软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(9):2235−2247 [doi: 10.3724/SP.J.1001.2011.03897]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

改进的能量最优 OpenMP 静态调度算法*

董 勇[1+], 陈 娟[1], 杨学军[2]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

Improved Energy-Optimal OpenMP Static Scheduling Algorithm

DONG Yong[1+], CHEN Juan[1], YANG Xue-Jun[2]

[1](College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](National Key Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: yongdong@nudt.edu.cn

Dong Y, Chen J, Yang XJ. Improved energy-optimal OpenMP static scheduling algorithm. *Journal of Software*, 2011,22(9):2235−2247. http://www.jos.org.cn/1000-9825/3897.htm

Abstract: This paper presents expanded hypothesis based OpenMP static scheduling energy optimization algorithm—Improved Energy-Optimal OpenMP Static Scheduling, IEOSS. Based on EOSS algorithm, IEOSS algorithm exploits the impact of memory access latency on performance and energy of parallel loop. Due to cache miss, the optimal chunk S^* scales down processors' voltage/frequency and obtains the minimal energy consumption. Five programs from NPB3.2-OMP for further evaluation. Take 480 processors, 64-byte cache line as an example: with 5% performance loss, IEOSS algorithm improves the energy savings of EP, IS, FT, CG, MG by 10.15%, 4.49%, 81.66%, 2.32%, 10.11% compared with energy consumption of OpenMP chunk scheduling. This shows that IEOSS can effectively improve energy optimization by combining DVS with choosing optimal chunk size.

Key words: energy optimization; OpenMP; loop scheduling; voltage/frequency scaling; IEOSS (improved energy-optimal static scheduling)

摘 要: 基于前期工作的 EOSS 算法,给出了扩展条件下的 OpenMP 静态调度能量优化算法——改进的能量最优 OpenMP 静态调度算法(improved energy-optimal static scheduling,简称 IEOSS).该算法在原有 EOSS 算法的基础上,建模了数据 cache 失效造成的访存延迟对并行循环性能及能量的影响,选择最优调度块大小 S^*,同时结合动态电压/频率调节,获得最小能量消耗.选择 NPB3.2-OMP 的 5 个程序进行模拟,以 480 个处理器、64 字节大小的 cache line 为例,在 5%的性能损失条件下,对比 OpenMP 缺省的块调度的能量消耗,IEOSS 算法可使 EP,IS,FT,CG,MG 程序的并行循环能量消耗分别减少 10.15%,4.49%,81.66%,2.32%和 10.11%.实验结果验证了算法 IEOSS 通过 DVS 结合最优块大小的选择,能够明显改进能量优化效果.

关键词: 能量优化;OpenMP;循环调度;电压/频率调节;IEOSS (improved energy-optimal static scheduling)

* 基金项目: 国家自然科学基金(60921062, 60903044); 国家高技术研究发展计划(863)(2008AA01Z110); 核高基重大软件专项(2009ZX01036-001-003)

收稿时间: 2010-01-14; 修改时间: 2010-03-11; 定稿时间: 2010-06-10

中图法分类号: TP316 文献标识码: A

HPC 系统功耗问题日益严峻.以 2011 年 6 月的 TOP500 第 1 名的“京”系统,其功耗已接近 10MW。以 Jaguar 系统为例,其功耗接近 7MW[1].根据推断,未来 10 年 HPC 系统的功耗将超过 100MW[2].过高的功耗导致了低可靠性、低稳定性以及使用成本过高等问题.功耗优化已经成为 HPC 系统研究的焦点问题之一.

对于大多数并行科学计算来说,并行循环往往占据了整个并行程序执行时间的大部分.降低并行循环的能量消耗,也就成为降低整个并行程序能量的关键.实验结果表明[3−6],针对并行循环的能量优化,对整个科学计算程序的能量优化起到关键作用.

OpenMP 是一个被广泛使用的并行编程接口.多核技术的发展为 OpenMP 的广泛应用提供了更好的机遇,OpenMP 的天然并行性使其更适合多核处理器运算环境.但是,针对特定的 OpenMP 循环调度的能量优化还缺乏研究.本文利用现有 OpenMP 循环调度,在编译器中提供自动的电压调节指导,以用户透明的方式改善程序运行时的能量消耗.这种面向能量优化的编译技术无需更改程序便可以实现程序的低功耗运行,对现有的大量 OpenMP 用户十分有吸引力.

OpenMP 嵌套循环不同于一般的嵌套循环.OpenMP 循环调度不考虑跨迭代的依赖,由程序员保证程序正确性,用户在编程时主要考虑调度块大小的设置.OpenMP 的静态循环调度在程序运行以前就确定不同处理器(核)的循环迭代分配.在 OpenMP 的能量优化中,降低轻负载处理器的电压/频率,以保证在一定性能要求下获得能量节约,是动态电压/频率调节(DVS)的一般思路.但是,我们的研究发现:处理器 DVS 并不能保证所获得的能量节约是最大的,需要综合考虑循环调度和电压调节两个方面,才能获得更优的能量节省.在现有的静态调度策略下,如何将 DVS 和最优块大小选择结合起来,以获得理论上最大的能量节约,是本文研究的目标.在前期的工作中[7],我们得出了理想条件下获得最大能量节约的算法 EOSS.该算法假定每个迭代的执行时间完全相同,并且不存在跨迭代间的依赖,忽略 cache、数据局部性影响,忽略了访存延迟的影响,不考虑处理器间的等待同步的影响等等.EOSS 算法的结论是:静态调度块大小为 1 时,所获得的能量节约是最大的,并从理论上证明了该最优值的正确性.但由于条件过于理想化,得出的结论被认为是理论上的上限值,与实际情况有一定差距.理由是:调度块太小会使得连续的数据被零散地分布在不同处理器上,破坏了同一处理器上的数据局部性,因而不同的数据分布状况对循环的性能和能量是有影响的,导致选择块大小大于 1 更加合理.本文进一步修正了算法 EOSS,考虑了循环调度中 cache line 大小和访存延迟因素的影响,从而获得更加合理的能量优化结果.我们的一项相关工作[6]中,从另一个角度研究了能量和性能之间的关系,将能量作为约束条件,研究了在满足一定能量约束条件下的性能优化问题,提出了两个能量受限问题的算法:(1) 能量受限的静态调度算法 ECSS,它是与文献[6]中的 ESSS 算法相对应的;(2) 能量受限的最优静态调度算法 ECPOSS,证明了该算法在相同能量约束条件下可获得最好性能,与文献[6]中的 EOSS 算法相对应.本文中,我们关注数据 cache 失效造成的访存延迟对并行循环性能及能量的影响.

基于片上多核处理器(chip-multi-processor,简称 CMP)的 DVS 调度算法不同于本文提出的面向能量优化的 OpenMP 循环调度,后者更多的是利用 OpenMP 应用的特点,结合 OpenMP 并行编译器和运行时库,从能量优化的角度改进 OpenMP 调度算法.相比之下,基于 CMP 的 DVS 算法更多的是利用 CMP 的硬件特性,挖掘多核上电压/频率调节机会,这两者不属于同一个研究层面.

