处理器的频率,Clock Gating 则暂时停止指令的执行,从而高效地达到降温的目的,但是这些方法是以处理器性能部分丢失为代价的[4]。而软件层面的 DTM 机制由于小性能损失以及低成本成为研究的热点。

该文研究一种操作系统层面的多核温度管理机制——温度感知调度。Michaud 等人[4-6]提出的温度感知调度算法主要关注执行核之间的线程迁移,对单个执行核内部没有做过多的考虑。Stavrou 等人[7]提出的温度感知调度算法适合于 CMP[8]处理器架构。Christiana[9]的温度感知调度器提出了进程的温度贡献值的概念,但是并没有提出获取温度进程贡献值的方法,另外进程本身的属性,以及进程的执行序列对执行核的温度有很大的影响。该文兼顾了进程本身属性, 主要做了以下几点工作:(1)提出了在操作系统层面的温度感知进程分配调度策略,根据"最冷"与"邻居"优先相结合的策略将进程分配到执行核上,显著减小了线程迁移带来的性能损失。(2)在线实时获取性能计数器的值并给出计算各个执行核温度的方法、温度感知进程调度算法及相关的启发式方法。(3)针对一般非实时进程,提出一种根据进程温度贡献值动态调整优先级的机制,温度贡献值低的进程的优先级高于温度贡献值高的进程,在不影响系统性能前提下,很好地保证了系统的吞吐量。(4)对分析的结果进行仿真验证,得出该文的温度感知调度算法较无温度感知的算法以及 Merkel 等人[6]的温度感知算法可以创建更均匀的功率密度图,进而提高了处理器的稳定性和寿命[2]。

2 温度感知的多核调度算法

温度感知多核调度算法不同于硬件层面的 DTM(Dynamic Thermal Management)机制,其旨在不影响系统吞吐量前提下

作者简介: 吴国伟(1973-),男,副教授,主研方向:嵌入式系统、容错计算等;徐子川(1986-),男,硕士生,主研方向:嵌入式系统、容错计算等。
收稿日期: 2008-07-11 **修回日期:** 2008-10-27

解决多核处理器面临的温度相关问题,得到更统一的功率密度分布图,提高处理器的性能和寿命。在 SMP 调度算法中,进程可以在任何一个处理器上运行,但由于处理器的亲和力[6]是衡量算法的一个标准,即如果一个进程在某一个 CPU 上运行了一定的时间,无论数据还是指令缓存都存在这个进程的数据。如果此时,将进程迁移到其他的 CPU 上执行,就会大大减少缓存的命中率。因此,文中的多核调度算法考虑了 CPU 的亲和力。首先提出了进程分配策略,将新产生的进程分配到合适的执行核上执行,然后调度器为每一个执行核提供一个本地的运行队列,同样也就需要一个负载平衡器来协调各个执行核之间的负载。这里主要考虑了进程的调度策略和调度算法。

不同的进程通过不同的方式使用处理器的不同单元,因此产生的功耗也就不同。每个进程的行为以及进程的执行顺序[10]决定了处理器的功耗变化,故而每个进程对执行核功耗贡献值也就不同。算法实时计算每一个进程的功耗贡献值,并根据 2.1 节中的进程分配策略来确定将新产生的进程分配到哪个执行核上。在进程调度过程中,根据进程的温度贡献值动态改变进程的优先级,如果执行核超过温度阈值,则启动负载平衡算法,来平衡执行核之间的温度差异,以创建均衡的功率密度分布图。该调度器的结构如图 1。

