低功耗编译的若干相关技术

连瑞琦　张兆庆

（中国科学院计算技术研究所系统结构室　北京100080）

摘　要　本文综述低功耗编译相关技术。首先简要介绍了若干低功耗领域的基本术语之后，总结出了可用于降低功耗的三类编译手段：选取有助于降低功耗的传统优化，考虑功耗因素改造现有优化技术和通过编译制导配合硬件降低功耗。其次简单介绍了多线程系统和功耗模型的相关研究，最后，文章给出了低功耗编译领域研究的若干有潜力的方面，供有志进行这方面研究的研究人员参考。

关键词　低功耗，编译，多线程

Low-Power Compilation Related Techniques

LIAN Rui-Qi　ZHANG Zhao-Qing

(Computer Architecture Laboratory, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

Abstract　This paper gives a survey of some low-power compilation related techniques. After giving some basic conceptions on low-power compilation, this paper gives three primary methods to reduce power cost, including leveraging the traditional optimization which is helpful to reduce power cost, improving some traditional optimizations while considering the fact of power and using the compilation directives to help the hardware to reduce power cost. Next, this paper introduces some related researches on multi-thread system and power-cost model. Finally, the paper summarizes the potential field on lower-power compilation for reference.

Keywords　Low-power, Compilation, Multi-thread

1　引言

在微处理器的设计过程中，功耗问题已经成为一个必须考虑的重要问题。一般来说，在下面两种情况下，需要特别考虑处理器的功率和能量的消耗问题：一是在移动芯片领域，由于电池的电量有限，因此需要考虑功耗问题。二是在高性能处理器领域，虽然，高性能处理器的能量供应一般不受限制，但是，时钟频率的持续增加致使功耗已经达到了当前封装(package)技术的极限，所以，在设计高性能处理器时，必须进行性能和功耗的权衡。

低功耗芯片的设计涉及到硬件设计，制造工艺，处理器的体系结构等诸多方面的技术，本文的主旨是希望能够较为全面地概括低功耗编译相关的问题。下面首先介绍一些和功耗相关的基本术语。

·*能量和功率*　功率是单位时间内所消耗的能量。在移动芯片领域，这二者是同等重要的，都直接关系到电池寿命的长短。在高性能处理器芯片领域，功耗则显得更为重要一些。因为，对于这个领域的芯片，电量供应一般是比较充足的，但是，过高的功率消耗则几乎已经达到了电路封装技术的上限，因而芯片设计者必须考虑功率消耗和性能之间的权衡问题。

·*单步功率，峰值功率和平均功率*　单步功率表示连续的时钟周期之间平均功率的差，在微体系结构级，它可以代表电感噪声(inductive noise)Ldt/di[7]，电感噪声造成的电压急冲会导致时序或逻辑上的错误。峰值功率表示一个给定程序的执行过程中最大的功率的消耗，它和芯片的温度密切相关。对于高性能处理器而言，单步功率和峰值功率是非常重要的设计限制，远比平均功率重要。在超长指令字处理器中，单步功率和峰值功率很大程度取决于指令调度的结果。

2　降低功耗的若干编译技术

从总体上讲，降低功耗的技术可以分为硬件方面的，操作系统方面的和编译方面的优化。但是，在更多的情况下，是通过多方面的技术结合来达到降低功耗的目的。现在，降低微处理器芯片的功耗的行之有效的技术有：缩放电压，动态缩小时钟频率，低功耗电路设计，以及体系结构层次的优化等。下面我们着重介绍和编译有关的技术。

2.1　降低功耗的编译优化

编译优化中，能删除原代码中的冗余操作的优化无疑对降低功耗是有好处的。但是多数编译优化技术对功耗的影响还没有得到全面、合理的评价。事实上，对编译优化的科学评价，对于功耗敏感的编译器中优化种类的选择，现有优化技术面向低功耗的改进和提出新的降低功耗的编译优化都是十分有用的。文[3]通过分析和实验给出了一些高层循环优化对功耗的影响。

