基于语法树的实时动态电压调节低功耗算法*

易会战+, 陈 娟, 杨学军, 刘 喆

(国防科学技术大学 计算机学院,湖南 长沙 410073)

A Real-Time Dynamic Voltage Scaling Algorithm Based on Syntax Tree for Low Power

YI Hui-Zhan+, CHEN Juan, YANG Xue-Jun, LIU Zhe

(School of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4575807, E-mail: huizhanyi@nudt.edu.cn, http://www.nudt.edu.cn

Received 2004-10-31; Accepted 2005-06-02

Yi HZ, Chen J, Yang XJ, Liu Z. A real-time dynamic voltage scaling algorithm based on syntax tree for low power. *Journal of Software*, 2005,16(10):1726–1734. DOI: 10.1360/jos161726

Abstract: Dynamic voltage scaling is an effective low-power technique. Using the technique, compiler-directed dynamic voltage scaling can reduce computer's energy consumption effectively. Based on programming language's syntax tree, a real-time dynamic voltage scaling algorithm for low power is presented, and the algorithm assisted with static timing analysis could make intra-task dynamic voltage scaling by automatically inserting dynamic voltage scaling source code. The algorithm has been realized in the real-time low-power system RTLPower, and obtained energy reduction of up to 50 percent over no power management in some real-time embedded benchmarks.

Key words: compiler; syntax tree; real-time; dynamic voltage scaling; low-power; RTLPower

摘 要: 动态电压调节是一种有效的低功耗技术.使用这种技术,编译器指导的动态电压调节能够有效地降低系统功耗.提出了基于语言语法树的实时动态电压调节低功耗算法.该算法在静态程序最差时间分析方法的辅助下,通过在程序内部自动插入电压调节代码来实现电压调节.在 RTLPower(real-time low-power)实时低功耗系统上完成了算法的实现,对嵌入式测试,程序集的初步测试证明该算法最大可节省 50%的能量消耗.

关键词: 编译器;语法树;实时;动态电压调节;低功耗;RTLPower

中图法分类号: TP314 文献标识码: A

CMOS 技术是现代计算机系统的基础.CMOS 的尺度缩放(MOS scaling)不断推动着计算机性能的提高.CMOS 技术的发展面临着越来越严峻的问题,系统功耗问题就是亟待解决的问题之一[1].功耗的急剧增加提高了芯片的封装和制冷成本.高温环境下运行增加了芯片的失效率,导致计算机系统的稳定性下降.

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2004AA1Z2210, 2002AA1Z2101 (国家高技术研究发展计划(863))

作者简介: 易会战(1976－),男,河北肃宁人,博士生,主要研究领域为低功耗编译优化,计算机体系结构;陈娟(1980－),女,博士生,主要研究领域为低功耗编译优化;杨学军(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为并行体系结构,并行编译,并行操作系统;刘喆(1980－),男,硕士生,主要研究领域为低功耗编译优化.

嵌入式移动计算技术是芯片行业最活跃的领域.嵌入式的移动设备往往依靠电池供电,电池的供电时间是它们的重要参数之一.与半导体技术的发展速度相比,电池技术发展慢得多[2],未来的移动设备必须在有限能量供应下发挥更大的效能,这对系统功耗提出了高要求.

IT 行业的设备消耗了大量能量,并且呈逐年增长的趋势[3].大量的能量消耗要求系统采用有效的管理策略以提高能量的使用效率.无论是嵌入式移动设备,还是桌面系统,功耗问题都是十分重要的问题.

动态电压调节是一种有效的低功耗技术,由于系统运行的能量消耗与运行电压呈平方关系,在运行期间动态改变系统电压可以有效调整系统的功耗.采用软件的电压调节策略提高了电压调节的效果.例如,操作系统具有系统的全局任务信息,可以使用操作系统通过多任务调度动态调整电压.然而,单个任务的执行时间同样存在不确定性,在任务执行期间动态调整系统电压,仍然存在节能空间.并且在实时嵌入式系统环境下,如果只有单个任务执行,操作系统不能通过多任务调度调节电压.完成任务内的电压调节需要借助于编译技术,在程序内部插入电压调节点,根据当前时间和任务的运行需求调整电压.

对于实时程序的电压调节,除了要满足降低系统功耗的需求以外,还必须保证程序在截止时间内完成,超过截止时间将会产生不可预知的后果,这就需要对应用的最差时间进行分析.当前程序的最差时间分析方法有两种:统计执行和静态估计.统计执行方法给程序提供各种输入,测试程序的运行时间,大致找到最差执行时间;静态估计方法采用程序分析技术分析程序的运行时间,可以正确给出程序的最差运行时间,但是由于现代复杂的计算机系统结构,这种方法往往给出的是很悲观的最差执行时间.