面向能量优化的 OpenMP 循环调度同 MPI 并行程序的能量优化不同.MPI 程序的能量优化使用库优化的方式,而不是编译优化.在 MPI 程序中,通常包括很多通信阶段,在这些通信阶段中,处理器因等待通信的完成而处于空闲等待状态.这些通信阶段或者其他的导致 CPU 空闲的阶段被标识出来,通过频率/电压调节等方式,降低处理器的功耗和能量消耗.更多的关于 MPI 能量优化的内容可以参见第 1 节.在我们以前的工作中,考虑了针对 MPI 全局操作进行功耗优化－降低非关键路径上的 CPU 的电压/频率[8].

本文第 1 节介绍相关工作.第 2 节在讨论 EOSS 算法不足的基础上给出 IEOSS(improved energy-optimal static scheduling)算法.第 3 节介绍 IEOSS 算法的实验验证.第 4 节进行总结.

1 相关工作

基于 CMP 的任务分配与调度的功耗优化方法已有许多相关研究[9−13].从系统类型看,既包含实时系统下的 CMP 任务调度[9],也包含一般系统的任务调度[11−14].从优化目标看:一种是从功耗/能量优化的角度进行 CMP 任务调度[9−13];另一种是满足功耗约束条件下的性能最优[14].其中,Yan 等人[9]在 CMP 结构上进行能量有效的实时任务调度,在所有任务具有相同的执行时间限制条件下最小化能量消耗.Chang 等人[10]提出 ETAHM 算法,该算法综合了任务的调度、映射以及 DVS,以降低 CMP 的功耗.Teodoresc 等人[12]针对 CMP 提出一个变化感知的应用调度算法 LinOpt,该算法通过在线阶段性运行,使用线性规划为 CMP 中的每个核找到合适的电压值,在给定的能量条件下,最大化吞吐量,增加吞吐量,降低能量和延迟平方的乘积.Ranga 和 Krishna 等人[13]对 CMP 使用 thread motion(TM)将线程在多个具有固定功耗性能级别的不同处理器核上快速迁移,以满足优化能量的目的.同前面的工作获取最优功耗不同,Zhan 等人[14]实现 CMP 中能量约束条件下,任务集性能最大化.文献[15]对 CMP 上的任务调度进行了能量和性能的权衡.从多核电压调节的设置来看,一类问题是假设每个处理器核的电压/频率可独立进行调节[16−19],也有一部分研究假设所有的处理器核统一进行电压/频率调节[9].

针对片上多处理器的并行循环能量优化类似于我们的工作.Kadayif[3]认为,在片上多处理器系统的嵌套循环执行过程中,每个循环嵌套适合选择不同数量的处理器,多余的处理器可以关闭,以节约能量.此外,Kadayif 用 ILP 公式[5]来决定每个循环嵌套所使用的处理器数.针对循环嵌套的能量优化,相比 OpenMP 的循环能量优化的不同在于:1) 不同的并行化问题.Kadayif 关注片上多处理器系统中数据密集程序的嵌套循环的并行化.在这种类型的程序中,数据访问的局部性对循环迭代的分配敏感,并且嵌套循环内迭代之间可能存在跨迭代的依赖关系.当循环迭代分配到多处理器上时,多个处理器的电压调节需要考虑跨迭代依赖给处理器电压调节带来的影响.当允许每个处理器以不同的比例降频时,无法准确估计因依赖等其他因素带来的性能损失.因此,对这类嵌套循环的多处理器电压调节通常采用实际测试的方式获得每个处理器的合适的低电压值.而 OpenMP 并行循环在 OpenMP 指导语句下不存在跨迭代间的依赖,由用户对存在跨迭代依赖的程序通过关键段方式进行改动保证无跨迭代间依赖;2) 不同的策略.Kadayif 在给定的并行化策略基础上研究了一种 DVS 算法,该方法并不改变并行策略本身.相比之下,我们采用 DVS+重调度的方式,改变了原有的调度,从根本上说是一个新的调度.

在并行循环的能量优化方面,Li[20]在循环段的基础上提出了并行计算的功耗-性能适应方法.该方法中,每个并行段的所有处理器被设置成相同的电压等级,选择一个最优的处理器数.Chen[21]基于指令级 list 调度进行能量优化,对 3 种能量优化方法进行了比较:list scheduling,list scheduling+ DVS 和 rotation scheduling+DVS.结果显示,最后一种方法消耗能量最少.

针对 MPI 的典型低功耗优化包括:Kappiah[22]提出了一个称为 Jitter 的系统,这个系统可以对节点间的间隙时间进行分析,从而降低节点的处理器频率,达到能量节约的目的.Freeh[23]针对低功耗、高性能集群进行研究,将程序分成不同的阶段,针对每个阶段选择一个合适的频率.Lim[24]对 MPI 程序中的通信阶段进行识别,并降低通信阶段处理器的频率.此外,易会战[25]通过对并行程序进行区域划分,使用 DVS 方法降低处理器的功耗,并提出针对 MPI 并行程序的编译指导的通信链路关闭技术.

2 改进的能量最优 OpenMP 静态调度算法(IEOSS)

2.1 基 础

本文使用的机器模型是共享内存多处理器系统.他包含 p 个同构自主的处理器,处理器通过互联网络和一组内存模块相连,所有 p 个处理器可以访问到所有的内存模块.在共享内存并行处理器系统中,一个程序可以在多个处理器上并发执行,从而可以获得高性能.

在大多数程序中,并行化的主要来源是循环.可以并行执行的工作单元通常是一个循环迭代.OpenMP 编译器的一个任务就是在不同的处理器上对循环迭代进行调度,这被称为循环调度.循环调度包括静态调度和动态调度.在本文中,我们只考虑静态调度.

OpenMP 静态调度包括 3 种迭代分配方式:块分配、块循环分配和循环分配.在块分配中,每个处理器分配一组连续的循环迭代.在块循环分配中,分配给每个处理器的迭代采用固定的 stride 进行间隔.在循环分配中,每个处理器一次只分配一个迭代.当 stride 大小为 1 时,块循环分配成为循环分配.

在 OpenMP 中,静态调度使用图 1 所示指导语句实现.其中,*size* 表示块大小.在本文中,无论是对嵌套或者非嵌套循环,我们只对最外层循环迭代进行并行化,并只考虑每个迭代的执行时间相同的情况.主要考虑:1) 对于非嵌套循环,只包括一层迭代,不存在是否为最外层迭代的情况;2) 对于嵌套循环,OpenMP DO 工作共享结构只对紧接其后的循环进行划分和调度,例如图 1 所示的程序,代码中的指导命令只对 *i*-loop 起作用;3) 如果每个迭代的执行时间都不相同,例如,图 1 所示的程序代码中,*j*-loop 变为 for *j*=1,*i*,则每个 block 的执行时间都会不同.此类循环使用静态调度难以获得合理的负载划分,一般采用动态调度,以便获得更好的负载平衡.本文针对静态调度展开研究,动态调度不在本文讨论的范围内.

```
!$ OMP PARALLEL DO
!$ OMP SCHEDULE(STATIC,size)
      for i=1,N,…
            for j=1,M
                  block
```

Fig.1 OpenMP directive

图 1 OpenMP 指导语句

2.2 问题的提出

OpenMP 的静态循环调度使用调度块大小来描述分配给每个处理器的迭代次数.选择不同的块大小对 OpenMP 循环的性能有很大影响:块大小过大,容易造成处理器间的负载不平衡;块大小太小,使得连续存放的数据被相邻的处理器访问,造成“数据碎片”,数据局部性不好.另外,静态调度的负载不平衡在很多时候还源于循环迭代的不规则(每个并行循环迭代的执行时间不同,例如循环中有分支语句等).尽管许多循环选择动态调度来改善负载平衡,但由于动态调度的开销很大,静态调度仍被广泛地应用在许多应用当中.本文中,我们的研究目的是选择合适的调度块大小,使得 OpenMP 静态循环调度在一定的条件假设下获得最大的能量节约.该最优性的问题的求解基于给定的假设条件.

