文章编号:1001 - 9081(2011)09 - 2567 - 04
doi:10.3724/SP.J.1087.2011.02567

基于多核微处理器温度感知的线程调度算法

屈双喜,张民选,刘　涛,刘光辉
(国防科学技术大学 计算机学院,长沙 410073)
(shuangxiqu@nudt.edu.cn)

摘　要:由于多核微处理消耗更多的能量,导致其热点数目增加,温度分布不平衡加剧,因而对性能产生更大的负面影响。为了解决这个问题,提出一种基于多核微处理器温度感知的线程调度算法来减少热紧急事件、提高性能,并在一个 Intel 的多核微处理器平台上实现了该算法。实验结果表明,在各种负载组合下,该算法可以减少 9.6% ~ 78.5% 的动态热管理次数。与 Linux 标准调度算法相比,吞吐率平均可以提高 5.2%,最大可提高 9.7%。

关键词:多核微处理器;温度感知;线程调度;贪心算法
中图分类号: TP302.7　　**文献标志码**:A

Temperature-aware thread scheduling algorithm for multi-core processors

QU Shuang-xi, ZHANG Min-xuan, LIU Tao, LIU Guang-hui
(*College of Computer, National University of Defense Technology, Changsha Hunan* 410073, *China*)

Abstract: Multi-core microprocessor consumes more power, leading to an increase in the number of its hot spots and uneven distribution of temperature, which creates a greater negative impact on performance. Concerning this problem, a temperature-aware thread scheduling algorithm was proposed to reduce the thermal emergencies and improve the throughput, which had been implemented in the Linux kernel on Intel's Quad-Core system. The experimental results show that this schedule algorithm can reduce 9.6% - 78.5% of thermal emergencies in various combinations of workloads, and has an average of 5.2% and up to 9.7% throughput higher than the Linux standard scheduler.

Key words: multi-core processor; temperature-aware; thread scheduling; greedy algorithm

0　引言

多核体系结构已经成为微处理器设计的主流,然而随着器件特征尺寸的不断缩小以及集成度的不断提高,处理器的功耗密度和温度也不断上升[1]。过高的温度不仅降低了芯片的可靠性,同时也对处理器的性能带来了很大的影响。与此同时,芯片的漏流功耗也会随着温度的上升呈指数上升,而漏流功耗的增加反过来又会导致芯片温度进一步上升,从而可能导致热失控[2]。芯片温度在空间和时间上的巨大变化也给处理器的封装和冷却带来了巨大挑战。因此,利用动态热管理技术(Dynamic Thermal Management, DTM)动态地管理芯片的温度十分重要。

目前已经提出了许多基于硬件和软件的 DTM 机制。基于硬件的动态热管理技术,如动态电压频率缩放和门控时钟等,其主要思想是当处理器温度达到某一触发温度时,利用各种硬件机制对处理器温度进行控制[2-6]。然而,采用硬件 DTM 机制往往会对性能造成较大影响。而基于软件的 DTM 技术,比如线程迁移等,主要思想是利用不同任务之间的温度变化,在适当的时候对任务进行交换,从而控制处理器的温度[7-13]。

Donald 等人提出了综合利用硬件 DTM 技术与任务迁移策略使芯片获得最大吞吐率[9]。Heat-and-Run 技术通过运行时动态分配和迁移线程来平衡芯片的温度[12]。Yang 等人提出了一种启发式调度算法 ThresHot 来减轻处理器的热压力[13],但仅针对单核做了分析。在文献[14]中,提出了一种基于热行为分组的多核系统温度感知调度器。根据应用程序的热行为,使用稳态温度 T_{ss} 对程序进行分组,将热的线程迁移到一个需要最长时间达到阈值温度的核上。在文献[10]中,提出了一种使用软硬件技术(优先权调度,门控时钟)、细粒度、正交的热管理技术:HybDTM。为了估算温度,HybDTM 在使用硬件性能计数器的基础上,建立了一个基于衰退的热模型。在文献[15]中,对具有最少代码的任务赋有优先迁移权,以此来降低任务迁移所带来的额外性能开销。