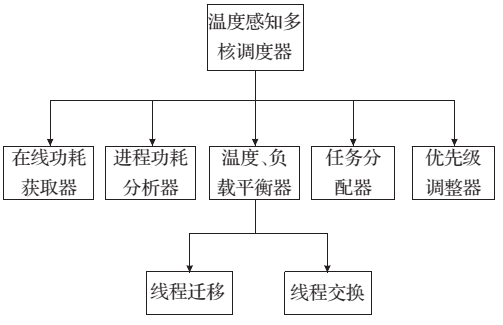


图 1　温度感知多核调度器结构

其中在线功耗获取器负责读取执行核的性能计数器的值,然后传递给进程功耗分析器,进程功耗分析器将性能计数器中的值转换为进程的温度贡献值;任务分配器负责在新进程产生时将进程分配到合适的执行核上;温度、负载平衡器是 Linux 负载平衡器和温度平衡器的结合,在考虑执行核之间负载平衡的基础上考虑了温度的平衡,当有一个平衡被破坏时,算法根据当前执行核的温度、负载情况来判断是进行线程迁移还是进行线程交换;优先级调整器完成对不同温度贡献值进程优先级的调整。

2.1　进程分配策略

每一个执行核拥有一个运行队列,调度器可以提供不同的进程分配策略到不同的核,温度感知调度器考虑以下几种任务分配策略:

(1)随机:新建进程或者重新就绪进程将随机选择执行核来执行。当所有的执行核的功率密度相近的时候,全部被高度利用或者全部空闲,这个分配策略是可行的。

(2)最"冷":进程将选择最"冷"的执行核来运行。这种方法唯一需要知道当前所有执行核的温度情况,提出了利用性能计数器来计算执行核的温度,获取执行核温度情况的方法。

(3)"邻居"优先:这种启发式方法利用了处理器内部各个执行核之间的热交换原理。一些在处理器边缘的执行核能通过散热片很好地释放自己的能量,调度器标识这些核,当线程迁移发生时,调度器查看周围的执行核是否被调度器标识,如果被标识则选择被标识的执行核为迁移线程的目标核。显然这个分配策略适合执行核数目较多的多核处理器。

(4)"邻居"优先和最"冷":是最"冷"分配策略和"邻居"优先分配策略的结合。当线程迁移需要发生时,如果源执行核周围有一个以上被调度器标识的目标执行核,那么调度器从中选择一个最"冷"的执行核作为线程迁移的目标核,另外如果对处于处理器边缘的执行核采用"邻居"优先,则对处理器中心部分的执行核采用"最冷"策略。

文中的温度感知多核调度器采用了随机以及"邻居"优先和最"冷"相结合的分配策略。

2.2　进程功耗贡献值

算法通过提供进程功耗贡献值来使操作系统调度器具有温度感知能力。

当前处理器采用温度传感器来获取处理器某一个单元的温度值,这种方法获取的温度值精确,参考价值高,由于温度传感器的个数有限以及传感器本身的缺陷,这种方法无法高效地实时在线获取执行核的温度值。因此,选择了基于性能计数器[11]的获取温度的方法来间接计算执行核的温度和功耗。

性能计数器可以用来对处理器运行时的内部事件计数,为优化处理器性能提供数据。算法假设每一种事件产生的功耗是不变的,再通过读取一种或者多种事件的计数来获取当前执行核的功耗来计算每一个执行核的功耗。单一执行核的功耗公式如下:

$$E_i=\sum_{j=1}^{n} a_j \cdot c_j (i=1,2,\cdots) \tag{1}$$

其中,a_j 是事件 j 产生的功耗,c_j 是事件 j 对应性能计数器的值。i 执行核索引。a_j 的获取办法[12]:进程的不同行为会对处理器带来不同的功耗,这里假设性能计数器记录的每一种事件发生产生的功耗是不变的,利用文[12]获取功耗方法可以得到如下不同处理器事件产生的功耗:

表 1　不同事件温度贡献值 a_j

事件	功耗/W
time stamp counter	11.23
unhalted cycles	14.53
retired branches	26.79

分时操作系统将处理器资源以时间片的形式分配给进程,根据公式(1)和表 1 可以得出几百个时间片内进程的功耗,然后对进程下一个时间片的功耗做预测,以预测值作为进程的功耗贡献值。算法采用了经济学中的指数加权平均法来预测进程下一个时间片的功耗值:

$$Contr_i=p \cdot E_i+(1-p) \cdot Contr_{i-1} \tag{2}$$

其中,$Contr_i$ 为当前时间片进程功率贡献值,E_i 为当前时间片周期内通过公式(1)计算出的功耗,$Contr_{i-1}$ 是上一个采样周期进程功率贡献值,p 为权重。根据指数加权平均的概念,可以得出以前采样单元的功耗值对当前采样单元功耗值的影响度示意图(图 2)。

由图 2 可以得出,这种方法可以有效预测下一个周期的功耗,并且可以很好地避免由于频繁更改进程贡献值带来的震荡。在第 3 章对算法的仿真中,采用了公式(2)来计算进程的温度贡献值。

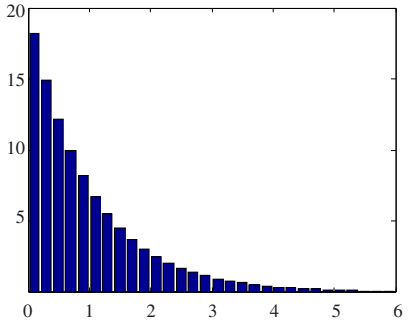


图2 指数加权平均

2.3 动态改变进程优先级

对于普通非实时进程，采用基于优先级的调度策略来调度。优先级高的进程先运行，获得的时间片自然就高，低的后运行，相同优先级的进程按轮转方式进行调度。进程的调度过程中可以根据进程贡献值的不同来动态改变进程的优先级，对于温度贡献值较大的进程获得优先级较低，温度贡献值小的优先级较高。这样可以很大程度上保证系统的吞吐量。Linux 通过向每个进程提供一个 nice 值来实现动态优先级，借助这种思想，通过动态改变 nice 值实习了普通进程优先级的改变。nice 值的计算公式如下：

$$N_{dyn}=\frac{(C-C_{\min})\times(N_{\max}-N_{\min})}{C_{\max}-C_{\min}}+N_{\min} \tag{3}$$

其中，N_{dyn} 代表动态计算的 nice 值，C 为进程当前的温度贡献值，$C_{\max}$ 和 $C_{\min}$ 分别表示所有进程贡献值的最大值和最小值，它们是在调度器启用前提前确定的，$N_{\max}$ 和 $N_{\min}$ 分别代表 nice 值的最大值和最小值，在 Linux 操作系统中这两个值分别为-20和19[13]。

2.4 线程迁移

当执行核的温度达到阈值时，算法将当前执行核标识为调度域内最热执行核，并触发负载平衡程序，将当前运行队列中温度贡献值最大的进程 $P_{hottest,source}$ 迁移到同一个调度域内最冷的执行核 $Core_{coolest}$ 上去。为了减少线程迁移的次数，从而增加执行核的亲和力[7]以及线程迁移带来的性能损失，算法规定目标执行核功耗 E_d 至少比当前执行核的功耗 E_s 小 ψ，即：$E_s-E_d\geqslant\psi$。并且只有执行核 $Core_{coolest}$ 处于空闲状态时才将当前进程 $P_{hottest,source}$ 迁移到执行核 $Core_{coolest}$ 上。如果 $Core_{coolest}$ 没有空闲，则选择 $Core_{coolest}$ 上“最冷”的进程 $P_{coolest,destination}$ 与执行核 $Core_{hottest}$ 上进程 $P_{hottest,source}$ 互换。迁移算法如图3。