在过去的工作中,我们已证明[6]:在理想情况下**,负载最平衡的调度可获得最大的能量节约.具体调度方法是:在处理器的最后一轮块调度时将负载进行重新分配,使得任何两个处理器上的总负载之差都不大于 1 个迭代.该工作存在以下需要改进之处:1) 该方法只对静态调度中最后一次调度进行重新分配,从迭代分配数来看,与调度块为 1 的调度结果相同,但此调度不能利用现有的 OpenMP 指导语句实现;2) 在这个证明过程中,没有考虑每次迭代过程中因为数据访问所引发的 cache 失效问题.实际上,这里使用了一个假定,即每个处理器的 cache 无限大,在循环执行过程中只需要一次访问即可将全部的数据放入 cache 中.在计算处理器执行时间时,也只考虑了处理器从 cache 中读取数据.

2.3 IEOSS算法

首先用一个具体的例子(图 2)引出需要求解的问题.假定存在一个 OpenMP 的循环包括 37 次迭代,处理器数为 5.图 3(a)给出了块大小为 3 的调度结果.EOSS 调度方法是在最后一轮调度时对每个处理器上的迭代数量进行调整,使得任何两个处理器上的迭代总数相差不超过 1 个迭代,即负载最平衡,如图 3(b)所示.它和调度块为 1 的调度相比(如图 3(c)所示),从每个处理器的所分配的迭代数量上看是相同的.两者不同的是:EOSS 的大部分调度块大小都没有改变,大于 1,保证数据局部性不改变,只是在最后的调度阶段为了平衡负载把调度块大小调

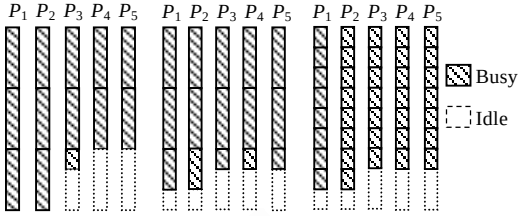
** 该理想条件在文献[6]中已给出定义,具体是指假定每个迭代的执行时间相同,不计算访存时间,不考虑数据在多个处理器上分布造成的数据局部性问题等,只根据处理器所分配的迭代数来计算每个处理器的执行时间.

整小了;而块大小为 1 的调度中块大小始终保持为 1.

```
DOALL i=1,37
  {block}
END DOALL
```

Fig.2 An example of OpenMP

图 2 一个 OpenMP 的例子

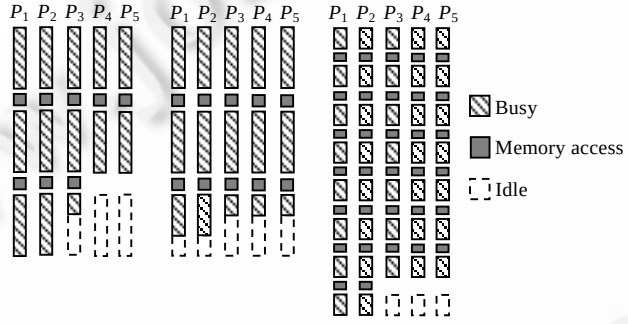


(a) Chunk size=3 (b) EOSS (c) Chunk size=1

Fig.3 Scheduling result of different schedulings without considering memory access latency

图 3 不考虑访存延迟时不同调度方案下的调度结果

我们已经证明[6]:在理想条件下,图 3(b)所示的 EOSS 产生的处理器能量最小;在不考虑数据局部性的影响下,可以认为图 3(c)的调度等同于图 3(b)的调度,即调度块大小为 1 的调度是能量最优的.但是,块大小为 1 时,连续数据跨处理器分布所产生的 cache 空间局部性很差,cache 失效所带来的访存延迟不可忽略.这也是为什么很多时候并不采用块大小为 1 的静态调度的原因.因此,我们在改进的模型中需要考虑适当调大块大小,以增大数据局部性,增加 cache 命中率,减小访存延迟及访存能量消耗.我们在改进的模型中考虑了不同块大小对访存延迟的影响,图 4 为考虑访存延迟条件下图 3 的调度结果.



(a) Chunk size=3 (b) EOSS (c) Chunk size=1

Fig.4 Scheduling result of different schedulings considering memory access latency

图 4 考虑访存延迟时不同调度方案下的调度结果

在图 4 中,两个调度块之间增加了因访存所引发的时间开销,时间开销必然伴随着相应的能量消耗.随着调度块大小的增大,被调度到同一处理器上的连续数据量增大,cache 命中率提高,访存延迟减小,体现在图 4 中调度块之间的间隙减小.当把 cache line 大小的因素考虑加入原来的理想条件中去,EOSS 算法指导下,块大小为 1 的调度不再能获得最大的能量节约.为了平衡访存延迟增加所引发的额外能量消耗,需要对算法进行修正,选择一个更合适的块大小来满足新条件下的能量最优问题.

本文中,能量最优是指选择合适的调度块大小,并结合 DVS 所得到的 OpenMP 循环执行过程中处理器的最小能量消耗.这个能量最优很大程度上取决于如何建模能量(这里的能量是指处理器的能量).

用Γ表示一个静态调度序列,$\Gamma=\{(a_1,f_1),(a_2,f_2),\ldots,(a_p,f_p)\}$.其中,$p$ 是处理器个数,a_i,f_i 分别表示处理器 P_i 的总迭代数和频率.当块大小为 S,处理器处于最高频率时,$\Gamma=\{(a_1,f_{\max}),(a_2,f_{\max}),\ldots,(a_p,f_{\max})\}$,简记为$\Gamma(S)$,此时的并行执行时间记作 $T(\Gamma(S))$.令 $S_0=\left\lceil N/p \right\rceil$,调度$\Gamma(S_0)$的并行执行时间为 $T(\Gamma(S_0))$,下文中用 T_d 标记并行执行时间 $T(\Gamma(S_0))$.IEOSS 以 $T(\Gamma(S_0))\times(1+\beta)$作为截止时间对每个处理器实施 DVS,$\beta$为性能损失度.