2.2.1　*循环优化对功耗的影响*　一般来说，编译优化在提高性能的同时，对于降低功耗可以分为两个方面：一方面因为这些优化能使整体执行时间的减少，所以从这方面来说对降低功耗是有帮助的。另一方面，某些优化对降低功耗又有不利的影响，如开发指令级并行的某些多数优化都会增加单位时间的功耗。所以，对各种优化对功耗的影响的评价十分有助于在进行优化时在提供性能和增加功耗二者之间做权衡。高层循环优化(如循环置换，循环分块等)和低层循环优化(如循环展开，软流水等)对于功耗的影响是紧密相关的，所以不应当单独地评价某种优化的作用，而是应当综合评价这些优化

的整体效果。

概括地说，结论是这样的：

循环展开因为能充分发掘不同迭代之间的指令级并行性，并且能减少循环控制语句的执行次数，所以能减少整体的执行时间，从而减少功耗。

软流水也可以通过减少整体的执行时间而对降低功耗有帮助。但是，实验证明，软流水对功耗的影响很大程度上依赖于它之前执行的高层循环变换。

循环置换，循环分块，循环合并等高层优化能够明显地降低功耗，因为这些优化既能减少总体的执行时间，又能改善cache的局部性。

2.2 对现有编译优化技术的改进

有些编译优化技术对于提高性能是有好处的，但是却会增加功率的损耗，这些优化包括：指令机并行编译中的控制/数据投机(control/data speculation)，条件转换等等。在需要考虑功耗的情况下，这些编译优化技术需要进行改造。文献中已经出现了在其中某些算法中加入功耗方面的考虑，从而得到了性能和功耗都比较理想的结果。文[6]就是这样的一个例子。

文[6]中提出一种用于VLIW的高性能处理器的功率敏感的模调度算法。概括地讲，该算法是在基本不损失性能的情况下，通过构造一个在各个周期更加平衡的调度结果来减少单步功率和峰值功率。

文[6]是以Rau[8]的迭代模调度算法为原型做了适当的修改。原型算法可以描述为：

```
procedure IMS
    II := MII();/* initialize the candidate II to MII */
    while ( FINDFLATSCHEDULE(II)!=SUCCESS)
        II = II +1;
    end procedure
    function FINDFLATSCHEDULE(II)
        /* compute the priority of each operations */
        COMPUTEPRIORITY();
        /* repeat picking the highest priority operation and select-
         ing the best desirable time slot at which the operation is to
         be scheduled until all operations have been scheduled. */
        while ( some operation is not scheduled)
            /* pick the highest priority operation */
            CurrOper:= HIGHESTPRIORITYOPERATION();
            /* compute the time bounds in which the selected op-
             eration can be scheduled satisfying the dependence
             constraint */
            (MinTime, MaxTime): = COMPUTESLACK (Cur-
            rOper);
            /* select the best desirable time slot */
            TimeSlot: = FINDTIMESLOT (CurrOper, MinTime,
            MaxTime);
            /* schedule the operation at TimeSlot */
            SCHEDULEOPERTION(CurrOper, TimeSlot);
        end while
```

概括地讲，该算法首先把启动间隔初始化为最小的可能值，然后逐步增大这个值，直到得到一个成功的软流水调度方案为止。调度的过程是不断地选择优先级最高的操作进行调度，并努力把这个操作调度到最佳的时间槽内。为了加入功耗方面的考虑，文[6]对该算法进行适当的改进，改进的主导思想是：在不牺牲性能的情况下，尽可能使各个时钟周期之间的功耗平衡分布，以减少单步功率和峰值功率。具体地讲，进行了如下两方面的改进：

1)优先函数的改进　原始的算法是运用关键的依赖环长度作为优先函数，而改进后的算法将选择可调度区间(由函数COMPUTESLACK计算得到)宽度的倒数作为优先函数，也就是说，可调度区间越窄的操作将越早得到调度。

2)时间槽选择的改进　在为每个操作选择时间槽时，改进后的算法引入的软流水循环中功耗的平坦函数来指导调度。这里的平坦函数是这样定义的：

首先，如果用 $p(op,i)$ 来表示 op 的第 i 个流水线阶段的功耗，那么，给定执行路径 T 的第 i 周期的功耗 P_i 可表示为：

$$P_i=\sum_{j=1}^{ns}\sum_{op\in T_{i-j+1}}p(op,i)$$

其中，ns 表示 op 的流水线阶段的个数，T_i 表示 T 的第 i 个周期所执行的指令中的操作集合。这样，软流水循环内第 i 个时间槽的功耗就可以表示为：

$$P_{Lsp,i}=\sum_{j=1}^{ns}\sum_{op\in O(Lsp,t),\,where\,t=(i-j+1)\bmod II}p(op,i)$$

