低功耗编译技术综述

胡定磊,陈书明
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 功耗问题已经成为制约电子系统发展的重要因素. 功耗是由硬件在运行软件时产生的,软件的数据存取和指令执行都会使硬件产生功耗. 编译器可以通过适当的调度优化,改变软件在硬件上的运行轨迹,使得硬件执行某一个程序时的功耗变小. 本文从如何对软件的功耗进行评估和如何实现低功耗的编译两大方面对低功耗编译的相关研究进行了广泛介绍,着重评述了专门的低功耗编译技术. 最后对当前低功耗编译存在的问题做了分析,给出了对于低功耗编译新方向的预测.

关键词: 低功耗; 编译技术; 功耗模型
中图分类号: TP3 **文献标识码**: A **文章编号**: 0372-2112 (2005) 04-0676-07

Low Power/ Energy Compilation Technology

HU Ding-lei, CHEN Shu-ming
(Computer School, National University of Defence Technology, Changsha, Hunan 410073, China)

Abstract: Power/ energy has been an important constraint for development of electronic system. Power/ energy is consumed by hardware when software runs on it, for accessing data and executing instructions of software can both make hardware consume power/ energy. Compiler can change the behavior of software running on the hardware by optimized scheduling, such that the power/ energy consumption of certain program decreases. This paper comprehensively reviews low power/ energy compilation technology related researches from two aspects, how to evaluate software power/ energy consumption and how to achieve low power/ energy by means of compilation technologies. In the end, we analysis the existing problems in current researches on low power/ energy compilation Technology, and predicate some new directions on the low power/ energy compilation technology.

Key words: low power/ energy; compilation technology; power/ energy model

1 引言

近年来随着移动处理、嵌入式应用的大量涌现,以及通用微处理器工艺水平和主频的不断提升,功耗日益成为信息系统设计者必须关心的问题,功耗问题业已成为处理器发展的一个重要瓶颈. 国际上,有关在硬件的门级、RTL 级、体系结构级上低功耗优化的研究也有较长时间了. 毫无疑问,功耗是由硬件系统产生的,但影响功耗的决不只是硬件本身. 硬件依赖于运行其上的软件来实现其处理信息的功能,功耗只是软件执行时的"副产品"而已. 软件本身不会产生功耗,但是软件的数据存取和指令执行都会使硬件产生功耗. 对于某个硬件来说,执行程序所产生的功耗取决于它的机器代码(假设不考虑操作系统的影响),而机器代码是从源代码编译而来的,可以说编译过程也影响了硬件的功耗. 既然编译器可以很大程度上控制硬件的运行轨迹,除了传统的优化目标——性能之外,编译器也可以通过适当的调度优化,使得硬件执行某一个程序时的功耗变小.

国际上对于低功耗编译研究的历史并不长,是从九十年代初才开始研究的. 最早这方面的文章出现于[1,2],Tiwari 等人在这些文章中提出了对软件进行功耗分析的一些基本概念,建立了基本的指令级功耗模型,以 486DX 为例初步探讨了低功耗编译技术. 到目前为止,有关低功耗的研究,无论是从硬件还是从软件来说,都是非常热门的话题,有许多问题有待解决.

本文对近年来低功耗编译方面的研究做了一个总的概括. 文章的第二部分引出了了低功耗编译需要解决的两个主要问题;第三、第四部分论述了对低功耗编译过程的两个主要阶段:软件功耗分析和低功耗编译的相关研究;最后对当前低功耗编译存在的问题做了分析,给出了对于低功耗编译新方向的预测.

2 低功耗编译需要解决的主要问题

编译器都有调度优化的功能,所谓调度优化无非是指在保持程序语义和遵守目标机器资源限制的情况下,使得目标

收稿日期:2004-06-01;修回日期:2004-10-09
基金项目:国家 863 项目(No. 2002AA1Z1110);国家自然科学基金(No. 60473079)

函数代价达到极值.在对低功耗编译进行说明之前,我们先回顾一下传统的编译器.它的调度优化目标是性能,即使得程序的执行时间最短.而对于低功耗调度而言,它的调度优化是多目标的,通常是性能和功耗一起考虑,即在满足一定的性能约束的情况下,使得程序运行的功耗最小.值得注意的是,大多数情况下,性能和功耗并不是矛盾的,减少程序执行时间,同样会起到减少程序功耗的目的.Chung 等用图 1 表示了在一个简单的硬件体系结构下,编译器对源代码的变换对于功耗和性能的影响[3].从中我们可以看出有时候对于功耗的优化会导致性能的的降低(能量方程、执行时间方程与 y 轴的三角区域),有时候对于性能的优化会导致功耗的增加(能量方程、执行时间方程与 x 轴的三角区域).