最优问题描述为:给定 p 个处理器,一个 DOALL 循环,最外层循环迭代个数为 N.假定每个循环迭代的执行时间相同,以一个最外层迭代作为一个调度单位.目标是寻找一个最优的调度块大小 S^*,使得

$$E_{opt}(S^*)=\min_{S\in\{1,2,\ldots,N\}}E_{opt}(S),$$

其中,$E_{opt}(S)$表示按上述方法进行 DVS 之后的能量值.S^*相应的调度 $\Gamma^*=\{(a_1^*,f_1^*),(a_2^*,f_2^*),\ldots,(a_p^*,f_p^*)\}$ 即为最优调度.

在计算截止时间时,选择 S_0 的值为“平均分”大小 $\left\lceil N/p \right\rceil$.选择其作为基准,具有一定的代表性.

设 L 为 cache line 大小,C 为 cache line 中能容纳的数据量,其值可以通过 cache line 大小除以每个元素的大小获得.例如,一个 cache line 大小为 16 bytes,一个 float 类型的数据占 4 bytes,则该 cache line 可以容纳 4 个 float 类型的数据,C 值为 4.设 S 为静态调度中的调度块大小.设 D_i 为处理器 P_i 上因为访存所引发的额外时间开销.

$$D_i=T_{mem}\times N_{mem}(a_i)\times N_{data}(a_i) \tag{1}$$

其中,T_{mem} 表示处理器一次访存所需要的时间,$N_{mem}(a_i)$为处理器 P_i 在执行 a_i 个迭代过程中针对一个数据元素产生的访存次数,$N_{data}(a_i)$表示处理器 P_i 在执行 a_i 个迭代过程中所涉及到的数据量.假定每次读取内存,都会读取一个 cache line 大小的数据块到 cache 中.如果 $p\times S<C$,则说明 cache line 足够大,能够容纳第 1 次调度中所有 p 个处理器所需同一数据的所有连续值,因此只需访存一次.如果 cache line 再够大(或许处理器数够小),一次访存读取到 cache 中的数据可使后续多个调度块的数据也 cache 命中,具体用 $\left\lceil \frac{C}{p\times S} \right\rceil$ 计数在一个处理器上发生连续 cache 命中的调度次数,$\frac{a_i}{S}$ 表示处理器 P_i 的总调度次数.因此,$\frac{a_i/S}{\left\lceil C/(p\times S)\right\rceil}$ 表示同一数据需访存的次数.还有一种情况需要考虑,即如果 S 足够大,一个 cache line 无法容纳 S 个迭代内的数据,则每次调度都需要多次访存操作,该次数由 $\left\lceil S/C \right\rceil$ 给出,则 $N_{mem}(a_i)$由下列公式给出:

$$N_{mem}(a_i)=\frac{a_i/S}{\left\lceil C/(p\times S)\right\rceil}\times\left\lceil \frac{S}{C} \right\rceil \tag{2}$$

将公式(2)代入公式(1),得到

$$D_i(S)=T_{mem}\times\frac{a_i/S}{\left\lceil C/(p\times S)\right\rceil}\times\left\lceil \frac{S}{C} \right\rceil\times N_{data}(a_i) \tag{3}$$

显然,D_i 是关于块大小 S 的函数,记作 $D_i(S)$.

在调度 Γ 中所有处理器的能量消耗包括以下 3 部分:

$$E(\Gamma)=E_{busy}+E_{idle}+E_{delay} \tag{4}$$

其中,E_{busy} 表示处理器计算所产生的能量消耗,E_{idle} 表示处理器空闲引发的能量消耗,E_{delay} 表示因访存延迟所造成的处理器空转所产生的能量消耗.其中,E_{busy} 可以使用式(5)表示:

$$E_{busy}=\sum_{i=1}^{P}(\delta.f_i^3.T_i^w(a_i,f_i)) \tag{5}$$

其中,δ为功耗和频率之间的比例因子,f_i 表示处理器 P_i 的执行频率,$T_i^w(a_i,f_i)$ 表示处理器 P_i 执行迭代的忙时间.频率调节之前,处理器的忙时间为

$$T_i^w(a_i,f_{\max})=t_0\cdot a_i \tag{6}$$

其中,t_0 表示最高频率下每个迭代的执行时间,单位秒(这里假定每个迭代的执行时间相同).

如前所述,将 $T\left(\Gamma\left(\left\lceil\frac{N}{p}\right\rceil\right)\right)$ 作为并行计算时间,简记为 T_d.定义 α_{idle} 为处理器处于空转时的功耗与最大功耗的比值,$0<\alpha_{idle}<1$.假定处理器的最大功耗为 $\delta\cdot f_{\max}^3$,则处理器空转时的功耗为 $\alpha_{idle}\cdot\delta\cdot f_{\max}^3$.

$$E_{idle}=\sum_{i=1}^{p}(\alpha_{idle}.\delta.f_{\max}^3.(T_d-T_i^w(a_i,f_i))) \tag{7}$$

$$E_{delay}=\sum_{i=1}^{p}(\alpha_{idle}.\delta.f_{\max}^3.D_i) \tag{8}$$

当调度块大小为 S 时,频率调节前的总能量为

$$E_{ori}(S)=\sum_{i=1}^{p}(\delta.f_{\max}^3.T_i^w(a_i,f_{\max}))+\sum_{i=1}^{p}(\alpha_{idle}.\delta.f_{\max}^3.(T_d-T_i^w(a_i,f_{\max})))+\sum_{i=1}^{p}(\alpha_{idle}.\delta.f_{\max}^3.D_i) \tag{9}$$

根据处理器 P_i 的忙时间 $T_i^w(a_i,f_i)$ 和并行计算的截止时间 $T_d\times(1+\beta)$,将 P_i 的频率从 $f_{\max}$ 按比例地降低.

$$f_i=\frac{T_i^w(a_i,f_{\max})}{T_d\times(1+\beta)}\times f_{\max},i=1,2,...,p \tag{10}$$

频率调节后,块大小为 S 的能量计算公式为

$$E_{opt}(S)=\sum_{i=1}^{p}(\delta.f_i^3.T_d\cdot(1+\beta))+\sum_{i=1}^{p}(\alpha_{idle}.\delta.f_{\max}^3.D_i) \tag{11}$$

可知$\exists S^*\in\{1,2,3,\ldots,N\}$,使得

$$E_{opt}(S^*)=\min_{S\in\{1,2,...,N\}}E_{opt}(S) \tag{12}$$

最优块大小 S^*遍历了 S可取的所有范围.我们为初始的静态调度计算出 $E_{ori}(S_0)$.根据公式(12)得出最优能量时的调度块大小 S^*及最优能量值 $E_{opt}(S^*)$,因此可获得取最优块大小 S^*时 IEOSS 算法的能量节约$\Delta E_{save}(S^*)$.计算方法为:

$$\Delta E_{save}(S^*)=\frac{E_{ori}(S_0)-E_{opt}(S^*)}{E_{ori}(S_0)}\times 100\% \tag{13}$$

图 5 给出了 IEOSS 的描述.