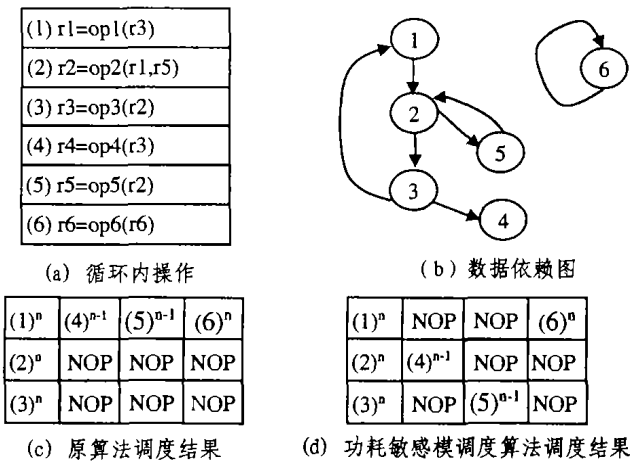
其中，Lsp 表示软流水循环，$O(Lsp,t)$ 表示这个软流水循环中的第 t 个时间槽。显然，当软流水循环的每个时间槽的功耗都等于 $P_{Lsp,(ideal)}$ 时，这个软流水循环具有最小的峰值功率和单步功率，其中，$P_{Lsp,(ideal)}$ 由下面的公式给出：

$$P_{Lsp(ideal)}=[\sum_{op\in\bigcup_{k=0}^{II-1}O(Lsp,k)}\sum_{j=1}^{ns}p(op,j)]/II$$

此时，软流水循环的单步功率为0，峰值功率为 $P_{Lsp,(ideal)}$。改进这个算法的目的是要得到一个调度方案，在这个调度方案中，$P_{Lsp,i}$ 尽可能地平坦。所以，接下来还需要引入一个平坦函数 $F(Lsp)$：

$$F(Lsp)=\sum_{i=0}^{II-1}[P_{Lsp,i}-P_{Lsp,(ideal)}]^2$$

有了这样一个平坦函数，改进后的算法在调度时，为每个操作选择时间槽的标准是不增大或尽可能少地增大平坦函数 $F(L^{lp})$。这样，就可以在考虑调度性能的同时，尽可能地使各个周期之间的功率消耗是平衡的。图1是一个例子，用来说明这个算法在功耗控制方面的优越性。



(a) 循环内操作　(b) 数据依赖图

(c) 原算法调度结果　(d) 功耗敏感模调度算法调度结果

图1　功耗敏感模调度算法实例

2.3 编译器配合硬件中的降低功耗优化

在一些情况下，编译器虽然不能直接起到降低功耗的作用，但是，编译器凭借可以进行大范围的分析，可以进行程序变换等优点辅助硬件的降低功耗的优化，提高硬件优化的性能。下面是这方面的两个例子。

2.3.1 *编译制导的动态缩放电压和频率*　CPU中动态缩放电压和频率是最有效的降低功耗的手段之一。这种优化是针对内存和CPU速度不匹配的情况。在内存无法供给CPU足够的数据时，会导致CPU较长时间的停顿等待，此时缩小电压级别和时钟频率，可以减少能量和功率的消耗。如果在运行时靠硬件识别缩放的机会，其开销会非常大，而且无法进行全局的分析。因此，靠编译器来识别这样的优化机会是十分理想的，而且编译器还能依靠程序变换制造更多的优化机会。

文[12]中提出的方案是：在编译时识别有优化潜力的缩放电压和功率的机会，并通过制导的方式指定需要的时钟频率和电压等级，最后由操作系统和硬件配合，执行电压和时钟频率的缩放。该文指出，这样做可以在基本不影响性能的情况下，大幅度地降低能量和功率的消耗。

文[12]首先提出了一种根据性能损失程度指数 d 来决定CPU的减慢指数 δ 的方法，如下：

$$1 \leqslant \delta \leqslant 1 + \min(d/W_c, W_m, W_m/W_b) \quad (1)$$

访存延迟(memory latency)可以被 δ 整除 (2)