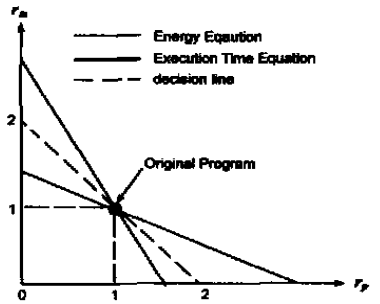

图 1 能量方程与性能方程[3]

低功耗编译为了通过合理的调度优化,使得程序的功耗变小,需要解决两个问题:如何对软件功耗进行分析,及如何实现低功耗的调度.通过对软件的功耗特征进行分析,用以指导编译进行相关的调度优化,并对调度优化的结果进行评估.下面我们将就这两方面进行详细描述.

3　软件的功耗分析

对程序进行功耗分析一般过程是这样的,先建立软件的功耗模型,然后通过模拟或实验的方法得出模型参数,最后用功耗模型对软件的功耗进行评估.

通常描述硬件功耗的模型有三类:RTL 级(或门级)功耗模型、微体系结构级功耗模型、指令级功耗模型[4].但是前两种模型对于编译来说都不合适:使用目标机器的 RTL 级描述进行功耗分析,速度非常慢,且需要专门昂贵的功耗分析工具,而且有时候目标机器 RTL 描述未必能够得到;微体系结构级的功耗模型,就目前的水平而言,还是比较粗糙,误差也比较大[5],也不适合于编译使用.Julien 等提出了一种新奇的功耗模型 ——功能级功耗分析 FLPA,它将显著影响功耗的体系结构参数和程序代码的算法参数作为功耗模型参数,通过使用简单汇编代码在目标硬件上执行时测量硬件的电气值,就可以拟合出功耗方程[6].使用这种方法甚至可以估计源代码的功耗[7].应当说这种方法可以快速的对软件功耗进行分析,这是它最大的优点;同时由于其隐藏了太多的细节,可能不利于指导低功耗编译,对这种方法还需要进一步的评价.

目前而言,对软件进行功耗分析的主流做法是建立指令级功耗模型、通过实验方法得出功耗模型的参数、通过模拟对软件的功耗做出评估.

3.1　指令级功耗模型

软件的组成单位是指令,指令级功耗模型赋予每条指令一定的功耗值,配合模拟器的模拟运行,可以给出软件的精确功耗特性,指导编译调度.基本的指令级功耗模型是由 Tiwari 提出的,他认为一个程序执行所耗用的能量,分为三部分:每条指令的能量与该指令执行次数的积、指令间效应(Inter-instruction Effects)造成的附加能量与执行次数的积、由于资源限制引起的流水线停顿所造成的附加能量[1].

$$E = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k$$

这里,指令的基本能耗(B_i)是指一条指令从取指到写回结果,在流水线中执行所耗费的能量.功能不同的指令,其基本能耗往往差别较大.对于同一部件而言,指令序列$\{i, j\}$和$\{j, i\}$流过时,所产生的能耗是不一样的,因为不同指令间切换所引起的翻转率是不同的.通常情况下,指令序列$\{i, j\}$所产生的能耗大于 i 和 j 两条指令基本能耗的和.这种由于不同指令间切换所引起的附加能耗,就称为指令间效应($O_{i,j}$).而由于资源限制及 Cache 失效所造成的流水线停顿,会延长程序的执行时间,使程序执行的总能耗增加,E_k 即描述这种附加的能耗.

现在许多研究者所提出的指令级功耗模型,都是适应新体系架构的出现,对基本指令级功耗模型的改造和补充,主要体现在以下几个方面.

A. 使指令级功耗模型可以正确反映程序代码的流水线处理.现代的处理器一般都是流水线结构,在同一拍中可能有多条指令执行.为了获得功耗的实时信息,必须考虑指令在流水线各个阶段的功耗.下图显示了考虑流水线各阶段功耗与传统方法的不同.

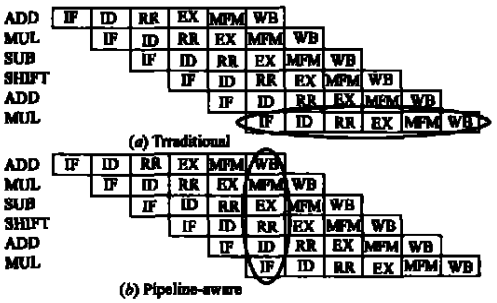

图 2 传统功耗模型与考虑流水线的功耗模型[4]

B. 使指令级功耗模型可以正确反映程序代码在超长指令字 VLIW 体系结构中的处理.VLIW 体系结构中一般有多个数据通路,这些通路可能是同构,也可能是异构的.Sami 等给出了考虑流水线和 VLIW 体系结构这两种情况的下的能耗模型公式如下,这里假设硬件有 K 个数据通路、指令序列 $\overline{W} = \langle W_1, W_2, \cdots, W_{n-1}, W_n, \cdots, W_N \rangle$[4]:

$$E(\overline{W}) = \{ \sum_{\forall n \in N} [\sum_{\forall s \in S} (U_s(0|0) + \sum_{\forall k \in K} v_s(w_n^k | w_{n-1}^k) + \sigma_s^n + \mu_s^n) + I(W_n | W_{n-1})] \} + E_c + c_0$$

