第36卷 第14期 计 算 机 工 程 2010年7月
Vol.36 No.14 Computer Engineering July 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)14—0259—03 文献标识码：A 中图分类号：TP368

基于线程级的同时多线程处理器功耗评估

张朝中，何立强，徐晓东
(内蒙古大学计算机学院，呼和浩特 010021)

摘 要：针对同时多线程处理器中每个线程的功耗评估问题，提出一种同时多线程基于线程级的功耗评估方法。该方法可使系统在运行过程中统计出各线程对各部件的详细功耗情况，方便地衡量在多线程运行时各线程所产生的功耗。为同时多线程处理器进行基于功耗已知的线程调度和取指策略研究提供了基础条件。实验结果表明，各线程的功耗之和与总功耗相等。

关键词：同时多线程；线程级；功耗；取指策略

Power Evaluation for Simultaneous MultiThreading Processor Based on Thread-level

ZHANG Chao-zhong, HE Li-qiang, XU Xiao-dong
(College of Computer Science, Inner Mongolia University, Hohhot 010021)

【Abstract】 To estimate the power consumptions of each thread in the Simultaneous MultiThreading(SMT) processor, this paper presents a new thread-level power estimation scheme for SMT processor. It can count the power consumptions of each unit in the processor for the concurrent running threads. The scheme provides a functionality for power-aware thread scheduling and instruction fetching policies in a SMT processor. Experimental result shows that the sum of power consumptions of each thread is equal to the whole.

【Key words】 Simultaneous MultiThreading(SMT); thread-level; power; fetch policy

1 概述

同时多线程(Simultaneous Multithreading, SMT)处理器能提高系统的指令级并行性(ILP)和线程级并行性(TLP)，是目前比较流行的处理器设计。当今，工业界流行的趋势就是将片上多核处理器(CMP)与 SMT 2 种体系结构相结合，如 Intel Core2 Duo、IBM Power5 和 Power6 以及 SUN 公司的 T1、T2、Niagara[1]等处理器都已经支持了这种结合的体系结构。同时，随着晶体管集成度的不断增大，处理器的功耗问题已经成为处理器设计者必须考虑的重要问题之一。此外，由于同时多线程取指策略对线程的选择，一些恶意线程会对系统造成如拒绝服务攻击(DoS)或热点(Hotspot)[2]等问题，严重影响了系统以及其他线程的性能。因此，研究并了解同时多线程处理器中各线程所产生的功耗以及由功耗转变成的热量等因素至关重要。但目前对于同时多线程功耗问题的研究，基本上只关注在整个部件级(如 cache、ALU、register 文件等)功耗的研究，其功耗值是多个同时运行线程对整个部件的功耗之和，尚未对单线程独立的功耗进行统计并进行研究。

本文提出一种线程级同时多线程处理器的功耗评估方法，该方法可以计算出每个线程在运行过程中对各部件所产生的具体功耗。同时为对同时多线程处理器进行功耗密度可知或温度可知的动态控制，以及基于各线程功耗进行取指线程调度提供了基础条件。模拟实验结果表明，该方法正确地统计了各线程在各个功能部件的功耗值，其值之和与原始程序中给出的总和完全相等。

2 同时多线程体系结构

同时多线程结构由文献[3]提出，其结构基于 Alpha21164 处理器，如图 1 所示。该结构是基于乱序执行的超标量处理器来执行的。但为了实现同时多线程，即同时多个线程在处理器上执行，同时多线程处理器为每个线程单独设置了一套名为“线程上下文(context)”的硬件，用来保存记录每个线程在处理器中的运行状态。其中包括程序计数器、返回地址栈和寄存器文件。系统中的其他部件由多个线程共享，可通过各个线程 ID 来对各个线程进行区分。本文就是在此结构的处理器基础之上进行研究。

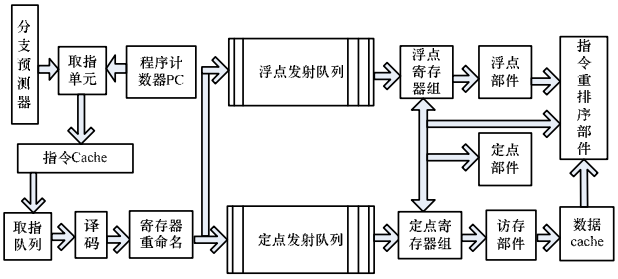

图 1 同时多线程处理器体系结构

3 WATTCH 功耗模型

WATTCH 功耗模型[4]是普林斯顿大学的 David Brooks、Margaret Martonosi 以及 Intel 公司的 Vivek Tiwari 于 2000 年提出的一种体系结构级的功耗模型。它可以快速有效地通过每个周期系统资源的使用情况来评估 CPU 各单元的功耗。