上述研究绝大多数都是采用模拟的方法,核与核之间的相互影响以及芯片其余部分所消耗的功耗都被忽略了。本文基于多核微处理器温度管理问题,提出了一种贪心调度算法(Greedy Scheduling Algorithm,GSA),并在一个实际的多核处理器上实现了该算法。GSA 的主要目标是减少热紧急事件(热紧急事件是指芯片温度达到一定值而触发硬件 DTM 机制)所带来的性能损失,并尽量不增加额外开销。GSA 主要是基于以下现象提出的:处理器越早达到一个较高的温度状态,处理器就能耗散更多的热量,这是因为热量耗散的速度与处理器温度和环境温度之差成正比。因此,GSA 在每次调度时选择最热的任务执行,只要该任务不引起热紧急事件。

收稿日期:2011 - 04 - 06;**修回日期**:2011 - 06 - 17。　**基金项目**:国家自然科学基金资助项目(60970036,60703074);国家 863 计划项目(2009AA01Z124);教育部"高性能微处理器技术"创新团队资助项目(IRT0614)。

作者简介:屈双喜(1980 -),男,湖南永州人,博士研究生,主要研究方向:计算机系统结构、微电子技术;　张民选(1954 -),男,湖南邵阳人,教授,博士生导师,主要研究方向:高性能计算机系统及其实现、微电子技术;　刘涛(1980 -),男,湖北鄂州人,博士研究生,主要研究方向:计算机系统结构;　刘光辉(1982 -),男,河南鹤壁人,博士研究生,主要研究方向:计算机系统结构。

1 贪心调度算法

当处理器过热时，必须降频降压甚至停顿，这会导致处理器的利用率和吞吐率下降，增加响应时间，从而给性能带来负面影响。GSA 的主要目标是最大化 CPU 的利用率和吞吐率。Linux 操作系统周期性地打断任务的执行并决定是否该调入一个其他的任务执行，通过修改 Linux 的调度算法可以实现温度感知调度策略。在每个调度间隔，GSA 选择一个任务执行，期望使得总体热紧急事件最少。

1.1 基本原理

为了使温度不超过阈值，传统线程迁移算法将热的核上的线程迁移到冷的核上，让热的核得以冷却。当处理器运行在一个比较温和的环境中，并且有一定数量的冷的线程时，这种策略的确可以降低平均温度并减少热紧急事件。但是，如果处理器运行在一个较热的环境或者冷线程很少时，这种方法就会不可避免地频繁引起热紧急事件。下面通过一个示意图来说明这个问题，见图 1。

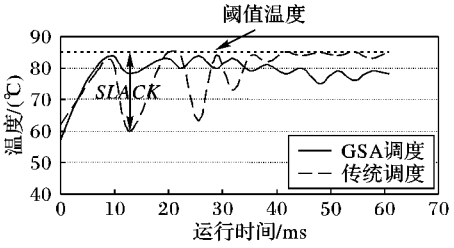

图 1 传统温度感知调度与使用 GSA 处理器温度响应示意图

如图 1 中实线所示，当某个核的温度接近阈值温度时，传统的温度调度策略会将该核上热的线程迁移出去，调入一个冷的任务执行，使得热的核能很快冷却。如果冷的任务较少，那么当冷的任务执行完之后，就会频繁出现热紧急事件。造成这个问题的主要原因是传统调度算法没有充分利用处理器的“温度余量”(*SLACK*)(“温度余量”在这里被定义为处理器的温度与阈值温度之差，“温度余量”可以抽象为一种冷却资源)。传统的调度算法在程序开始执行时浪费了太多的“温度余量”，以致后面需要的时候却没有了。

不同于传统的调度算法，GSA 从一开始就选择最热的线程运行，只要不引起热紧急事件，将冷的线程尽可能留在后面执行。这种策略尽可能地保存“温度余量”，只有在真正需要时才使用这些“温度余量”。