C. 明确寄存器文件的功耗模型.在基本指令级功耗模型中,寄存器文件的功耗隐藏在了指令的基本功耗和指令间效应的背后.Benini 等提出了多端口寄存器文件的功耗模型,认

为寄存器文件的功耗与同时对多个端口的读写数目成正比，而读写寄存器的能耗则与上一次读写寄存器的哈明距有(Hamming distance)关[8]：

$$P_{RF} = P_i + \frac{1}{T}\sum_{1\le n\le N}(E_{r,n} + E_{w,n})$$

$$E_{r,n} = \sum_{1\le i\le N_{rp}} H(RR_{i,n}, RR_{i,n-1}) * E_{rb}$$

$$E_{w,n} = \sum_{1\le i\le N_{wp}} H(RW_{i,n}, old_{i,n}) * E_{wb}$$

D. 指令级功耗模型中的 Cache 功耗模型. 因为程序运行过程中对 Cache 的访问难以精确的估计，所以多数指令级功耗模型使用平均的 Cache 功耗作为由于 Cache Miss 引起的流水线停顿造成的附加功耗. Sami 等在其指令级功耗模型中对于 Cache 的功耗估计即采用的平均功耗[4]：

$$O_s^n = m_s^n * p_s^n * S_s, \mu_s^n = l_s^n * q_s^n * M_s$$

为了精确估计 Cache 功耗，Benini 提出了一个分级式的 Cache 功耗模型：对于多体 SRAM 构成的 Cache，首先建立不同的 SRAM 体的功耗模型，然后将这些功耗模型依照 Cache 的组织结构进行组合，以产生正确的读写 Cache 的功耗模型. 其中 SRAM 的功耗模型主要基于 SRAM 的操作模式(读、写、空闲等)，而功耗则是模式转换的函数. 在 Cache 这一层次，需要将对它的访问转化为对 SRAM 体的访问，以期产生精确的功耗[8].

3.2 功耗模型参数的获取及验证

为了得到功耗模型的各项参数，一般要经过以下过程：

A. 提取模型参数. 应当说指令级功耗模型的模型参数是比较多的，特别是对于"VLIW + 深度流水 + 多级 Cache"的体系结构而言. Julien 等就认为要为 TMS320C6201 *DSP* 建立指令级功耗模型需要 71^8 次实验[6]，因此非常有必要采取措施减少模型参数. 这主要是通过对功耗模型进行合理假设得到的. 如 Sami 等通过假设 VLIW 体系结构中的数据通路在功耗上是不相关的，将其指令级功耗模型的特征量从 $O(N^{2k})$ 减少到 $O(k * N^2)$[4]. 通过指令分组，将功耗相近的指令划分为一类，可以进一步达到减少模型特征量的目的[9].

B. 实验. 实验方法分两种情况：一是在有目标芯片的 RTL 级描述的情况下，通过硬件功耗分析工具分析 RTL 级处理器执行指令时的功耗，得出功耗数据，这是指令级功耗模型的常用方法[4,8]；二是只有实际的目标芯片，没有目标芯片的 RTL 级描述的情况下，通过实际测量处理器执行指令时的电气特征值(电压、电流等)，获得其功耗实验结果，这种方法适用于比较粗糙的指令级功耗模型[1]或更高层的功耗模型[6].

C. 计算功耗模型的参数并进行验证功耗模型. 若功耗模型与实际情况误差较大，则需要重新计算模型参数甚至修改功耗模型.

3.3 评估软件功耗

建立功耗模型的目的是对软件的功耗进行评估. 对于指令级功耗模型来说，通常还要建立指令级模拟器，将功耗模型融入其中，通过在模拟器上运行程序代码，给出其功耗值[4,8]. 这一点对于嵌入式应用尤其重要. 比如为 DSP 开发代码，一般采用交叉编译的形式，有一个目标 DSP 的指令级模拟器，对于验证代码正确性、方便程序开发，进行低功耗编译都非常有用.

为了对编译提供有用的功耗信息，指令级模拟器最好具有这些功能：显示程序执行时实时的功耗信息；可以给出统计信息，如 Profile 信息等；可以给出软件功耗在硬件体系结构上的分布. 而对于功能级功耗模型 FLPA 而言，可以直接从源代码级对软件的功耗进行评估[7]，更适合于对算法级的低功耗优化进行评估，而非编译级.

4 低功耗编译的实现

编译器一般由三部分组成：前端、机器无关优化和后端，如图 3 所示. 前端负责进行语法语义分析并生成中间代码，机器无关优化就是对中间代码的优化，后端与硬件的体系结构密切相关，因此又经常将后端称为代码生成. 通常低功耗编译的相关优化调度是在代码生成阶段进行的，毕竟后端与硬件的体系结构直接相关. 但是在编译器中进行与机器无关优化达到降低功耗的目的还是可能的，例如在源代码级就可以通过 loop-unrolling、loop-blocking 变换来降低软件的功耗[3]. 在第二节中，我们已经讲过大多数可以提高性能的编译技术同样会起到降低功耗的目的，如一些传统的编译技术——软件流水，函数内联(function inline)等，对于这类技术，我们在此不做论述. 我们将关注那些专门为实现低功耗提出的编译技术，这些技术一般是在代码生成阶段实现的.