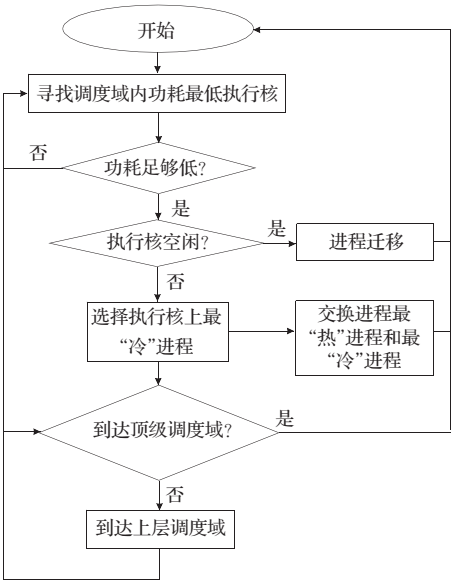


图3 线程迁移算法

算法使用 MST(Maximum Scheduling Threshold)启发式方法[7]确定线程迁移的目标核 $Core_{destination}$ 是否足够“冷”。MST 启发式方法不允许将 $P_{hottest,source}$ 迁移到即将达到阈值的执行核上，从而降低了迁移频率。

3 仿真实验及结果分析

3.1 功率模型和温度模型

采用以下功率模型：

$$PowerDensity_i=p_s+\beta_\mu\times E+\beta_c\times f \tag{4}$$

其中 p_s 是静态功率密度，$\beta_\mu\times E$ 是对用于有用功的动态功率密度，$\beta_c\times f$ 是时钟电路产生的值，参数 p_s、β_μ 和 β_c 取决于电压。考虑恒定电压，因此 p_s、β_μ 和 β_c 也保持恒定。由时钟电路产生的 $\beta_c\times f$ 值对于当前处理器是重要的。例如，在英特尔 Itanium Montecito 上，大约25%的功耗都源于时钟电路[14]。时钟门控技术能够缓和这一问题，但是不能完全解决[15]。因此模型主要考虑了 $\beta_\mu\times E$ 部分的功耗，即由于调度不同的进程以及不同的进程序列产生的功耗。模型的初始状态假设有一个执行核动态功率密度 $\beta_\mu\times E$ 比较低。实际上，由于现在处理器都实现了 DTM 机制，如果一个执行核长期处于空闲状态下，把动态功率密度 $\beta_\mu\times E$ 降到一个比较低的值是有可能的[5]。时钟频率、功率密度参数以及温度最大值在表2中给出。

表2 时钟频率、功率密度和温度最大值

f	2.31×10^9
p_s	0.5 W/mm^2
$\beta_c\times f$	0.5 W/mm^2
β_μ	2×10^{-4} J/m^2
$T_{\max}$	85℃

模型的所有温度的获取都是通过 ATMI 仿真器来实现的，ATMI(http://www.irisa.fr/caps/projects/ATMI/)是一个用 C 语言实现的对处理器温度分析的模型，输入功耗返回处理器的温度。仿真中用到的 ATMI 参数如表3：

表3 ATMI 模型参数

参数	单位	意义	值
z_1	m	layer1 厚度	0.000 5
$d=z_2-z_1$	m	layer2 厚度	0.005
k_1	W/mK	layer1 温度传导率	109
k_2	W/mK	layer2 温度传导率	410
α_1	m^2/s	layer1 散热率	6×10^{-5}
α_2	m^2/s	layer2 散热率	1.0×10^{-4}
h_1	W/m^{2}K	layer1 和 layer2 之间传导率	4×10^4
h_2	W/m^{2}K	layer2 和外界之间传导率	510
$L*L$	m^2	die 面积	126

3.2 仿真方法

ATMI 用 image 的方法[16]来建模处理器内部浮点处理单元等不同的功能单元之间的相互影响和对整体处理器温度的影响。该文利用 ATMI 的这个特性来仿真多核处理器。算法只是考虑执行核的温度，没有考虑浮点运算单元、指令 Cache、数据 Cache 以及 L2 Cache 等执行单元的温度，所以只利用ATMI 对

执行核进行了建模。当然,处理器内部是高度集成的,其他执行单元对执行核的温度有一定的影响,在仿真过程中假设这个影响为 0,因此文中利用的处理器模型可以简化为图4,其中每一个执行核对应到 ATMI 模型中是一个 ATMI 矩形(热源矩形),模型通过一个定时器每隔 1 s 读取每一个执行核上的运行队列的功耗来作为 ATMI 功率密度,输出处理器温度 $T_{processor}$ 以及执行核温度 $T_{core,i}$,其中 i 用来标识执行核。