1.2 理论分析

热传导与电现象之间存在一个很著名的对偶性，这一对偶性为建立体系结构级热模型提供了基础[1]：

$$\frac{1}{R}T + C\frac{\mathrm{d}T}{\mathrm{d}t} = P \tag{1}$$

其中：T 表示当前温度与环境温度之差，R 和 C 表示芯片的有效热阻和热容，P 表示功耗。

将式(1)两边乘以 $\mathrm{d}t$，然后从 t_0 至 t_2 进行积分(t_0 表示某一完整线程调度周期的开始时刻，t_2 代表该调度周期结束时刻)，可以得到：

$$\int_{t_0}^{t_2}\frac{1}{R}T\mathrm{d}t + \int_{t_0}^{t_2}C\mathrm{d}T = \int_{t_0}^{t_2}P\mathrm{d}t \tag{2}$$

式(2)右边表示能量 E，将时间片分成(t_0，t_1)和(t_1，t_2)两个区间，式(2)可以写成：

$$\frac{1}{R}S_1 + CT\Big|_{t_0}^{t_1} + \frac{1}{R}S_2 + CT\Big|_{t_1}^{t_2} = E \tag{3}$$

其中 S_1 和 S_2 表示面积，见图 2。

在式(3)中，R 和 C 都是常数，对于一个特定的调度周期，E 也可以认为是常数。要想减少出现热紧急事件的概率，即减少 $T(t) \geqslant T_{\mathrm{threshold}}(t_1 < t < t_2)$ 出现的概率，则应该在尽可能不引起热紧急时间的前提下，使 S_1 和 $T(t)(t_0 < t < t_1)$ 尽可能大。因此，GSA 的思想可以描述为：在每个调度点，尽可能选择热的且不会产生热紧急事件的线程来运行。值得注意的是，GSA 是基于对某一个调度周期进行定性分析并推理得到的一个合理解，并非全局最优解，因此不一定会使全局热紧急事件最少。事实上，要寻找一个全局最优解是一个 NP 完全问题[16]，因此在运行时实时地获得全局最优解是不现实的。

在式(1)中，如果 $\mathrm{d}T/\mathrm{d}t = 0$，表示芯片达到了稳态温度 T_{ss}，且 $T_{ss} = RP$。这时，P 既代表热量产生的速度，也代表热量耗散的速度。P 越大，T_{ss} 越大，这就意味着热耗散的速度与 T 成正比。因此，如本文引言部分所述，GSA 可以通过这样一个现象来直观地理解：处理器越早达到一个较高的温度状态，耗散的热量也越多。

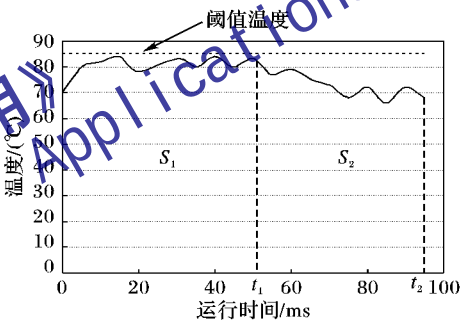

图 2 T 随时间变化曲线

1.3 温度预测

为了正确地选择线程，GSA 必须知道哪个线程是最热的线程并且不会引起热紧急事件。为此，可以建立一个温度预测器来进行温度预测：

式(1)可以写成：

$$\frac{\mathrm{d}T}{\mathrm{d}t} = b \times (T_{ss} - T) \tag{4}$$

其中：$b = 1/RC$，是与硬件相关的一个常数，T_{ss} 是程序的稳态温度。令 $T(0) = T_{init}$，$T(\infty) = T_{ss}$，求解式(4)，可得：

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times \mathrm{e}^{-bt} \tag{5}$$

稳态温度 T_{ss} 可以通过在每个核上运行 SPEC CPU2000 benchmark 来获得。真实的温度可以通过读取核内的数字热传感器(Digital Thermal Sensor，DTS)获取，这样利用式(5)即可计算出 b 的值。一旦 T_{ss} 与 b 确定，每个核在下一调度周期的温度就可以利用式(5)进行计算得到。

1.4 GSA 工作流

在 Intel 体系结构中，可以通过访问机器特定寄存器(Machine Specific Register，MSR)来获取 DTS 的值，在 MSR 中该值是一个无符号数，单位为摄氏度。