图 3 编译器的基本任务[10]

硬件的功耗主要是由动态功耗和静态功耗组成，动态功耗是硬件的负载电容的充放电(门电路开关)所造成的功耗，而静态功耗则是由漏电流(Leakage Current)造成的功耗. 可以用下式来表示：

$$P_{total} = P_{dynamic} + P_{static} = CNV^2 f + VI_{leak}$$

这里，C 代表负载电容、N 代表电路每拍的信号翻转次数、V 代表电压、f 代表时钟频率、I_{leak} 代表漏电流. 传统上，动态功耗占据了总功耗的的大部分；在当前的深亚微米工艺条件下，两者已大致相当；但随着 CMOS 工艺尺寸的不断缩小，静态功耗会逐渐超过动态功耗成为主导[11]. 从减少总功耗来说，无非是从减少动态和静态功耗入手. 硬件设计上经常采用深度流水、门限时钟等技术达到降低功耗的目的，但从编译角度来说，通过调度优化来实现低功耗主要靠两方面：一是减少程序执行时引起的电路的翻转次数(因子 N)或降低电压(因子 V)来降低动态功耗；二是分析并调整程序在硬件上的运行情况，将暂时不用的部件关闭电源，从而达到减少静态功耗的目的(因子 I_{leak}). 这其中，减少因子 N 可以单纯依靠编译调度来实现，不需要硬件支持，而其它调度措施(降低 V、切断 I_{leak})则需要硬件有支持低功耗的设置或指令，软件可对其进行操作. 下面将对动态功耗和静态功耗的编译优化技术分别

进行阐述.

4.1 动态功耗的编译优化

减少程序执行时电路的信号翻转次数,主要通过两个途径:使用低功耗指令和进行低功耗调度.而电压的调整也有静态和动态电压调整两种策略.

4.1.1 使用低功耗指令 不同的指令,其功耗特性是不同的.编译时可以通过采用低功耗的指令替代高功耗的指令作为程序的实现,达到低功耗的目的.指令的选取标准是指令的基本功耗,这种思想是由 Tiwari 提出的,他认为含有存储器操作数的指令的功耗比只有寄存器操作数的指令大的多,尽量使用只含寄存器操作数的指令可以降低程序功耗[2].

随着多媒体应用的迅猛发展,许多硬件现在都支持 SIMD 指令或短向量指令,使用这类指令可以大大的提升性能和降低功耗[12,13].应当说明支持 SIMD 指令的编译与一般的向量化编译并不一样,通常这些支持 SIMD 指令的硬件并没有独立的向量部件,SIMD 指令也是由标量部件完成的.SIMD 指令可以通过库函数的形式提供给编译器使用,但这影响了代码的可移植性,加重了开发人员的负担.如何从代码中将多个短数据操作合并为单个长数据操作是进行 SIMD 编译研究的主要内容.

在文[14]中,Sreraman 等针对 Intel 的多媒体扩展(MMX)指令,以 SUIF 编译器为基础,实现了一个向量化编译器.它主要是通过对代码中的标量数据和数组依赖性进行分析来识别数据的并行性.为了提高向量化程度,还采用许多循环变换技术,如 strip mining, scalar expansion, grouping and reduction, loop fission and distribution.

Lorenz 等提出的使用 SIMD 指令进行低功耗编译的方法[15],其实现策略与上述[14]基本相同,但针对 M3-DSP 特殊的存储结构和寻址方式,对于如何优化数组和标量数据的存储地址分配进行了研究[16].

Leupers 在经典的树模式匹配和动态规划代码生成算法的基础上[17],提出了在代码选择(code selection)阶段进行 SIMD 指令转换的方法[18].为了从中间代码中识别尽可能多的 SIMD 指令,Leupers 采用的是图模式匹配,而且在模式匹配时可以有多条规则作为备选.图模式匹配完毕后,使用整数线性规划对于代码选择的约束条件进行建模,规划的目标函数使得 SIMD 指令的转换最大化.

4.1.2 低功耗指令调度 调度就是在不改变程序语义的前提下,重新安排指令执行的顺序,这样通过调度可以改变程序运行的功耗行为.为了减少程序执行的功耗,低功耗调度通常是通过适当安排指令执行的顺序减少指令间效应造成的功耗、减少功能部件的电路信号翻转,对应在硬件上就是减少程序在执行部件、通讯部件和存储部件因电路信号翻转造成的功耗.

在 VLIW 体系结构中虽然有多个执行部件,但大多数情况下每一拍流出的指令数是不同的,因此每一拍的功率也是不一样的.相邻拍之间的功率差——级差功率(step power)和峰值功率(peak power)对于处理器的可靠性会有很大的影响.Yun 等提出了功率敏感模调度算法,在不牺牲调度性能的情况下,调整模调度算法通常使用的 ASAP(As early as possible)策略,使得调度后每一拍流出的指令数尽可能一致,从而达到大幅降低级差功率和峰值功率的目的[19].