其中，W_b，W_c，W_m 分别代表CPU(包括访问一级和二级cache)和访存重叠的时间，访存不重叠的时间和由于等待访存结果所停顿的时间。d 表示性能损失的程度，例如，我们希望在此过程中性能损失不超过1%，那么在按照(1)(2)寻找 δ 的过程中，我们就可以使用1%作为 d 的值。文中还通过实验证明，对于整个程序实施电压和频率的缩放并没有好处，而只对于某些有潜力的区域实施优化才会真正降低能量消耗。那么，下面的问题就是如何选定这些有优化潜力的区域了。

文中给出了这样两个原则：(1)为了避免频繁转换电压和频率所带来的开销，选择算法应当选择程序中最有优化潜力的部分区域进行优化；(2)对于不同的区域，应当选择不同的减慢指数，这样才能在性能损失程度指数范围内，最大限度地节省能量。文[12]给出的算法只解决了以上第一个问题，并没有解决其中第二个问题。算法的框架如下：

(1)选出进行动态电压/频率缩放的所有候选区 R_i
(2)对于所有候选区域
  (a)利用程序执行的跟踪信息决定 $W_c^{R_i}$，$W_m^{R_i}$，$W_b^{R_i}$
  (b)计算减慢指数 δ_i
(3)选择预计受益最大的区域
(4)插入设置电压/频率的制导指令

具体地说，算法以自定向下的方式考察每个区域，估计每个区域可以节省多少能量，然后选定优化潜力最大的一个或几个区域。另外，文中指出区域应当足够大，以至于得到的好处能抵销电压和频率的转换带来的开销。文中给出的算法是以过程调用、循环嵌套和if-then-else结构作为候选区域的。

2.3.2 *profile信息驱动的代码执行* 为了降低功耗，文[13]给出了如下的观察和分析结果：对于数据路径占主导的体系结构(VLIW)和面向应用的体系结构，可以依靠动态调节执行速率来降低能量的消耗；对于控制占主导的体系结构(带乱序执行和投机执行的超标量结构)，可以通过变化取指速率来降低能量的消耗。

基于以上结论，文[13]提出了基于profiling结果的优化方案。简单地说，就是对于要执行的程序，模拟一个典型的输入流 K 次(K 是体系结构能达到的最快速率)，这其间从1至 K 变换取指速率和执行速率，并以基本块为单位收集每次执行时系统所消耗的能量。然后，根据所收集到的信息，选取基本块的最佳取指和执行速率，以某种形式标注在基本块上。当然，对于这种优化，操作系统和硬件的支持是必不可少的，必须能够支持根据代码中给出的速率适时地变动取指或执行速率。

3 低功耗和多线程

对于低功耗或对功耗有限制的设备而言，多线程是很有用的，因为多线程是通过允许多种线程的方式来提高并行性，而不是依赖投机的方式提高并行性[1]。多线程体系结构的结构特点决定了它本身就能充分利用能量，另外，同步多线程(Simultaneous MultiThreaded)体系结构允许在设计时进行功耗和性能方面的权衡，这一点单线程的机器是做不到的。文[1]给出了若干基于多线程的功耗优化，这些优化可以减少多线程处理器的能量消耗或功率消耗，包括：减小执行带宽，动态功耗控制和取线程优化。

3.1 减小执行带宽

研究显示，使用同样的资源，SMT处理器所能处理的指令的吞吐量是单线程处理器的2倍。这就意味着对于同样的性能目标来说，比较小的执行带宽的多线程处理器和较大带宽的单线程处理器是等价的，因此，以上多线程处理器所需的功耗也较小。

3.2 通过反馈信息控制功耗

这个优化是把功耗看作是处理器的一个反馈输入，反馈信息是根据当前功耗的一个估计值，可用片上传感器得到或根据一个性能计数器的结果计算得到。然后，我们设定一个平均的功耗目标，把处理器的功耗限制在这个目标附近一定的范围内。控制功耗的策略有很多种，基本的做法是如果当前功耗超过了设定的范围，那么，取指部件或分支预测部件的行为将会有所改变以控制功耗。