在每个调度周期结束时，根据 DTS 的值以及事先计算好的参数，温度预测器可以计算每个核在下一个调度周期的温度。GSA 根据计算出的温度信息，调度相应的线程执行。GSA 的工作流图见图 3。

图 3　GSA 工作流程

2　实验方法

2.1　Linux 标准调度器

Linux 操作系统将任务分为三类:交互式任务、批处理任务和实时任务。实时任务具有最高优先权,其他两类任务的初始优先权相同。Linux 标准调度器对每个任务分配一定的时间片。运行时,所有任务放入相应的优先权队列中,然后按照优先权降序执行。每个任务根据所分配的时间片,占用 CPU 相应的时间,除非出现特定的事件触发线程交换。当任务使用完时间片后,就放入过期队列,调度器选择下一个任务执行。当所有的任务都运行完其相应的时间片之后,就完成了一个完整的调度周期。所有在过期队列中的任务就根据其优先权重新分配相应的时间片,一个新的调度周期开始。Linux 标准调度器只有在出现明显的负载不平衡时才会进行线程迁移。

2.2　GSA 调度器

Linux 基准调度器没有考虑温度问题,为了解决这一问题,提出了 GSA。GSA 不仅考虑多核中负载平衡问题,同时还考虑负载的温度特性。GSA 主要针对批处理任务,并且不改变任务的优先权,保证实时任务不受影响。

在每个调度时刻,内核的温度可以通过 DTS 获得,温度预测器根据 DTS 的值以及预先计算好的相关参数,计算出每个核在下一个调度周期的温度。GSA 选择使得当前内核温度最高同时又不引起热紧急事件的线程运行。为了减少不必要的迁移带来的额外性能开销,对选择算法做了一点改进:如果存在多个线程均可使温度达到一个较高的水平,则选择当前核上的活动线程运行。只有当其他核上满足条件的线程比当前核上的线程温度高出一定值时,才将其他核上的线程迁移过来。

2.3　算法描述

算法 1 描述了 GSA,算法中的一些变量和常量定义如下:N 表示当前核上等待调度的线程数;M 表示其他核上等待调度的线程数;$TemPre(i)$ 表示下一调度周期线程 i 在当前核上运行产生的温度;$SLACK$ 表示温度余量,即当前温度与 $T_{threshold}$ 之差;$T_{threshold}$ 表示引发硬件 DTM 时的温度;$T_{migration}$ 表示预定义的迁移温度差值。

算法 1　Greedy Scheduling Algorithm。

```
for every core
  Tpre ← 0
  SLACK ← 0
  SLACK_temp ← 0
  for i = 1 to N do
    SLACK ← T_threshold − TemPre(i)
    if SLACK ≤ SLACK_temp then
      SLACK_temp ← SLACK
      candidate_temp ← i
    end if
    if Tpre < TemPre(i) and SLACK > 0 then
      Tpre ← TP(i)
      candidate ← i
    end if
  end for
  for j = N + 1 to N + M do
    SLACK ← T_threshold − TemPre(j)
    if (TemPre(j) − Tpre) > T_migration and
      SLACK > 0 then
      Tpre ← TP(j)
      candidate ← j
    end if
  end for
  if all SLACK ≤ 0 than
    candidate ← candidate_temp
  endif
  select thread(candidate) to run
end for
```

3　实验结果与分析

以前的工作主要致力于降低处理器的峰值温度和平均温度,其结论大都是基于模拟的结果。而本文的主要目标是减少热紧急事件,提高吞吐率,并在一个基于 Q8300 处理器,Fedora 7 操作系统的平台上实现了 GSA。实验过程中,环境温度尽可能保持稳定。

首先运行 22 个 SPEC CPU2000 测试程序,根据其热行为对其进行分组,一共分为三组:热的、温和的、冷的。具体分组见表 1。

表 1　测试程序分类

热特性分类	程序名称
热	crafty, gap, gcc, mesa, mgrid, sixtrack, gzip, bzip
温和	applu, apsi, facerec, parser, vortex, wupwise, twolf
冷	ammp, equake, fma3d, lucas, swim, art, mcf