Bona 等对于所有功能单元相同的 VLIW 体系结构,提出了一种简单的 spatial scheduling 算法[9].在基本块内依次调整每条长指令中每个操作所在的执行部件,使得下式最小,从而达到降低功耗的目的.

$$\sum_k v(w_n^k \mid w_{n-1}^k),\ \forall (w_n^k \mid w_{n-1}^k) \in \text{Basic Block}$$

鉴于编译器代码生成阶段的指令选择、指令调度和寄存器分配中有许多 NP-hard 问题,而遗传算法则是解决此类问题的一种有效手段.Lorenz 等设计了多阶段耦合的遗传代码生成器,并建立了指令级功耗模型用于评估交换和变异的适应性[20].

如果硬件支持 Ramp-up 指令,则可以通过编译在程序中插入 Ramp-up 指令,达到电流随时钟斜线上升(Clock Ramping)的效果,解决门控时钟所引起的冲击电流(Surge Current)问题[21].

现在一般硬件的存储部件主要有寄存器、Cache 和存储器三种形式.实际上,在硬件执行程序的过程中,存储系统占整个功耗的很大一部分,有时甚至比计算部件的功耗还要大,如图 4 所示.

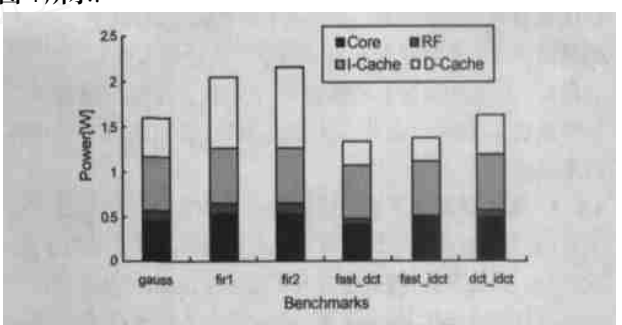

图 4 某 VLIW DSP 各部件的功耗[3]

虽然对于 cache 和存储器编译优化的研究由来已久,但对于专门实现 Cache 和存储器等存储部件低功耗的编译技术的研究,近三四年才刚刚开始.传统编译器对于存储系统进行的优化调度,中心思想是提高代码的局部性,减少程序运行时间,这些优化通常也可以起到降低功耗的作用.Kandemir 等对于当前流行的循环优化技术,如循环展开、循环合并、循环交换、标量扩展等,对于功耗的影响做出了评估[22].他认为这些优化技术令计算部件的负载增大,减少了对 Cache 和存储器的访问,从而使得 Cache 和存储器的功耗减小,计算部件的功耗增大,但总的功耗降低了.

减少寄存器文件的功耗,是通过改进寄存器分配算法实现的.传统上寄存器分配的主要目的是最大限度的利用寄存器,减少溢出代码.Mehta 等提出了低功耗的寄存器分配算法,这个算法建立在通过调整相邻寄存器访问,使得前后两次寄存器访问的哈明距变小,以减少寄存器译码部分和指令总线的信号翻转,从而降低功耗[23].与传统的着色图寄存器分配算法不同,Gebotys 等将寄存器的分配映射为网络流问题,以变量的生命周期为顶点、以寄存器中变量的转换为边、并以变

量转换所带来的功耗为边的权,使用最小费用网络流算法实现低功耗的寄存器分配[24]. Zhang 等在文[24]的基础上,对程序中存在分支、合并、循环等控制机构时,如何使用最小费用网络流算法实现低功耗的全局寄存器分配进行了研究[25].

Tomiyama 等提出了用于减少片上 Cache 和片外主存总线信号翻转的调度算法[26],其基本思想是重排序指令,使得整个程序指令的哈明距变小,于是当发生指令 Cache 失效时,从片外主存读入指令数据的时候,数据总线的翻转率降低.

Kandemir 等通过分析程序的数据访问模式,使用可以绕过 Cache 标签(tag)直接访问 Cache 数据的访问模式,降低了访问 Cache 的功耗[27]. 他们还研究了循环平铺(loop tiling)技术对于功耗的影响,循环平铺虽然会引起处理器数据通路和 Cache 的功耗增加,但可以使存储器的功耗大幅度降低[28].

Grun 等认为在存储器级,可通过编译有效利用存储器的访问模式(e. g. ,page- ,burst-mode),达到隐藏延迟,减少功耗的目的[29].

在嵌入式系统中,除了常见的存储器、cache 等,近年来开始出现了新的存储形式——高速暂存区(SPM, Scratch Pad Memory)[30]. 与 Cache 不同,SPM 是由编译器完全控制的,它的出现,除了可以提高性能外,还克服了 Cache 的存在造成的程序的不可预测性,这一点对实时系统尤其重要. Steinke 等对如何优化 SPM 的分配算法,降低程序功耗做了研究[31]. 基本思想是以将指令和数据放入 SPM 所节省的功耗为收益,把可以获得最大收益的指令和数据放入 SPM. 该分配算法是用整数线性规划实现的,以总收益为目标函数,以 SPM 的容量作为约束条件.