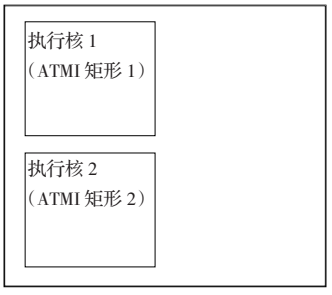


图 4　ATMI 热源矩形示意

模型首先产生一个十分不平衡的状态,即 $Core_1$ 处于温度饱和状态,$Core_2$ 温度较低(图 5 为初始状态的功率密度图,可以看出右边的执行核温度较低)。在仿真过程中,首先仿真该文的温度感知多核调度算法,算法中充分考虑了进程分配策略并具有 MST 启发式方法,然后仿真没有温度感知能力的 Linux 调度算法以及 Merkel 等人[6]的温度感知多核调度算法,最后对文中的温度感知调度算法进行了四核和八核仿真。

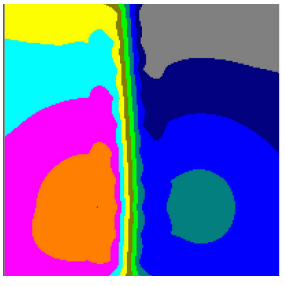


图 5　仿真初始状态

3.3　人造线程

在仿真中,模型利用表 4 中列出的人造线程属性,将线程分为冷、暖和热三种类型,利用 2.1 节的线程分配策略将线程分配到执行核上,执行核的功率密度由公式(4)获得。

表 4　人造线程类型

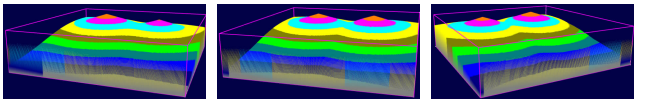
类型	功耗/W
热	6~10
温	4~6
冷	0~4

应该注意的是,一个给定的应用程序的暖热取决于当前并行运行的线程数量和它们的特点。例如,如果总线冲突限制了指令执行,存储受限并行运行的线程越多,线程越冷。然而,论文的研究没有关注这一问题,而是关注于对于一组给定的并发线程,线程迁移对处理器温度带来的收益。

3.4　仿真结果

三次仿真得到的功率密度图如图 6。

仿真过程中,发现该文温度感知调度算法达到稳定状态时的功率密度图比 Merkel 等人提出的算法更均匀,线程迁移次数明显减少(如图 8 所示),但是图 6(c)所示没有温度感知的调度算法的功率密度图反而比 Merkel 等人算法产生的功率密度图更一致,考虑到 Linux 调度算法的特性,分析是由于仿真的进程中三类人造线程比例不同造成,因此使系统中"热"进程占了多数,重新做了一次仿真,得到图 7 所示功率密度图。由此可以得出:无温度感知调度算法产生的稳定状态与进程迁移的次序及三类人造进程比例相关。



(a)Merkel 等人的温度感知调度算法　(b)该文温度感知调度算法　(c)无温度感知调度算法

图 6　仿真结果对比

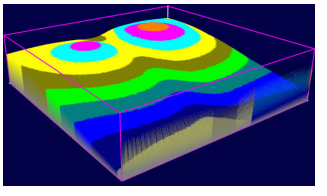


图 7　无温度感知调度算法的另一个结果

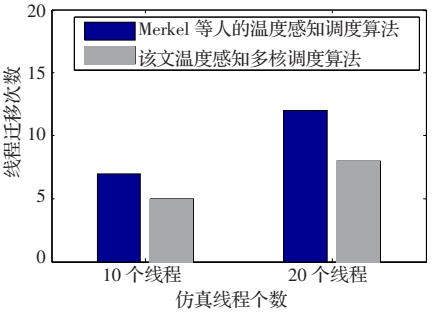


图 8　线程迁移次数对比

为了说明算法对核数目的自适应性,仿真了四核和八核的温度感知调度算法。从图 9 中的仿真结果中可以得出:无论是四核还是八核算法都有效地降低了各个执行核的功率密度而且"最冷"优先和"邻居"优先分配策略相结合能很好地提高执行核的稳定性。