本文提出了基于语言语法树的实时动态电压调节低功耗算法,该算法通过在任务内部自动插入电压调节代码实现电压调节.算法通过编译器前端分析高级语言,生成语法树结构,依据语句权重阈值的选择策略选择电压调节点,然后分析语法树查找辅助调用点,根据调节点和辅助调用点所在的语法结构插入电压调节代码或者辅助信息,然后将程序以源代码输出,自动生成具有实时电压调节能力的程序.算法的整个过程在语言的语法树上完成,仅仅需要很少的处理器电压调节手段的信息,具有很好的可移植性.基于静态最差时间分析技术,该算法应用于实时环境下,综合了代码插入、估计程序最差时间和对程序最差时间的回填.在 RTLPower 实时低功耗模拟环境下完成了算法的初步实现,模拟结果证明,该算法有效降低了系统功耗.

本文第 1 节给出动态电压调节技术的相关研究工作.第 2 节介绍实时电压调节算法的整体结构和本文研究基于的最差时间分析技术.第 3 节对算法的一些具体问题进行详细介绍.第 4 节给出算法的模拟结果.第 5 节是结论.

1 相关工作

系统的动态功耗和电压成二次方关系,降低供应电压和运行频率可以降低系统的动态功耗[4].程序运行需要不同的处理能力,动态电压调节技术可以根据需求动态调节电压,用最低的能量消耗运行.具有动态电压调节能力的系统一般可以在几个离散频率电压层次上运行,软件可以根据负载需求在几个电压值之间进行动态调节.为了保证系统正确运行,降低电压的同时一般伴随着减少处理器的运行频率.电压层次切换存在一定的能量开销和时间开销.支持动态电压调节的实用的处理器包括 Transmeta Crusoe,Intel Xscale 和 AMD K6-IIIE+.

操作系统指导的动态电压调节技术已经有很多研究.文献[5]最早提出假定 CPU 具有动态频率和电压调整能力下如何提高系统能量效率的方法.文献[6]对操作系统管理的动态频率和电压调节技术进行了比较全面的总结.

近年有大量编译指导的动态电压调节算法方面的研究工作,特别是实时算法.文献[7,8]研究了任务内的 DVS 算法,将程序运行分为若干区域,根据不同区域的计算速度和截止时间的关系,对频率进行动态设置.文献[9]使用 0-1 整数规划研究了当电压层次为几个离散的等级、程序分为几个有限的区域的情况下如何指派电压以达到各种优化目标函数的要求.文献[10]假定程序经常以平均时间运行,将初始频率设定为平均速度,运行时发现程序不能满足最后时限的要求时再逐渐提高频率.文献[11]讨论了如何使用程序的 profile 信息来提高程序电压调节的效果.文献[12]为了解决电压调节的粒度问题,提出协同操作系统和编译技术解决任务内的电压调节.文献[13]基于现有存储系统和 CPU 的性能差异,针对内存受限的程序执行区域,降低 CPU 运行电压,节省

CPU 的能量消耗.文献[14]总结了低功耗编译优化技术,提出在具有动态电压调节和部件关闭能力的系统上进行低功耗编译优化研究的分析模型,归结出部件使用局部化的概念.

过去的算法研究或者仅仅停留在理论分析的角度,或者并不适应新的系统环境.例如,文献[7]较早提出编译指导的实时动态电压调节,提出了 DPM-P,DPM-G 和 DPM-S 三种动态调节策略,具有一般的普适性,但是没有分析在编程语言中如何插入电压调节点完成动态电压调节.文献[8]仅仅利用条件和循环结构可能产生的程序时间变化调整电压,没有讨论函数引入的多次调用的情况,这导致该方法不适合处理大的程序,并且,基于当前复杂的体系结构和现有的静态程序最差时间分析方法,程序实际运行和最差时间估计仍然有很大误差,不可能完全准确地得到程序的最差执行时间,仅仅考虑控制结构引入的时间变化是不够的.文献[12]为了控制动态调节点的数目,减少电压调节的开销,提出使用编译技术和操作系统协同完成调度,使问题进一步复杂化.以往的研究大都基于程序的控制流图,涉及到较多的后端信息;没有考虑如何结合程序的最差时间分析技术,往往只是仅通过简单的测试运行来分析程序的最差执行时间.

2 基本动态电压调节算法

在实时环境下,程序运行存在截止时间(deadline),超过截止时间的运行会导致难以预计的后果.但是实时应用在截止时间之前提早完成也没有任何收益,可以通过降低电压频率降低功耗,使程序尽可能地接近截止时间完成.