4.1.3 编译器指导下的电压调整 动态功耗与电压是幂次关系(V^2),因此降低电压是减少动态功耗的非常有效的手段. 现在有一些处理器,特别是嵌入式处理器,可以对某些部件的电压进行调整设置,如 Intel 的 Xscale 嵌入式处理器. 编译器通过分析程序各部分的时间特性,为它们设置不同的运行电压,达到降低动态功耗的目的. 电压调整策略也可以分为静态和动态两种. 静态电压调整是指编译器在满足时序要求的前提下,为整个程序设定一个唯一的电压;而动态电压调整则为程序的各个部分设定不同的电压. 相比较而言,动态电压调整策略应用更加广泛一些.

Saputra 等提出了一个在不影响程序性能的前提下,使用静态电压调整降低程序功耗的方法[32]. 为此他们使用了面向性能的循环优化,但这是为通过电压调整降低功耗创造机会,而不是提高程序性能. 循环优化之后,程序的执行时间缩短,于是可以将 CPU 的电压降低到一个值(频率随之降低),使得程序的执行时间与原来一致,而功耗比原来为低.

在文[32]中,Saputra 等也提出一个使用整数线性规划求解动态电压调整策略来实现低功耗的算法. 首先将程序划分为若干个子任务,并计算或测量出每个子任务在不同电压下的功耗. 用 E_{ij}、X_{ij}分别表示子任务 i 在第 j 个电压下的功耗、执行时间,S_{ij}表示子任务 i 是否运行在第 j 个电压下,则可建立整数线性规划模型如下,求得最优的动态电压调整策略:

$$E_{energy} = \min\left(\sum_{i=1}^{M}\sum_{j=1}^{N} S_{i,j}E_{i,j}\right)$$

$$\begin{cases} \sum_{j=1}^{N} S_{i,j} = 1 \quad \forall 1 \le i \le M \\ \sum_{i=1}^{M}\sum_{j=1}^{N} S_{i,j}X_{i,j} = X_{original} \end{cases}$$

赵荣彩等提出对于具有执行频率可动态调整的多处理器类多线程体系结构,可以在编译过程中基于对应用程序行为的分析,结合线程划分识别出可降频执行的线程并计算出线程的降频因子,根据每个线程的时序要求,动态调整执行部件的频率,以达到降低功耗的目的[33].

4.2 静态功耗的编译优化

与动态功耗不同,静态功耗是由漏电流引起的,与电路的信号翻转并无关系,降低动态功耗的方法并不适用于这里. 目前普遍的做法是将硬件中处于非活跃状态的部件关闭,从而减少了静态功耗. 但这部分硬件的关闭与恢复必须有相应的硬件技术支持才可以.

Zhang 等提出了用于减少数据 Cache 的漏电功耗(Leakage Power)的编译技术[34]. 因为数据具有局部性,程序运行时只用到 Cache 的一小部分,因此编译时在分析代码的基础上可以插入指令关闭当前不会用到的 Cache lines,从而降低了 Cache 的功耗. 为此针对数组集中和指针集中的两种应用,他们研究提出了代码转换技术,可以更好地支持对 Cache 的功耗优化.

Hu 等提出了用于减少指令 Cache 的漏电功耗的方法[35]. 该方法基于所谓的休眠(drowsy cache)技术,每个 cache line 有一个状态位用于控制它是否处于休眠状态,还有一个屏蔽位用于阻止状态转换. 系统周期性的向所有的 cache line 发送休眠信号. 通过分析分支目标缓冲(BTB)可以确定程序的"热点"(hotspot),对于属于"热点"范围的指令 cache line 设置屏蔽位,使之不受系统休眠信号的影响. 通过预测转移可以隐藏 cache line 状态转换引起的延迟. 这样一来,指令 cache 大多数时间将处于休眠状态,有效的减少了静态功耗.

5 当前存在问题及发展方向

低功耗编译作为一个新兴的研究领域,对于编译器开发人员提出了更高的要求. 主要表现在三个方面:第一,传统的编译器开发人员通常只要了解目标硬件的指令集和体系结构就足矣. 然而要开发低功耗编译,开发人员不但要了解硬件的指令集和体系结构,还要了解硬件在 RTL 级、逻辑级甚至门级的构造;不仅要有编译方面的知识,还要有较深的硬件知识. 第二,进行软件功耗分析的工作量非常巨大. 从功耗模型的建立,模型参数的获取及验证,到构造功耗模拟器,需要有团队的支持才可以完成. 第三,低功耗编译方面的基础设施缺乏. 对传统编译技术的研究,目前有许多工具、平台可资利用,而用于低功耗编译方面的基础设施还比较少. 以上三点的存在,一定程度上提高了进行低功耗编译研究的门槛,使得研究效率难以提高.