反馈机制保证了真实的功耗不会超过一个阈值。这项技术能减少平均功耗，但更主要的是可以减少峰值功耗。

3.3 选择适当的线程

多线程处理提供了选择线程的能力，从而我们可以在选择线程时综合考虑性能因素和功耗因素。简单地说，这个优化就是在选择线程时，尽可能不选择那些存在很多投机的线程。

这里，选择线程的启发式信息不是优先选择低功耗的线程，因为这样做只能推迟高功耗线程的执行，而不能真正降低功耗。

4 功耗模型

4.1 低功耗编译研究的模拟平台

一般情况下，进行低功耗编译的研究需要以下三方面的支持：1)编译器。这个编译器应能够执行研究者进行研究的相关优化，而且应具有较好的组织结构，便于研究者对编译器进行适当的修改。2)模拟器。在低功耗编译的研究中，能够进行功能和性能模拟的编译器是必不可少的。常用的微体系结构级模拟器有SimpleScalar[4]，SimplePower[11]等。3)功耗模型。这部分专门用于功耗的模拟，要能和模拟器集成一起工作。现在常用的功耗模型有Cai/Lim[5]模型，Wattch[14]等。一般来说，这三部分是以图2的组织方式工作的。

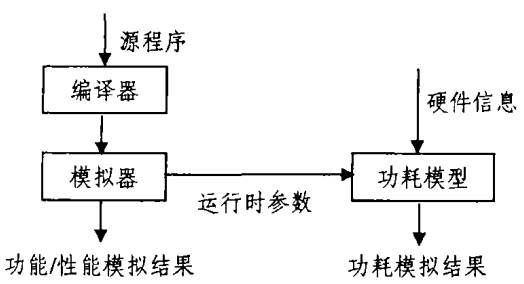


图2

4.2 功耗模型

上文提到，现在低功耗研究领域常用的功耗模型有：Cai/Lim[5]功耗模型和Wattch[14]。文[15]给出了这两种功耗模型的简单介绍并比较了它们的异同点。它们的相同点有：

1)它们的大体结构是类似的，并且都依赖基于行为的功

耗估计。所谓"基于行为的功耗估计"是给每个单独的设备结构的每个行为赋一个平均的功耗值。执行被测试的应用时，记录每次行为的出现次数，将它们和事先设定的相应行为的平均功耗值相乘，计算出最终的功耗。其中每个部件活跃时的平均功耗值一般是通过测试一些典型数据后计算得到的，而部件不活跃时的功耗通常被认为是活跃时的10%。这样的估计方法不是非常精确，但是消耗的时间比精确的功耗模拟要少很多。这种做法的致命弱点是，对于不能用单独的部件结构来表示的微体系结构模型，用这样的方法得到的结果就会毫无意义。

2)它们都是基于工具集 SimpleScalar 的，其中的模拟器是一个乱序执行的微处理器结构，并不特定模拟某个处理器，但它能快速地重新配置，模拟不同的处理器。

这两个功耗模型也都有各自的特点。Cai/Lim 功耗模型有如下特点：

1)它把 SimpleScalar 的体系结构分为17个硬件结构，进而又细分为32个块。

2)它提供了一种结构，基于这种结构，每种微处理器设计都可以用它自己的数据来使用这个功耗模型。

3)把访问某个结构的方式分为若干种类型，然后计算一个周期内每种类型访问的次数。这样的划分使所得的模拟结果更为精细。

4)所有部分的功耗值都是事先计算好的，并作为源代码的一部分，这样做的后果是：即使是对同一系列的处理器或体系结构，把它从一种结构扩展到另一种结构也相当困难。

5)它的精确度有望达到25%以内。

Wattch 的特点有：

1)Wattch 是一组功耗模型。其中一个是"所有部件都打开"的模型；另外三个都是对行为敏感的模型，它们之间的区别是条件时钟的允许程度不同。

2)Wattch 包含了若干基本组件，用这些组件来构造主要硬件结构的参数化模型。

3)使用了 H 树时钟分布，并且使用把时钟的功耗单独分离出来，这样，Wattch 所模拟的体系结构和 Alpha 较为类似。

4)利用电容来产生一个结构的每周期的开销，它可以根据 activity counter 所指定的使用情况进行缩放。

5)提供了一个易于扩展将来工作的框架结构，一个可比较不同的降低功率方案基础结构和一个可以精确量化功率的减少的参数化模型。

5)把基本的细节提供给了用户。

6)用和 Cai/lim 模型相似的方法跟踪硬件结构的使用情况，但是，它所支持的硬件结构的力度较 Cai/lim 模型要粗。

最后，文[15]中给出的结论并不乐观：两个功耗模型都无法为全范围的实验给出足够精确和完整的结果，而且二者的结构往往还不统一。可见，提供一个更精确有效功耗模型，是当前低功耗研究领域的当务之急。