Algorithm: Improved Energy-Optimal OpenMP Static Scheduling (IEOSS).
Input: OpenMP loop, chunk size S_0, processor number p, the maximum frequency of processor $f_{\max}$;
Output: find chunk size S^* and scheduling $\Gamma^*=\{(a_1^*,f_1^*),(a_2^*,f_2^*),...,(a_p^*,f_p^*)\}$ which minimizes energy consumption of OpenMP static loop scheduling.

1. minE_{opt}=E_{opt}(1);
2. S^*=1;
3. for S=1 to N
4. calculate $E_{opt}(S)$ according Eq.(11) when chunk size is S after DVS;
5. if ($E_{opt}(S)$<minE_{opt})
6. minE_{opt}=$E_{opt}(S)$;
7. S^*=S;
8. endif
9. end for
10. $\Gamma^*=\{(a_1^*,f_1^*),(a_2^*,f_2^*),...,(a_p^*,f_p^*)\}$.
11. End.

Fig.5 Description of IEOSS algorithm

图 5 IEOSS 算法描述

在该描述中,针对每个循环块,利用公式(12)计算不同调度块大小条件下使用 DVS 方法的能量值,并比较这些能量值,选择一个具有最小能量值的调度块大小 S^*.这里涉及的几个参数:迭代数 N,$N_{data}(a_i)$,cache line 大小,处理器数目 p,α_{idle} 都可以在编译时确定.此外,并行循环执行时间可通过编译预估获得.因此在编译过程中,公式(12)的值可以计算得到.算法的执行时间和循环迭代的数量以及处理器数有关.从图 5 的算法描述可以得到,对

于每一个调度块大小,算法的执行时间和公式(11)的执行时间相关.公式(11)中,p 为处理器数,公式(11)执行 1 次,共有 $10p$ 个浮点乘操作、$2(p-1)$个浮点加操作.当处理器固定后,每个操作的时间是固定的.因此对于整个循环来说,算法执行时间等于迭代次数 N 乘以公式(11)执行 1 次的时间.

3 实 验

本节对 IEOSS 算法进行详细的模拟实验评测,具体做法是:模拟器模拟程序的运行,得到每个 OpenMP 循环段的所需设置的最优块大小及运行该段程序所需设置的频率值,从而获得 OpenMP 循环的新调度:

$$\Gamma^* = \{(a_1^*, f_1^*),(a_2^*, f_2^*),...,(a_p^*, f_p^*)\}.$$

由于模拟环境可模拟程序运行时的能量消耗,因此可得到模拟环境下 IEOSS 算法的能量结果.第 3.1 节介绍实验平台,测试程序及实验方法.第 3.2 节给出详细的实验评测结果及分析.

3.1 实验环境

我们通过如下方法实现 Simplescalar[26]对 OpenMP 程序的支持:首先,Simplescalar 扩展针对程序的并行循环部分.对于程序中的循环语句,假定其已经确定指导命令,并定义好调度块大小.根据 OpenMP 指导命令以及处理器数量,可以采用下面的方法使 Simplescalar 由模拟单个处理器运行变为模拟多个处理器运行:由于使用静态调度,每个处理器所分配的迭代(包括迭代上下界、迭代数)是已知的,Simplescalar 只需将每个处理器所分配的任务(若干循环迭代)模拟执行 1 遍,所有处理器的模拟执行构成了并行循环的模拟结果.修改调度配置,重复上述过程,可以得到不同调度条件下的并行循环模拟结果.该模拟方式降低了模拟器扩展难度,将迭代分配工作交由程序员手工完成.由于 OpenMP 并行循环特点,使得我们无需考虑分配在不同处理器上循环迭代间的依赖关系.扩展后的并行循环模拟并没有修改程序的结构,程序的运行依然按照原有的逻辑继续运行,模拟结果不会对模拟的正确性产生影响.我们选择了 Wattch[27]功耗模型进行能量模拟.在实验中,我们假定频率可以连续地调节,范围从 300MHz~1GHz.

为了验证算法的有效性,我们使用了 5 个 NPB3.2-OMP 核心程序[28]:IS,EP,FT,CG 和 MG.NPB3.2-OMP 是 NPB3.2 测试程序的 OpenMP 实现,采用具有一定代表性 C 级规模进行测试,既能满足测试对数据规模的要求,又能满足模拟对时间的要求.表 2 给出了每个程序的相关统计信息,其中,迭代是指最外层循环迭代.

Table 1 Parameters in experiments

表 1 实验中所涉及的模拟参数

Parameter	Value
The maximum frequency/voltage	1GHz/1.9V
The minimum frequency	300MHz
Frequency/voltage switch overhead	2μs/2μJ
Restart overhead	50μs/50μJ
L_1 latency	1cycle

Table 2 Statistics of five NPB3.2-OMP kernel programs

表 2 5 个 NPB3.2-OMP 核心程序的统计信息

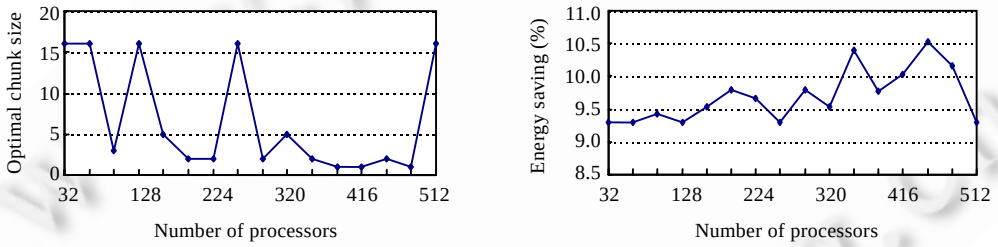
Program	Number of parallel regions	Max number of iterations	Min number of iterations
IS	24	268 435 456	134 217 728
EP	1	65 536	65 536
FT	38	1 024	512
CG	2 918	149 907	149 800
MG	770	139 264	512

为了简化能量模拟中的复杂度,我们将每次的电压/频率转换的时间及能量开销固定在 2μs 和 2μJ.