就低功耗编译技术本身而言,尚存在以下方面的缺憾:一是普遍缺乏对提出的低功耗编译技术的系统评价. 通常,验证

了编译技术对降低某部件功耗的有效性,但没有考虑对对其它部件乃至整个系统的影响.二是有些采用的功耗模型为理想的理论模型或实际硬件的简化抽象,与实际情况有所偏差,一定程度上影响了实验验证的可信度.三是现有的低功耗编译技术大多就某种程序结构进行优化,对一个应用程序而言,如何将其划分为各种结构,分别使用合适的优化技术,以及各种优化之间的相互作用和对整体的影响,都还没有深入探讨.

随着研究的发展,我们认为低功耗编译的研究今后将呈现出如下趋势:

A. 对软件功耗模型的改进与完善.功耗模型有不同的粒度,指令级功耗模型较精确,但又太复杂,体系结构级和功能级功耗模型虽快捷但又过简略.为适应不同的低功耗编译技术构造简洁高效的功耗模型,始终是验证和指导低功耗编译发展所必须的.

B. 针对不同的体系结构的低功耗编译实现的研究.即针对单线程与多线程、嵌入式与非嵌入式、实时与非实时、恒有电源与电池电源等不同的体系结构的特点,来实现低功耗编译.比如对使用电池的硬件(主要用在手持设备中),其编译优化的目标是使程序运行功耗有尽可能少的波动,从而使得电池寿命最大化,而不是单纯地以降低功耗为目的.

C. 各种传统编译优化技术在低功耗方面的改良,以形成各种 Energy/ Power-aware 编译技术.这方面将主要集中在各种循环优化、存储优化等技术上.

D. 硬件支持下的低功耗编译.随着可变电压、可变频率、可动态装配硬件的出现,为降低功耗提供了非常有效的手段,编译器如何利用硬件的这些功耗特性实现低功耗,将一直是非常值得研究的课题.

E. 低功耗编译与多线程、操作系统等的交互.低功耗编译如何为多线程、操作系统的调度提供有关提示信息,又如何利用多线程、操作系统等提供的有关硬件功耗信息指导编译.

6　结束语

本文从如何评估软件功耗和如何用编译技术实现低功耗两方面,尽可能广泛的阐述了与低功耗编译相关的研究,但还没有达到完备的程度.目前对于软件低功耗编译技术的研究,国内也是刚刚起步,比如我们正在就超长指令字 DSP 的软件低功耗编译技术进行研究.作为一个新兴而活跃的研究领域,软件低功耗编译技术有许多问题还亟待解决,新的研究成果还在不断涌现,我们拭目以待.

参考文献:

[1] V Tiwari, S Malik, and A Wolfe. Power analysis of embedded software: a first step towards software power minimization[J]. IEEE Transactions on VLSI Systems, 1994, 2(4): 437 - 445.

[2] V Tiwari, S Malik, A Wolfe. Compilation techniques for low energy: an overview[A]. Proceedings of the IEEE Symposium on Low Power Electronics[C]. San Diego, US: 1994. 38 - 39.

[3] E Chung, L Benini, G Micheli. Source code transformation based on software cost analysis[A]. ISSS '01[C]. Montréal, Québec, Canada, 2001. 153 - 158.

[4] M Sami, et al. An instruction-level energy model for embedded VLIW architectures[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2002, 21(9): 998 - 1010.

[5] S Ghiasi, D Grunwald. A comparison of two architectural power models[A]. Proceedings of the First International Workshop on Power-Aware Computer Systems[C]. 2000. 137 - 152.

[6] N Julien, et al. Power consumption modeling and characterization of the TI C6201[J]. IEEE Micro, 2003, 23(5): 40 - 49.

[7] E Senn, et al. Power consumption estimation of a C program for data-intensive applications[A]. Proceedings of the 12th IEEE International Workshop on Power And Timing Modeling, Optimization and Simulation PATMOS '02[C]. Sevilla, Spain: 2002. 332 - 341.

[8] L Benini, et al. A power modeling and estimation framework for vliw-based embedded systems[A]. IEEE 11th International Workshop on Power and Timing Modeling, Optimization and Simulation[C]. Yverdon-les-Bains, Switzerland: 2001.

[9] A Bona, et al. Energy estimation and optimization of embedded VLIW processors based on instruction clustering[A]. Proceedings of the 39th conference on Design automation[C]. New Orleans, LA: 2002. 886 - 891.

[10] L Benini, G Micheli. System-level power optimization: techniques and tools[J]. ACM Transactions on Design Automation of Electric Systems, 2000, 5(2): 115 - 192.

[11] N Kim, et al. Leakage current: Moore's law meets static power[J]. IEEE Computer Special Issue on Power-and Temperature-Aware Computing, 2003, 36(12): 68 - 75.

[12] M Lorenz, et al. Compiler based Exploration of DSP Energy Savings by SIMD Operations[A]. Asia South Pacific Design Automation Conference[C]. Yokohama, Japan: 2004. 838 - 841.

[13] C Kozyrakis, D Patterson. Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks[A]. Proceedings of the 35th International Symposium on Microarchitecture[C]. Instabul, Turkey: 2002.