基金项目：内蒙古自然科学基金资助项目(2009BS0901)；内蒙古大学博士启动基金资助项目(208041)

作者简介：张朝中(1982－)，男，硕士研究生，主研方向：处理器微体系结构；何立强、徐晓东，副教授

收稿日期：2010-01-20 **E-mail**：zcz917@yahoo.com.cn

在 WATTCH 功耗模型中计算 ncc,cc1,cc2,cc3 等 4 种功耗值。各值详细解释见文献[4]。

本文提出的方法基于 WATTCH 功耗模型。在计算过程中，系统静态功耗为每周期加上一个固定的值。静态功耗不做细分，为部件级的功耗。而笔者又增加了一个 snamepower 参数值。该值是计算当所有的线程对部件的访问数为 0 时，该部件的 cc3 的值。例如 srenamepower 为计算当 renamepower 单元没有线程访问时，rename 部件 cc3 的值。其他参数的具体细分计算详见第 4 节。

4 基于线程级功耗评估的具体实现

根据 WATTCH 功耗模型，首先需要记录每个线程对每个部件，在一个周期内的访问次数。每个线程设置一个计数器，各个线程是通过线程的 id 来相互区分。这里需要说明的是，笔者采用的是每个周期记录一次各个线程对各个部件相应的计数器，计算相应的功耗逐次累加得到总功耗，每周期计算功耗后调用相应的清零函数，将所有的计数器全部清零，而不是采用所有程序运行完，再统计计数器的值。

得出各线程对各部件的访问次数以后，就要分别计算各线程对各部件的功耗。这个部分情况比较复杂。各个部件的情况不同，各个线程对于功耗的细节计算处理方法也不同。计算的过程中处理方法如下：

对于 ncc 的计算，认为该值为纯静态功耗，所以并不需要对其进行细分。直接在每个周期加上固定的值(由原有的 WATTCH 功耗模型提供)。

对于 cc1 的计算，该值的计算要细分成几种情况进行讨论。首先，当整个部件的访问值为 0 时，增加一个参数来记录此时的 cc1 功耗，此功耗不属于任何线程，类似于静态功耗。当有些部件的访问值不等于 0 但小于一个阈值时，(阈值的设定，采用原有的 WATTCH 功耗模型中对各部件各功耗计算时划分方法。)此时，笔者按所有线程对该部件的访问次数所占比例分担此时 cc1 的功耗。如果线程对该部件访问次数为零，则所占比例也为 0，因此，无须分担。当整个部件的访问值大于阈值时，各线程按照相同的公式代入各自的访问值进行计算此时各自的 cc1 功耗。

对于 cc2 的计算，cc2 是根据各自的访问值，通过线性公式计算得出功耗值,，计算比较简单，只需要将各自的访问值代入相同的公式便可计算各线程在各部件的 cc2 功耗。

对于 cc3 的计算，该值的计算分为 2 种情况，当该部件的访问值为 0 时，设置一个值记录此时的 cc3 功耗。(如 rename 部件，设置 srenamepower 来记录此种情况的功耗值)。该功耗值独立成块，不属于任何线程，当该部件的访问值不为 0，直接根据各自的访问值代入公式计算得出各自的 cc3 功耗。

值得注意的是，因为各部件功耗的计算情况不同，所以有极个别的部件要特殊考虑，如 ALU 单元。除了要考虑 ALU 单元的访问情况，还要细分考虑是整型 ALU 的访问和浮点型 ALU 的访问 2 种情况。在各自的情况下，在细分 cc1、cc2、cc3 的计算。本文是在 WATTCH 功耗模型对各部件各功耗值计算基础上，细分各个线程对该部件的各种功耗值。

各线程对各部件总共统计出 cc1、cc2、cc3 另外还有部件零访问时 cc3 功耗，记作 spower_cc3。上述 4 个部分的功耗构成原有的整个部件的动态功耗。

5 实验方法与结果

实验使用的模拟器是 M-Sim[5]。该模拟器是由 Simplescalar3.0 模拟器修改后支持同时多线程的一个很好地模拟器。模拟环境的主要配置见表 1。工作负载使用的是 Spec CPU 2000。先对工作负载做一个分类，以各线程的功耗高低划分。笔者将工作负载划分成 3 类 :高功耗(H)，中等功耗(M)，低功耗(L)。

表 1 模拟环境的配置

参数	配置
处理器宽度	8-width fetch, issue, commit
ROB 大小	512
LSQ 大小	256
res:memport	6
IL1 Cache	64 KB, 2 way, 2 cycle
DL1 Cache	32 KB, 4 way, 2 cycle
L2 Cache	512 KB, 8 way, 12 cycle
ITLB	16 entry
DTLB	32 entry
Memory 延迟	200 cycle