3.2 实验结果

3.2.1 EP 程序的测试结果

由表 2 的统计可知,EP 程序仅包含一个 OpenMP 循环段,因此我们将 EP 程序作为重点分析对象,比较在不同的处理器和 cache line 配置下 IEOSS 算法可以获得的能量优化效果.首先设定 cache line 大小为 64 字节,比较不同处理器数目下的 IEOSS 能量优化结果.如图 6 所示,处理器数目变化范围在 32~512,性能损失度β等于 5%.图 6(a)给出了在不同处理器数目情况下算法 IEOSS 获得的最优静态块大小S^*.根据公式(12)获得 IEOSS 算法计算的最优能量结果$E_{opt}(S^*)$.根据公式(13)计算出算法 IEOSS 获得的能量节约$\Delta E_{save}(S^*)$,如图 6(b)所示,能量节约范围为 9.30%~10.52%.能量节约的曲线具有一定的规律性,其中有几个明显的"能量最低点":处理器数为 32,64,128, 256,512 时的能量节约值,且均为 9.30%.除了这 5 个能量最低值之外,在其他的处理器数情况下能量节约值均明显大于 9.30%.这 5 个"能量最低点"的特殊性在于 32,64,128,256,512 均能被 EP 循环的迭代数整除,即 $p|N$.整除表明此时处理器间负载最均衡,能量节约完全取决于性能损失度β的大小.负载均衡使得条件 $\forall i, i\in\{1,2,\ldots,p\}, T_i^w(a_i, f_{\max}) = T_d$ 成立.依据公式(10),每个处理器的频率均由 $f_{\max}$ 变为 $\frac{1}{1+\beta} f_{\max}$,执行时间由 T_d 延长为 $T_d\times(1+\beta)$.因此,优化后的能量节约为 $1-\frac{1}{(1+\beta)^2}$.当β=5%时,能量节约为 9.30%.除了这 5 个处理器数之外,其他的处理器数不能够被 65536 整除,负载不均衡,因此能量节约比例大于 9.30%.从不同处理器数目下能量节约的变化趋势整体看来,能量节约比例变化趋势相对处理器变化平缓得多.这是因为对应曲线上不同的处理器数目,基准能量值 $E_{ori}(S_0)$时刻都在变化.如公式(13)所示,分母 $E_{ori}(S_0)$在不同的处理器数目下是不同的,p 越大,S_0 越小.尽管直观上越多的处理器带来的负载不平衡的可能性越大,能量节约的机会越大,但是不同处理器数目下基准能量值的差异使得这种可比性降低.从图 6(b)反映出算法 IEOSS 的可扩展性是好的.



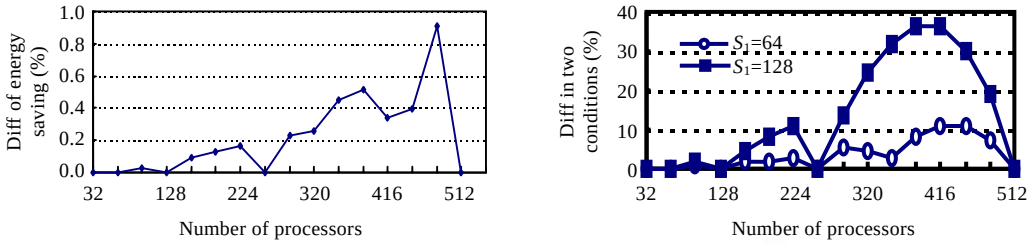
(a) Optimal chunk size
(a) 最优高度块大小

(b) $\Delta E_{save}(S^*)$
(b) $\Delta E_{save}(S^*)$

Fig.6 Impact of the vary processors on optimal chunk size and energy savings (performance loss=5%) (EP)

图 6 处理器数目变化对最优块大小及能量节约值的影响(性能损失 5%)(EP)

图 6 假定 cache line 大小为 64 字节,且浮点类型数占 4 字节,即每个 cache line 可以存放 16 个浮点类型数.如果将 S_1=16 作为块大小替换 IEOSS 算法得到的最优块大小,可以获得对应的能量优化值 $E_{opt}(S_1)$.根据公式(13),记此时的能量节约为$\Delta E_{save}(S_1)$.图 7(a)比较了分别采用 S^*和 S_1 为调度块时的能量节约.从比较结果看,调度块为 S^*时能量节约要大,但二者差距不大,这主要由于 S_1=16 和最优块大小 S^*比较接近.当进一步改变块大小,取更大的块,例如 64,128,则可以明显看到最优块对能量节约的贡献,如图 7(b)所示.



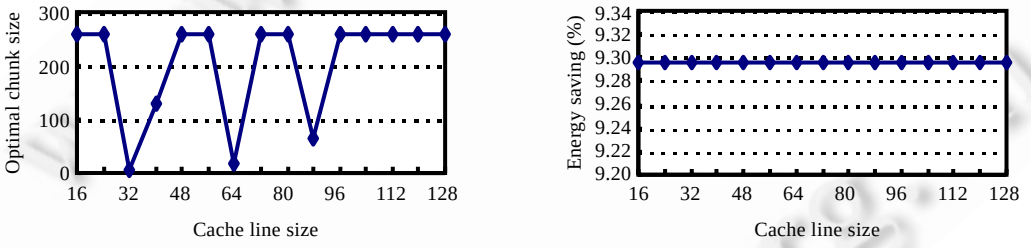
(a) $\Delta E_{save}(S^*)-\Delta E_{save}(16)$ (b) $\Delta E_{save}(S^*)-\Delta E_{save}(S_1)$

Fig.7 Impact of the vary chunk size on energy savings (performance loss=5%) (EP)

图 7 对比不同的块对能量节约的影响(性能损失 5%)(EP)

同理,分析 cache line 变化对 IEOSS 能量优化效果的影响.如图 8 所示,固定处理器数目为 256,变化 cache line 大小可得到 IEOSS 的优化结果.这里性能损失度β仍旧定义为 5%.从图 8 中可看出,在大部分 cache line 大小情况下,基于初始块大小 S_0,IEOSS 可获得 9.30%的能量节约,cache line 的变化对能量节约没有什么影响.原因解释:在该图中,处理器数量和循环迭代的数量是固定的,经过 DVS,处理器能量由两部分组成,E_{delay} 和 E_{busy},无空转能量 E_{idle}.其中,cache line 变化主要影响 E_{delay}.在 EP 程序中,E_{delay} 占总能量的比重较小,所以 cache line 变化对总能量节约的影响不能够反映出来,因而对最优块的选择不会产生太大影响.

从图 8(a)可以看出,在大部分情况下,最优块大小保持在 256,EP 程序中共包括了 65536 个循环迭代,图 8(a)中的处理器数目固定为 256.此时,处理器数可以整除迭代数,每个处理器负载相同.由于 E_{delay} 在全部能量消耗中所占比例小,并不会对全部能量消耗产生大的影响,所以大部分情况下最优块大小仍然可以取到 256.但在某些 cache line 大小时,E_{delay} 还是影响了最优块大小的选择,使得少数点的最优块大小不是 256.



(a) Optimal chunk size
(a) 最优调度块大小

(b) $\Delta E_{save}(S^*)$
(b) $\Delta E_{save}(S^*)$

Fig.8 Impact of vary cache line on optimal chunk size and energy savings (performance loss=5%) (EP)

图 8 Cache line 变化对最优块大小及能量节约的影响(性能损失 5%)(EP)

3.2.2 其他 4 个程序的测试结果

类似地,图 9 给出了 NPB 其他 4 个程序的能量结果.表 2 除了 EP 程序之外,其他每个程序中包含许多个待优化的并行循环,根据 IEOSS 算法,每个并行循环都会获得一个最优块大小及对应的能量节约值.考虑到数据量太大,图中没有一一给出所有并行循环段的最优块大小设置及能量节约.每个程序的能量统计由该程序中所有循环在相应最优调度块大小条件下的能量消耗累加得到,能量节约比值的计算方法同 EP 程序.图 9 给出的结果是在允许 5%的性能损失条件下获得的.其中,左半部分表示 L=64 字节时的结果,右半部分表示 p=256 时的结果.以 p=480,L=64 字节为例,对比 OpenMP 缺省的块调度的能量消耗,IEOSS 算法可使 EP,IS,FT,CG,MG 程序的并行循环能量消耗分别减少 10.15%,4.49%,81.66%,2.32%,10.11%(EP 的结果如图 6(b)所示).图 9 的处理器变化曲线不是完全和图 6(b)曲线不是完全一致,原因是 IS,FT,CG,MG 程序包含多个并行段,而每个并行段的最外层循环

迭代数和给出的处理器数不一定满足整除关系.从现有的处理器变化曲线来看,并不是处理器数目越多,能量节约的效果越好,它取决于循环迭代数和处理器之间的关系.