[14] N Sreraman, R Govindarajan. A vectorizing compiler for multimedia extensions[J]. International Journal of Parallel Programming, 2000, 28(4): 363 - 400.

[15] M Lorenz, L Wehmeyer, T Dräger. Energy aware compilation for DSPs with SIMD instructions[J]. ACM SIGPLAN Notices, 2002, 37(7): 94 - 101.

[16] M Lorenz, et al. Optimized address assignment for DSPs with SIMD memory accesses[A]. Asia South Pacific Design Automation Conference[C]. Yokohama, Japan: 2001. 415 - 420.

[17] A V Aho, et al. Code generation using tree matching and dynamic programming[J]. ACM Trans. on Programming Languages and Systems, 1989, 11(4): 491 - 516.

[18] R Leupers. Code selection for media processors with SIMD instructions[A]. Proceedings of the conference on Design Automation and Test in Europe[C]. Paris, France: 2000. 4 - 8.

[19] H Yun, J Kim. Power aware modulo scheduling for high performance VLIW processors[A]. Proceedings of the 2001 International Symposium on Low Power Electronics and Design[C]. Huntington Beach, US: 2001. 40 - 45.

[20] M Lorenz ,et al. Low-energy DSP code generation using a genetic algorithm[A]. Proceedings of the IEEE International Conference on Computer Design 2001[C]. Austin ,US:2001. 431 - 437.

[21] W Liao ,L He. Power modeling and reduction for VLIW processors. L Benini. Compilers and Operating Systems for Low Power[C]. Kluwer Academic Publishers ,2003. 155 - 171.

[22] M Kandemir ,et al. Influence of compiler optimizations on system power [A]. Proceedings of the 37th Design Automation Conference[C]. Los Angeles ,USA:2000. 304 - 307.

[23] H Mehta ,et al. Techniques for low energy software[A]. Proceedings of the 1997 International Symposium on Low Power Electronics and Design [C]. Monterey ,US:1997. 72 - 75.

[24] C H Gebotys ,Low energy memory and register allocation using network flow[A]. Proceedings of the 34th Annual Conference on Design Automation[C]. Anaheim ,US:1997. 435 - 440.

[25] Y Zhang ,X Hu ,D Z Chen. Efficient global register allocation for minimizing energy consumption[J]. SIGPLAN Notices ,2002 ,37(4) :42 - 53.

[26] H Tomiyama ,et al. Instruction scheduling for power reduction in processor-based system design[A]. Proceedings of the Conference on Design ,Automation and Test in Europe 98[C]. Paris ,France ,1998. 855 - 860.

[27] M Kandemir ,et al. Toward an energy-aware iteration space tiling[A]. Workshop on Language ,Compilers ,Tools of Embedded System[C]. Vancouver ,BC:2000.

[28] M Kandemir ,I Kolcu. Reducing cache access energy in array-intensive applications[A]. Proceedings of the Design ,Automation and Test in Europe 2002[C]. Paris ,France:2002. 1530 - 1591.

[29] P Grun ,N Dutt ,A Nicolau. Memory aware compilation through accurate timing extraction[A]. Proceedings of the 37th Conference on Design Automation[C]. Los Angeles ,US:2000. 316 - 321.

[30] P R Panda ,N Dutt ,and A Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications[A]. Proceedings of the Conference on Design ,Automation and Test in Europe[C]. Paris ,France:1997. 7.

[31] S Steinke ,et al. Assigning program and data objects to scratchpad for energy reduction[A]. Proceedings of the Conference on Design ,Automation and Test in Europe[C]. Paris ,France:2002. 409.

[32] H Saputra ,et al. Energy-conscious compilation based on voltage scaling [A]. ACM SIGPLAN Joint Conference on Languages ,Compilers ,and Tools for Embedded Systems[C]. Berlin ,Germany:2002. 2 - 11.

[33] 赵荣彩 ,等. 低功耗多线程编译优化技术[J]. 软件学报 ,2002 ,13(6) :1123 - 1129.

[34] W Zhang ,et al. A compiler approach for reducing data cache energy [A]. 17th Annual ACM International Conference on Supercomputting [C]. San Francisco ,US:2003. 76 - 85.

[35] J Hu ,et al. Exploiting program hotspots and code sequentiality for instruction cache leakage management[A]. Proceedings of the 2003 International Symposium on Low Power Electronics and Design[C]. Seoul ,Korea:2003. 402 - 407.

作者简介:

胡定磊 男,1976 年生于山东章丘,1998 年毕业于国防科技大学系统工程与数学系,获学士学位;2001 年毕业于国防科技大学人文与管理学院,获硕士学位;现为国防科技大学计算机学院博士研究生. 主要研究方向为超长指令字编译、低功耗编译等. Email :dl_hu@163.com.

陈书明 男,1961 年生于安徽六安,教授,博士生导师. 1993 年毕业于国防科技大学计算机系,获博士学位. 主要研究方向为计算机体系结构,通用微处理器、DSP 及超大规模集成电路设计、低功耗电路设计等. E-mail :smchen@nudt.edu.cn.