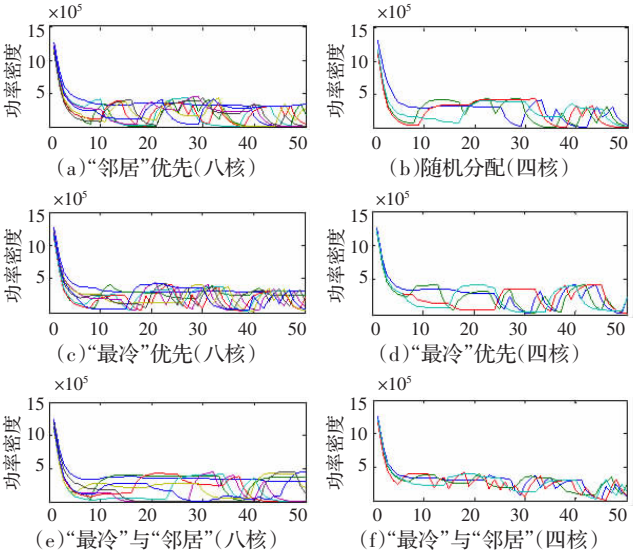


图 9　八核和四核分配策略对比

4 结论

针对一般进程提出了一个多核温度感知调度算法,该算法考虑了“最冷”优先、“邻居”优先等进程分配策略,然后结合Linux中的负载平衡器提出了温度和负载结合的负载均衡器,实现了在线读取性能计数器并对进程的行为做指数加权平均预测,最后通过动态更改进程优先级使Linux调度器具有温度感知能力。仿真表明:(1)针对一个2、4、8核处理器,该调度算法较无温度感知的算法可以创建更均匀的功率密度图。(2)带有MST启发式方法的调度算法得到的功率密度图更加均匀。(3)“最冷”优先和“邻居”优先相结合的进程分配策略更能提高执行核的稳定性,提高了处理器的寿命和稳定性。

参考文献:

[1] Bailey C.Modelling the effect of temperature on product reliability[C]//Semiconductor Thermal Measurement and Management Symposium,Ninteenth Annual IEEE 2003.[S.l.]:IEEE Computer Science,2003:324-331.

[2] Viswanath R,Wakharkar V,Watwe A,et al.Thermal performance challenges from silicon to systems[J].Intel Technology Journal,2000(Q3):57-89.

[3] Mahajan R,Brown K,Atluri V.The evolution of microprocessor packaging[J].Intel Technology Journal,2002(Q3):89-101.

[4] Heo S,Barr K,Asanovic K.Reducing power density through activity migration[C]//Proceedings of the 2003 the International Symposium on Low Power Electronics and Design.[S.l.]:ACM,2003:217-222.

[5] Michaud P,Seznec A,Fetis D,et al.A study of thread migration in temperature-constrained multicores[J].ACM Transactions on Architecture and Code Optimization,2007(4):23-36.

[6] Merkel A,Bellosa F.Balancing power consumption in multiprocessor systems[C]//Proceedings of the 2006 EuroSys Conference on ACM SIGOPS Operating Systems Review.[S.l.]:ACM,2006(40):403-414.

[7] Stavrou K,Trancoso P.Thermal-aware scheduling:A solution for future chip multiprocessors thermal problems[C]//Proceedings of the 9th EUROMICRO Conference on Digital System Design.[S.l.]:IEEE Computer Science,2006:123-126.

[8] Jason R,Roberts J.Multi-core programming:Increasing performance through software multithreading[M].[S.l.]:Intel Press,2007:33-40.