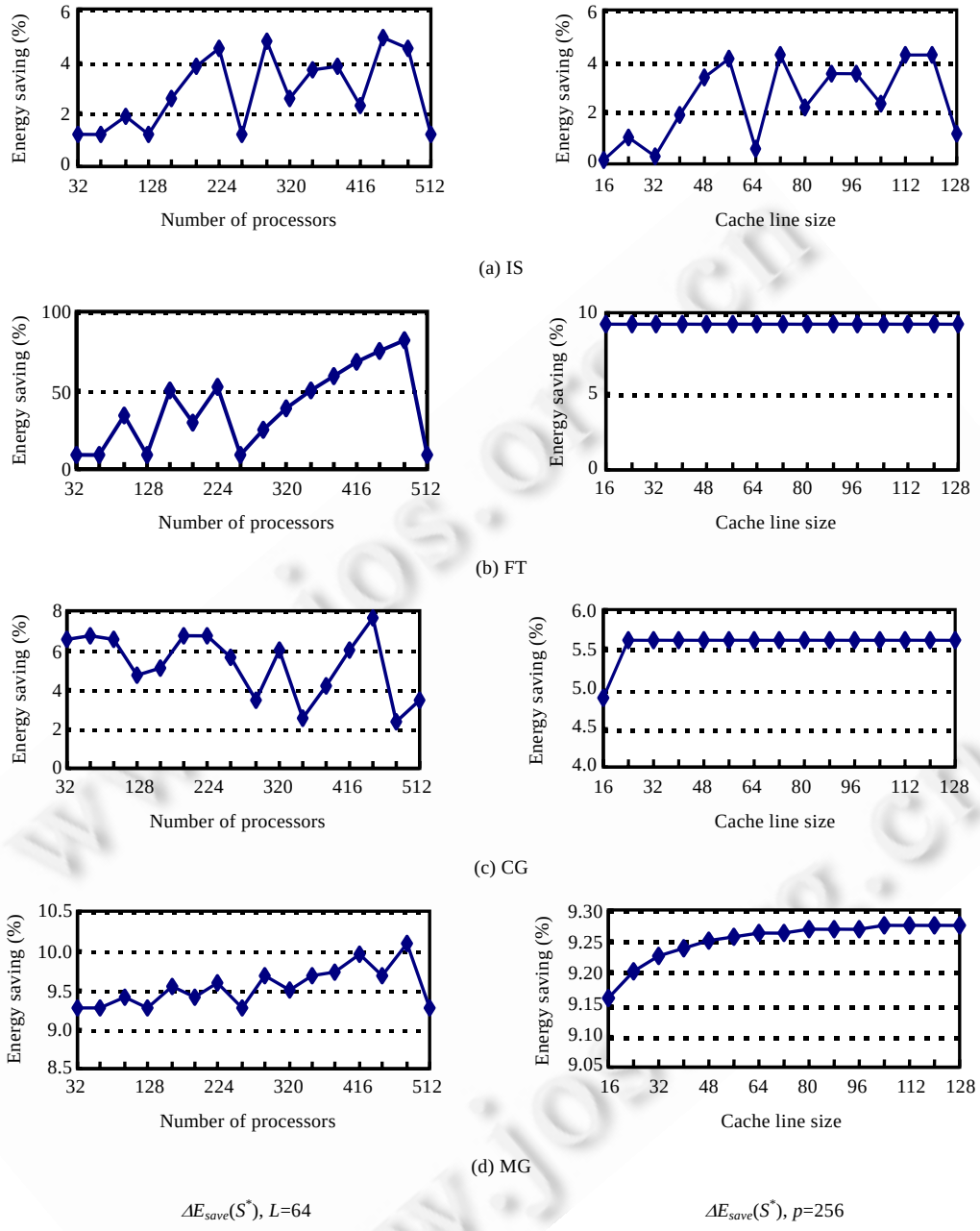


Fig.9 Experimental results of other four NPB benchmarks (performance loss=5%)

图 9 NPB 其他 4 个程序测试结果(性能损失 5%)

对比 Kadayif[3]的实验结果,他们选择的 benchmark 是一组数据密集型代码,并且所指的能量包括数据通路 datapath、指令/数据 cache、主存以及连接多核的总线.他们在实验中分别给出了整数线性规划方法获得的性能及能量优化的效果.平均的性能提高有 6.9%;平均的能量改进更大,较大的 3 个能量节约为 aps 程序 35.3%,lms

程序 52.1%,tsf 程序 54.4%,此时处理器核数为 8.而我们在实验中选择了科学计算的典型 benchmark——NPB 程序,且对比了处理器在 32~512 变化下 IEOSS 算法的能量优化效果.最大的能量节约由 FT 程序在 480 个处理器时获得,为 81.66%.5 个 NPB 程序在 480 个处理器时平均的能量节约也达到 21.75%. Kadayif 的其他相关工作[4,5]和文献[3]是属于同一类问题,这里不再重复比较了.

4 结束语

本文在原有 EOSS 算法的基础上给出了扩展条件下的 OpenMP 静态调度能量优化算法——改进的能量最优 OpenMP 静态调度算法 IEOSS.算法 IEOSS 改进了原有算法 EOSS 不考虑访存延迟对循环调度性能和能量影响的不足,在能量模型中加入了 cache line 大小、访存延迟及访存能量因子.我们选择了 5 个 NPB3.2 程序进行了详细的实验,实验结果验证了算法 IEOSS 在 DVS 中加入了最优块大小的选择是能够明显改进能量优化效果的;同时,我们比较了不同处理器数目以及不同 cache line 配置下算法 IEOSS 的效果.

致谢 感谢审稿人的修改建议与辛苦工作.感谢黄春博士对文章提出的修改建议.