为了验证 GSA,将不同的负载进行组合。将不同组合的测试程序运行结果与 Linux 基准调度器、随机调度器、PTDM 调度器进行比较。实验结果表明 GSA 相比于其他算法,可以更有效地减少热紧急事件,提高系统的吞吐率。各种负载组合见表 2。

表 2　负载组合

序号	程序分组	程序名称
1	HWCCC	mesa, applu, ammp, swim, mcf
2	HWWCC	bzip, parser, apsi, fma3d, art
3	HWWWC	gzip, apsi, facere, vortex, art
4	HHWCC	gcc, mesa, apsi, lucas, art
5	HHWWC	crafty, mgrid, applu, apsi, mcf
6	HHHCC	gap, sixtrap, bzip, ammp, equake
7	HHHWC	gap, gcc, gzip, wupwise, swim
8	HHHHC	crafty, gcc, mesa, gzip, fma3d

图 4 给出了不同调度器对不同负载组合进行调度产生的热紧急事件的次数,并按照 Linux 基准调度器进行了规格化。

从图 4 可以看出,在所比较的几种调度器中,GSA 调度器优于其他调度器。GSA 可以减少 9.6% ~78.5% 热紧急事件(平均减少 41%)。当冷任务较多的时候,随机调度器通过随机选择任务执行,可以有效地减少热紧急事件的次数。这是因为随机调度器在一定程度上可以使热量在整个芯片上分布更加均匀。但是,当冷任务较少时,整个操作温度上升,随机

调度很可能使多个热的任务连续执行,这时就很可能会产生热紧急事件。PTDM 调度器由于采取前摄的温度感知调度方法,在一定程度上避免了调度的盲目性,其效果要好于随机调度器。不过,由于这些调度器都没有充分利用“温度余量”,因此其总体效果均不如 GSA 调度器。

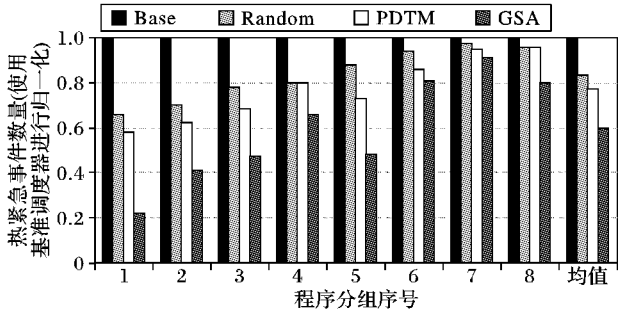

图 4 热紧急事件次数(按 Linux 标准调度器进行归一化)

图 5 给出了各种调度器相对于 Linux 标准调度器对性能的提升。GSA 调度器相比 Linux 基准调度器平均可以获得 5.2% 的性能提升,最大可达 9.7%。从图 5 中还可以发现,对于各种负载而言,各种调度器并非在最热或最冷情况下获得最大性能提升。这是因为如果太冷,热紧急事件本身就很少发生,因此采用什么样的调度算法,对性能影响不大;如果太热,可供调度的空间和选择都很小,因此性能提升也有限。在那些中等温度环境下,温度感知调度算法通过合理安排冷热任务的执行顺序,可以获得最大性能提升。

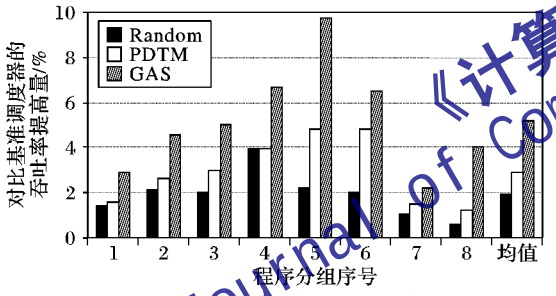

图 5 GSA 调度器对性能的提高

4 结语

本文提出了一种基于多核微处理器的温度感知贪心调度算法 GSA,并在一个四核 Q8300 处理器、Fedora 7 操作系统的平台上实现了该算法。实验结果表明,与 Linux 基准调度器及其他几种温度感知调度器相比,GSA 可以有效地减少热紧急事件,提高系统的吞吐率。GSA 充分利用“温度余量”,使处理器温度尽快上升并保持在一个较高的温度状态(但不超过阈值),这样可以加快热耗散的速度。最后,GSA 调度器不需要额外的硬件支持,可以在任何一个实际的多核处理器上实现。

参考文献:

[1] SKADRON K, STAN M R, HUANG W, *et al*. Temperature-aware microarchitecture [C]// ISCA '03: The Thirtieth International Symposium on Computer Architecture. New York: ACM, 2003: 2 – 13.