实验分别选取了不同的取指策略，对 2 线程和 3 线程进行了实验结果比较分析。选取的取指策略分别为 ICOUNT、ICOUNT-half、IPCBFP[6]和 IPCBFP2。其中，ICOUNT-half 是根据 ICOUNT 策略的一种简单的变化，而 IPCBFP2 是对 IPCBFP 2 个实现公式[6]的一个区分 IPCBFP，记作 IPCBFP2。实验对 4 种取指策略对各个线程的调度，功耗调配的影响进行分析对比。两线程和三线程的组合情况，如表 2 所示。当运行最快的线程达到 1 亿条指令时，模拟过程结束。

表 2 工作负载的组合

类别	工作负载	组合名称
H-H	galgel, wupwise	MIX1
M-M	applu, vortex	MIX2
L-L	lucas, twolf	MIX3
M-L	gzip, twolf	MIX4
M-H	gzip, galgel	MIX5
H-L	galgel, twolf	MIX6
H-H-H	galgel, gcc, wupwise	MIX7
H-H-L	mesa, equake, art	MIX8
H-M-H	sixtrace, vpr, ammp	MIX9
L-L-L	parser, lucas, twolf	MIX10
H-L-L	eon, ammp, lucas	MIX11
M-M-L	fma3d, mgrid, twolf	MIX12

(1)不同取指策略所取线程的功耗大小和所占比例

为了能清楚地反映出取指策略对个线程功耗的影响，笔者通过实验，对比负载组合 MIX1, MIX5, MIX9 在 4 种不同的取指策略下功耗的变化。

由图 2 可知，各线程所占功耗不同，，MIX5 在 ICOUNT 取指策略下，0 号线程功耗占 57%左右，1 号线程功耗占 40%左右，剩下为 static 功耗。(注意：这里的 static 为部件零访问时，该部件 cc3 的功耗。)而在 IPCBFP2 取指策略下，0 号线程的功耗比例却增大到了 70%。

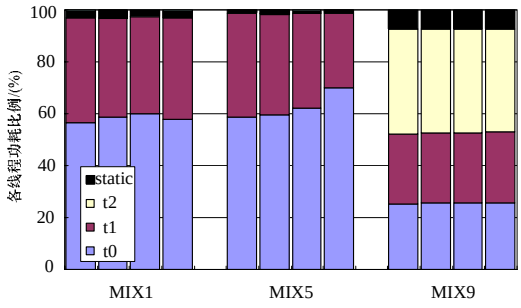

图 2 3 个工作负载组合在各取指策略下的功耗比例

由图 3 可知系统整体的性能(IPC)并没有降低。因此，在不影响性能的前提下，动态调节各线程功耗比例，防止拒绝服务攻击(QoS)或某个线程产生功耗过大等问题，是可以通过

取指策略来实现，即基于功耗意识的取指策略。在程序运行过程中，统计各线程的功耗，如果某线程的功耗过大，有可能产生 Hotspot 或 power-density 等问题时，停止对该线程取指，这样在解决上述问题的同时，还可以均衡各线程功耗，降低系统的峰值功耗。

图 3 说明了在 4 种不同的取指策略下，选择 3 个代表性的组合，各组合的各个线程的功耗。可以看出，取指策略对每个线程的功耗影响。取指策略的不同，对于各个线程的功耗会有很大的影响。因此，可以通过对各线程的调度，来调整系统的功耗的大小当指令周期数不断增大的同时，功耗的影响就会越大。因此，根据线程的功耗调度，具有重要的意义。

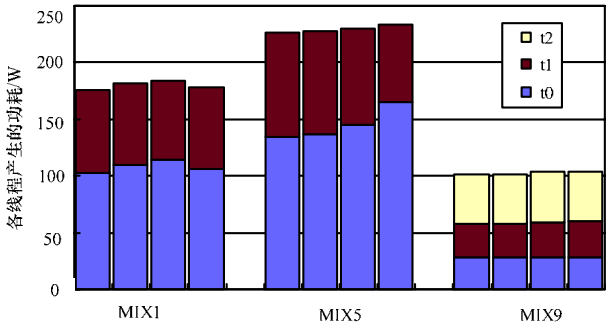

图 3　3 个工作负载在各取指策略下的功耗

(2)各线程各单元所占功耗比例

通过实验可以观察出各个线程的功耗的不同，因而进行线程调度。但是对于不同的工作负载，由于程序运行行为或特征不同，对各个部件的使用情况也不尽相同，因此还需要清楚了解每个负载在不同取指策略下，各单元所产生的功耗情况。图 4 为 MIX9 在 4 种取指策略下，t0、t1 和 t2 各个单元的功耗比例。