References:

[1] Top 500 list. http://www.top500.org

[2] Yelick K. Ten ways to waste a parallel computer. In: Proc. of the 36th Annual Int'l Symp. on Computer Architecture (ISCA 2009). 2009. [doi: 10.1145/1555754.1555755]

[3] Kadayif I, Kandemir M, Sezer U. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. In: Proc. of the 39th IEEE/ACM Design Automation Conf. (DAC 2002). New Orleans: ACM, 2002. 703−708. [doi: 10.1109/DAC.2002.1012715]

[4] Kadayif I, Kandemir M, Vijaykrishnan N, Irwin MJ, Kolcu I. Exploiting processor workload heterogeneity for reducing energy consumption in chip multiprocessors. In: Proc. of the Conf. on Design, Automation and Test in Europe (DATE 2004). Paris: IEEE Computer Society, 2004. 1158−1163. [doi: 10.1109/DATE.2004.1269048]

[5] Kadayif I, Kandemir M, Karakoy M. An energy saving strategy based on adaptive loop parallelization. In: Proc. of the Design Automation Conf. (DAC 2002). New Orleans: ACM, 2002. 195−200. [doi: 10.1109/DAC.2002.1012619]

[6] Dong Y, Chen J, Yang XJ, Deng L, Zhang XM. Energy-Oriented OpenMP parallel loop scheduling. In: Proc. of the IEEE Int'l Symp. on Parallel and Distributed Processing with Applications (ISPA 2008). Sydney: IEEE, 2008. 162−169. [doi: 10.1109/ISPA.2008.68]

[7] Chen J, Dong Y, Yang XJ, Wang PF. Energy-Constrained OpenMP static loop scheduling. In: Proc. of the 10th IEEE Int'l Conf. on High Performance Computing and Communications (HPCC 2008). 2008. 139−146. [doi: 10.1109/HPCC.2008.132]

[8] Dong Y, Chen J, Yang XJ, Yang CQ, Peng L. Low power optimization for MPI collective operations. In: Proc. of the 9th Int'l Conf. for Young Computer Scientists (ICYCS 2008). Zhangjiajie: IEEE Computer Society, 2008. 1047−1052. [doi: 10.1109/ICYCS.2008.500]

[9] Yan CY, Che JJ, Kuo TW. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In: Proc. of the Conf. on Design, Automation and Test in Europe—Vol.1 (DATE 2005). IEEE Computer Society, 2005. 468−473. [doi: 10.1109/DATE.2005.51]

[10] Chang PC, Wu IW, Shann JJ, Chung CP. ETAHM: An energy-aware task allocation algorithm for heterogeneous multiprocessor. In: Fix L, ed. Proc. of the Design Automation Conf. (DAC 2008). 2008. 776−779.

[11] Coskun AK, Strong R, Tullsen DM, Rosing TS. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In: Douceur JR, ed. Proc. of the 11th Int'l Joint Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance). Seattle: ACM, 2009. 169−180. [doi: 10.1145/1555349.1555369]

[12] Teodorescu R, Torrellas J. Variation-Aware application scheduling and power management for chip multiprocessors. In: Proc. of the 35th Int'l Symp. on Computer Architecture (ISCA 2008). Beijing: IEEE Computer Society, 2008. 363−374. [doi: 10.1109/ISCA.2008.40]

[13] Ranga KK, We GY, Brooks D. Thread motion: Fine-grained power management for multi-core systems. In: Keckler SW, ed. Proc. of the 36th Annual Int'l Symp. on Computer Architecture (ISCA 2009). 2009. 302−313. [doi: 10.1145/1555754.1555793]

[14] Zhan SS, Chatha KS. Automated techniques for energy efficient scheduling on homogeneous and heterogeneous chip multi-processor architectures. In: Proc. of the 2008 Asia and South Pacific Design Automation Conf. Seoul: IEEE Computer Society, 2008. 61−66. [doi: 10.1109/ASPDAC.2008.4484026]

[15] Elyada A, Ginosar R, Weiser U. Low-Complexity policies for energy-performance tradeoff in chip-multi-processors. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2008,16(9):1243−1248. [doi: 10.1109/TVLSI.2008.2000867]

[16] Anderson JH, Baruah SK. Energy-Efficient synthesis of periodic task systems upon identical multiprocessor platforms. In: Proc. of the 24th Int'l Conf. on Distributed Computing Systems (ICDCS 2004). Tokyo: IEEE Computer Society, 2004. 428−435. [doi: 10.1109/ICDCS.2004.1281609]

[17] Chen JJ, Hsu HR, Chuang KH, Yang CL, Pang AC, Kuo TW. Multiprocessor energy-efficient scheduling with task migration conside- rations. In: Proc. of the 16th Euromicro Conf. on Real-Time Systems (ECRTS 2004). IEEE Computer Society, 2004. 101−108. [doi: 10.1109/EMRTS.2004.1311011]

[18] Gruian F. System-Level design methods for low-energy architectures containing variable voltage processors. In: Proc. of the 1st Int'l Workshop on Power-Aware Computer Systems-Revised Papers. Springer-Verlag, 2001. 1−12.

[19] Zhan YM, Hu XB, Chen DZ. Task scheduling and voltage selection for energy minimization. In: Proc. of the 39th Annual Design Automation Conf. New Orleans: ACM, 2002. 183−188. [doi: 10.1109/DAC.2002.1012617]

[20] Li J, Martinez JF. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In: Proc. of the Int'l Symp. on High Performance Computer Architecture (HPCA 2006). Austin: IEEE Computer Society, 2006. 77−87. [doi: 10.1109/HPCA.2006.1598114]

[21] Chen Y, Shao ZL, Zhuge QF, Xue C, Xiao B, Sha EHM. Minimizing energy via loop scheduling and DVS for multi-core embedded systems. In: Proc. of the 2005 11th Int'l Conf. on Parallel and Distributed Systems (ICPADS 2005). Fuduoka: IEEE Computer Society, 2005. 2−6. [doi: 10.1109/ICPADS.2005.196]

[22] Kappiah N, Freeh VW, Lowenthal DK. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In: Proc. of the ACM/IEEE Conf. on High Performance Networking and Computing (SC 2005). Seattle: IEEE Computer Society, 2005. 33. [doi: 10.1109/SC.2005.39]

[23] Freeh VW, Lowenthal DK. Using multiple energy gears in MPI programs on a power-scalable cluster. In: Pingali K, ed. Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP 2005). Chicago: ACM, 2005. 164−173. [doi: 10.1145/1065944.1065967]

[24] Lim MY, Freeh VW, Lowenthal DK. Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In: Proc. of the ACM/IEEE Conf. on High Performance Networking and Computing (SC 2006). Tampa: ACM, 2006. 14. [doi: 10.1109/SC.2006.11]

[25] Yi HZ. Low-Power techniques for architecture and compiler optimization [Ph.D. Thesis]. Changsha: National University of Defense Technology, 2006 (in Chinese with English abstract).

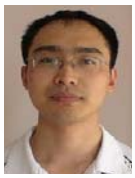
[26] Burge D, Austin TM. The SimpleScalar tool set, version 2.0. Technical Report, CS-TR-1342, Madison: University of Wisconsin, 1997. [doi: 10.1145/268806.268810]

[27] Brooks D, Tiwari V, Martonosi M. Wattch: A framework for architectural-level power analysis and optimizations. In: Proc. of the 27th Int'l Symp. on Computer Architecture (ISCA 2000). 2000. 83−94. [doi: 10.1109/ISCA.2000.854380]

[28] NAS parallel benchmarks. http://www.nas.nasa.gov/Resources/Software/npb_changes.html

附中文参考文献:

[25] 易会战.低功耗技术研究——体系结构和编译优化[博士学位论文].长沙:国防科学技术大学,2006.



董勇(1980－),男,山东宁阳人,博士生,助理研究员,主要研究领域为并行系统功耗优化,并行文件系统.

陈娟(1980－),女,博士,助理研究员,主要研究领域为并行系统功耗优化,高性能编译,GPGPU.

杨学军(1963－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为并行体系结构,低功耗编译,流计算,容错计算.