[2] BROOKS D, MARTONOSI M. Dynamic thermal management for high-performance microprocessors [C]// HPCA '01: Proceedings of the Seventh International Symposium on High-Performance Computer Architecture. Washington, DC: IEEE Computer Society, 2001: 171.

[3] GUNTHER S, BINNS F, CARMEAN D M, *et al*. Managing the impact of increasing microprocessor power consumption [J]. Intel Technology Journal, 2001, 5(1): 1 – 9.

[4] HEO S, BARR K, ASANOVIC K. Reducing power density through activity migration [C]// ISLPED '03: The International Symposium on Low Power Electronics and Design. New York: ACM, 2003: 217 – 222.

[5] SKADRON K. Hybrid architectural dynamic thermal management [C]// Proceedings of Design, Automation and Test in Europe Conference and Exhibition. Washington, DC: IEEE Computer Society, 2004, 1: 1530 – 1591.

[6] HANSON H, KECKLER S W, GHIASI S, *et al*. Thermal response to DVFS: Analysis with an Intel Pentium M [C]// ISLPED '07: The International Symposium on Low Power Electronics and Design. New York: ACM, 2007: 219 – 224.

[7] SRINIVASAN J, ADVE S V. Predictive dynamic thermal management for multimedia applications [C]// Proceedings of the 17th Annual International Conference on Supercomputing. New York: ACM, 2003: 109 – 120.

[8] CHOI J, CHER C-Y, FRANKE H, *et al*. Thermal-aware task scheduling at the system software level [C]// ISLPED '07: The International Symposium on Low Power Electronics and Design. New York: ACM, 2007: 213 – 218.

[9] DONALD J, MARTONOSI M. Techniques for multicore thermal management: classification and new exploration [C]// ISCA '06: 33rd International Symposium on Computer Architecture. Piscataway, NJ: IEEE, 2006: 78 – 88.

[10] KUMAR A, SHANG L, PEH L-S, *et al*. HybDTM: a coordinated hardware-software approach for dynamic thermal management [C]// DAC '06: Design Automation Conference. New York: ACM, 2006: 548 – 553.

[11] KURSUN E, CHER C-Y, BUYUKTOSUNOGLU A, *et al*. Investigating the effects of task scheduling on thermal behavior [C]// TACS '06: Third Workshop on Temperature-aware Computer Systems. Boston, MA, USA: [s.n.], 2006.

[12] GOMAA M, POWELL M D, VIJAYKUMAR T N. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system [C]// ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2004: 260 – 270.

[13] YANG JUN, ZHOU XIUYI, CHROBAK M, *et al*. Dynamic thermal management through task scheduling [C]// ISPASS 2008: IEEE International Symposium on Performance Analysis of Systems and software. Washington, DC: IEEE Computer Society, 2008: 191 – 201.

[14] YEO I, KIM E J. Temperature-aware scheduler based on thermal behavior grouping in multicore systems [C]// DATE '09: Design, Automation & Test in Europe Conference & Exhibition. New York: ACM, 2009: 946 – 951.

[15] MULAS F, PITTAU M, BUTTU M, *et al*. Thermal balancing policy for streaming computing on multiprocessor architectures [C]// DATE '08: Design, Automation & Test in Europe Conference & Exhibition. New York: ACM, 2008: 734 – 739.

[16] CHROBAK M, DüRR C, HURAND M, *et al*. Algorithms for temperature-aware task scheduling in microprocessor systems [C]// AAIM '08: Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management, LNCS 5034. Berlin: Springer-Verlag, 2008: 120 – 130.