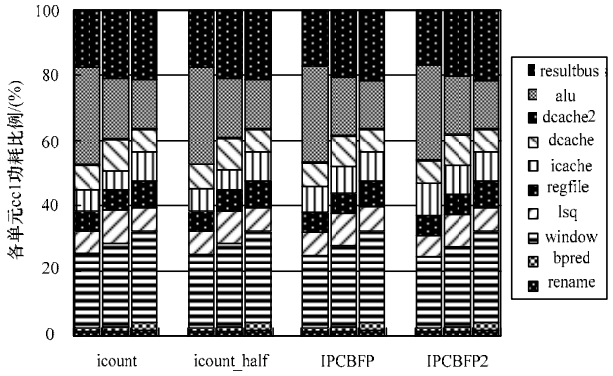

图 4　4 种取指策略下各单元 cc1 的功耗比例

不同线程的每个单元所在的功耗是不相同的。比如在各取指策略下，t0 的 ALU 单元的功耗明显比其他线程所占功耗大。而且本文所绘制的图是最后的结果。如果，在程序运行的过程中进行跟踪各线程各单元的功耗情况，可以选择几个比较重要的单元进行跟踪，如果某个单元产生的功耗过大，可以根据跟踪的情况选择该单元功耗较小的线程。这样就能有效地防止 Power-Density 和 Hotspot 问题的产生(限于篇幅，本文只给出三线程各单元 cc1 的功耗比例)。

6　相关工作

在以往的研究中，文献[2]提出的防止 power-density 和拒绝服务攻击问题的方法，是根据对某个单元的访问次数的情况，来判断该线程是否是恶意线程。如果某线程对该单元的访问频率超过所设定的阈值，则视该线程为恶意线程，从而限制了对该线程的取指。但是该方法只是通过对单元访问次数的跟踪来近似地估算该线程会产生的功耗，而没有直接地给出该线程的功耗或该线程对单元的功耗。

在文献[7]提出了基于功能单元的使用率，在 SMT 中，对多线程进行取指。它通过跟踪功能单元的使用情况，而由编译器进行对一些信息的保存，然后进行调度。该方法的缺点是一方面需要编译器干预，另一方面，也没有给出明确地功能单元的功耗值而只是利用功能单元的使用情况，近似地评估系统的功能单元的功耗。

文献[8]提出了一种算法，与寄存器文件有关，主要是根据检测出数据相关等问题，采用一种机制来减少对寄存器文件的访问，从而降低寄存器的功耗，进一步降低系统功耗。该观点类似于文献[7]，需要编译器的干预。同时也没有给出寄存器文件的具体功耗。而且该思想是基于整个部件级上，没有细分到各线程对寄存器文件的访问情况。

7　结束语

实验表明，基于线程级功耗的评估方法细分后的各线程对各部件的功耗之和与未细分的整个功耗的数据，以及 cc1,cc2,cc3 的功耗数据完全吻合。下一步工作将针对各线程的功耗情况，在程序运行过程中进行适当地线程调度，使尽可能减少系统总功耗。

参考文献

[1] Acosta C, Cazorla F J, Ramirez A, et al. MFLUSH: Handling Long-latency Loads in SMT On-chip Multiprocessors[C]//Proc. of the 37th International Conference on Parallel Processing. Portland, Oregon, USA: [s. n.], 2008.

[2] Hasan J, Jalote A, Vijaykumar T N, et al. Heat Stroke: Power-density-based Denial of Service in SMT[C]//Proc. of HPCA'05. San Francisco, CA, USA: [s. n.], 2005: 166-177.

[3] Tullsen D, Eggers S J, Levy H M. Simultaneous Multithreading: Maximizing On-chip Parallelism[C]//Proc. of ISCA'95. Santa Margherita Ligure, Italy: [s. n.], 1995.

[4] Brooks D, Tiwari V, Martonosi M. Wattch: A Framework for Architectural-level Power Analysis and Optimizations[C]//Proc. of ISCA'00. New York, USA: [s. n.], 2000.

[5] Sharkey J J, Ponomarev D, Ghose K. M-SIM: A Flexible, Multithreaded Architectural Simulation Environment[R]. New York, USA: State University of New York at Binghamton, Tech. Rep.: CS-TR-05-DP01, 2005.

[6] 何立强, 刘志勇. 基于负载瞬时 IPC 性能的同时多线程处理器取指策略[J]. 计算机学报, 2007, 30(4): 629-637.

[7] Deepak B M I, Lakshmi N B, Madhu S S G. Functional Unit Usage Based Thread Selection in a Simultaneous Mutithreaded Processor[C]//Proc. of HIPC'06. [S. l.]: IEEE Press, 2006.

[8] Park S, Shrivastava A, Dutt N. Bypass Aware Instruction Scheduling for Register File Power Reduction[C]//Proc. of LCTES'06. Ottawa, Ontario, Canada: [s. n.], 2006.

编辑　金胡考