[9] Christiana I,Sazeides Y,Michaud P,et al.Third International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems[Z].European Network of Excellence on High Performance and Embedded Architecture and Compilation,2007(7):45-49.

[10] Mesa-Martinez F J,Brown M,Nayfach-Battilana J,et al.Measuring performance,power,and temperature from real processors[C]//Proceedings of the 2007 Workshop on Experimental Computer Science.[S.l.]:USENIX Association,2007:17-27.

[11] Donald J,Martonosi M.Techniques for multicore thermal management:Classification and new exploration[C]//Proceedings of the International Symposium on Computer Architecture.[S.l.]:IEEE Computer Science,2006:78-88.

[12] Frank B,Kellner S,Waitz M,et al.Event-driven energy accounting of dynamic thermal management[C]//Proceedings of the Workshop on Compilers and Operating Systems for Low Power,USA,New Orleans,2003:44-53.

[13] Bovet D,Cesati M.Understanding the Linux kernel[M].3rd ed.[S.l.]:O'Reilly Publisher,2005.

[14] Naffziger S,Stackhouse B,Grutkowski T,et al.The implementation of a 2-core,multi-threaded Itanium family processor[C]//IEEE International Solid-State Circuits Conference Digest of Technical Papers.[S.l.]:IEEE Computer Science,2006(41):197-209.

[15] Jacobson H,Bose P,Hu Zhi-gang,et al.Stretching the limits of clock-gating efficiency in server-class processors[C]//11th International Symposium on High-Performance Computer Architecture,2005.[S.l.]:IEEE Computer Science,2005:238-242.

[16] Fisher T S,Avedisian C T,Krusius J P.Transient thermal response due to periodic heating on a convectively cooled substrate[J].IEEE Transactions on Components,Packaging,and Manufacturing Technology,1996,19(1):255-262.

(上接71页)

为了验证位平面编码器的性能,表2给出了对三幅标准图Lena、Baboon、Peppers小波变换后分辨率0、分辨率1和分辨率2(分辨率i为$N-i$次小波变换对应的子带)中64×64的码块编码所用时间。由表中数据可知该结构对标准图64×64的码块处理时间均小于3 ms,满足JPEG2000中位平面编码在照相机领域的硬件实现要求。

表2 标准图码块处理结果 ms

	Rlvl0	Rlvl1	Rlvl2
Lena	2.823	2.771	2.788
Baboon	2.809	2.780	2.781
Peppers	2.821	2.767	2.779

4 结论

采用3个状态机控制编码操作,并用局部优化和模板数据缓冲技术,提出了一种简单、灵活的新结构,提高了编码效率,减小了硬件实现的资源消耗,在码块处理上具有很大灵活性。设计了硬件结构的Verilog HDL模型,进行了仿真和逻辑综合,并用FPGA进行了验证。仿真和综合结果表明,设计的硬件结构是正确的,最高频率可达82 MHz,满足JPEG2000中位平面编码在照相机领域的硬件实现要求。

参考文献:

[1] Boliek M,Christopoulos C,Majani E.ISO/IEC FCD15444-1 JPEG2000 Part I Final Committee DraftVersion1.0[S].

[2] 王勇,郑南宁,梅魁志,等.一种高效的JPEG2000位平面编码器设计[J].西安交通大学学报,2005,39(2).

[3] 韩彦菊,许超.JPEG2000分数位平面编码器的FPGA电路实现[J].计算机工程,2005,31(15).

[4] 汪浩,罗伟栋.JPEG2000中位平面编码的存储优化方案设计和实现[J].微计算机信息,2005,21(2).

[5] 刘雷波.JPEG2000静止图像压缩关键技术研究及VLSI实现[D].北京:清华大学微电子学研究所,2004.

[6] 乔世杰,张益民,高勇.JPEG2000中位平面编码的VLSI结构设计[J].电子器件,2007,30(6).