结论 本文集中讨论了和低功耗编译相关的若干问题，从上面的讨论我们认识到：利用编译技术，或编译和操作系统、硬件相结合的技术降低功耗是从体系结构方面降低功耗的一种很重要的手段。但是，由于种种原因，这方面的研究工作做得并不充分。这也就给今后有志于低功耗编译研究的研究人员提供了机遇，提出了挑战。概括地讲，可以从如下方面开展研究工作：

1)全面、科学地评价各种编译优化手段对降低功耗的影响。从前面的分析，这样的评价结果对于构造功耗敏感的编译器有非常重要的指导意义。

2)某些编译优化技术对提高性能有很明显的效果，但是，会增加能量或功率的消耗。如果能改造这些优化，在其算法内加入功耗作为考虑因素之一，在基本不影响性能的情况下，降低功耗。

3)编译器利用自身优势为操作系统或硬件提供制导或其它有用的支持，可以最大程度地发掘某种降低功耗技术的潜力，是一种行之有效的方法，非常值得研究。

4)由于多线程体系结构在降低功耗方面的先天优越性，因此基于多线程体系结构的低功耗研究也是这方面研究的一个主要方面。

5)解决功耗模拟的精确性，有效性问题对于整个低功耗研究领域都会大有帮助。

参 考 文 献

1 Seng J S, Tullsen D M, Cai G Z N. Power-Sensitive Multithreaded Architecture. In: Proc. of 2000 Intl. Conf. on Computer Design

2 Cai G, Lim C H. Architectural level power/performance optimization and dynamic power estimation. In: Proc. of Cool Chips Tutorial at 32nd Intl. Symposium on Microarchitecture, Nov. 1999

3 Yang Hongbo, Gao G R, Marquez A, Cai G, Hu Z. Power and energy Impact by Loop Transformations

4 Austin T. The simplescalar tool set, version 2.0: [Technical Report 1342]. Computer Sciences Department, Univ of Wisconsin, 1997

5 Cai G, Lim C H. Architechture level power/performance optimization and dynamic power estimation. Cool Chips Tutorial, in conjunction with 32nd Annual International Symposium on Microarchitecture. Haifa, Israel, Nov. 1999

6 Yun H-S, Kim J. Power-Aware Modulo Scheduling for High-Performance VLIW Processors. In: the proc. of ISLPED'01, Huntington Beach, California, USA, 2001

7 Tang Z, et al. Ramp Up/Down Floating Point Unit to Reduce Inductive Noise. In: Proc. of Workshop on Power Aware Computer System, 2000

8 Rau B R. Iterative Modulo Scheduling: An Algorithm for Software Pipelining Loops. In: Proc. of the 27th Annual Intl. Symposium on Microarchitecture, 1994. 63～74

9 Kin J, Gupta M, Mangione-Smith W H. Filter Cache: An Energy Efficient Memory Structure. In MICRO-30, Dec. 1997

10 Yang J, Gupta R. Energy-Efficient Load and Store Reuse. In ISLPED'01, Huntington Beach, California, USA, 2001

11 Vijaykrishnan N, et al. Energy-Driven Integrated Hardware-Software Optimization Using SimplePower, in ISCA-27, May 2000. 95～106

12 Hsu C-H, Kremer U, Hsiao M. Compiler-Directed Dynamic Voltage/Frequency Scheduling for Energy Reduction in Microprecessors. In ISLPED'01, Huntington Beach, California, USA, 2001

13 Marculescu D. Profile-Driven Code Execution for Low Power Dissipation

14 Brooks D, Tiwari V, Martonosi M. Wattch: A Framework for Architectual-Level Power Analysis and Optimizations. In: the Proc. of the 27th Intl. Symposium on Computer Architecture, Vancouver, BC, 2000. 83～84

15 Ghiasi S, Grunwald D. A Comparison of Two Architectural Power Models