在程序的初始阶段,需要设置程序的初始运行频率,满足程序完成的截止时间.进行动态电压调节导致程序的最差运行时间 WCET(worst-case execution time)随频率变化而改变,而最差执行周期 WCEC(worst-case execution cycle)不发生改变,因此本文使用 WCEC 估计初始频率.采用静态时间估计方法能够获得程序的 WCEC,然后利用设定的截止时间 D,可以得到程序的初始频率.本文假定处理器可以在一个极大值 $f_{\max}$ 和一个极小值 $f_{\min}$ 之间连续变化,实际的处理器只需根据计算结果设定最接近的电压频率层次.

当程序运行到达电压调节点,根据程序剩余的最差执行周期 RWEC(reduced worst-case execution cycle)、调节点的当前时间 Curtime 以及电压调节开销 Overhead,计算下面程序运行的频率.频率的设定方法为

$$f_{next} = RWEC/(D - Curtime - Overhead) \tag{1}$$

初始频率和调节点的频率设定方法如下:

```
1  D=截止时间
2  WCEC=程序最差执行周期数
3  f_initial=计算初始频率(WCEC/D)
4  if (f_initial>f_max)
5     不能在截止时间完成,stop
6  else if (f_initial<f_min)
7        f_initial=f_min
8  setFreq(f_initial)
```

```
1  Curtime=当前时间
2  RWEC=程序剩余最差执行周期数
3  Overhead=电压调节开销
4  f_next=RWEC/(D–Curtime–Overhead)
5  if(f_next>f_max)
6     不能在截止时间完成,stop
7  else if(f_next<f_min)
8        f_next=f_min
9  setFreq(f_next)
```

除了 setFreq 函数,其他都不涉及系统体系结构信息.setFreq 函数涉及系统的电压调节机制,不同系统的方式不同,算法移植仅仅需要更改该函数,使用相应体系结构的最差时间分析方法.本文采用 gcc 的内嵌汇编机制,利用内存端口映射手段实现频率调节,频率的重新设定导致系统电压和频率的调节.

频率调节存在时间和能量开销,这与系统的初始电压 V_{DD2},结束电压 V_{DD1},转换电容 C 等参数相关,文献[15]给出了时间开销公式和能量开销公式:

$$t_{TRAN} = 2 \cdot C \cdot \left| V_{DD2} - V_{DD1} \right| / I_{\max} \tag{2}$$

$$E_{tran} = (1 - \eta) \cdot C \cdot \left| V_{DD2}^2 - V_{DD1}^2 \right| \tag{3}$$

一般频率转换的开销在 μs 和 μJ 的量级.

上述方法尚未解决的问题是 WCEC 和 RWEC 的估计以及调节点的选取问题.对于顺序程序结构,每个点的 RWEC 是确定的值,使用最差时间分析工具确定 RWEC 比较简单.但是当引入循环和函数调用时,情况就复杂得

多.如果调节点在循环内部,每次循环,程序的 RWEC 都不同;而函数结构的 RWEC 在不同调用点也不同,必须恰当地更改代码结构,正确计算每点的 RWEC.由于电压调节存在时间开销和能量损失,调节点的选择也存在问题,过细粒度的调节导致电压调节开销抵消了能量节省.下一节详细介绍存在循环和函数调用情况下,电压调节点的 RWEC 计算方法和估计方法,并且给出调节点的选择策略.

下面给出程序的最差执行周期估计方法.实时系统的研究不可避免地要涉及到最差执行时间分析问题,在最早的体系结构中,程序的运行时间是比较容易估计的,时间的不确定性主要源于程序的条件控制结构,随着体系结构的发展,多级内存结构、流水、多流出的处理器使程序的最差时间分析变得复杂,准确分析程序的运行时间更加困难.随之出现了各种考虑低层硬件特性的静态估计方法,例如考虑多级 cache、分支预测和流水线等低层特性,文献[16]总结了最差时间分析的研究工作.本文的工作基于文献[17,18]的最差时间分析研究、静态 cache 模拟技术考虑了指令 cache 和分支硬件对程序运行时间的影响,通过建立程序的上下文树结构完成最差时间分析.图 1 是典型的代码和相应的最差时间分析的上下文树,带有循环执行次数注释的高级语言,编译生成语法树,然后根据程序的运行关系生成上下文树,上下文树对应程序的执行过程.通过编译工具为每个叶节点找到对应代码的实际运行地址,通过数据流方法获得 cache 和 BTB 的信息,分析叶结点的最差执行周期,然后从叶节点逐层向上得到父节点的最差执行周期,最终完成整个程序执行周期的估计.根据估计过程可以看出,直接采用基于上下文树的静态估计方法,可以计算程序的 WCEC,但是不能计算 RWEC.

```
int f1(int a)
{
  if(a>5)
    return a;
  return 0;
}
int main()
{
  int i,j,a;
  a = 0;
  f1(a);
  for(i=0; i<100; i++) [100]
    for(j=0; j<10; j++) [10]
      a++;
  f1(a);
  return 0;
}
```

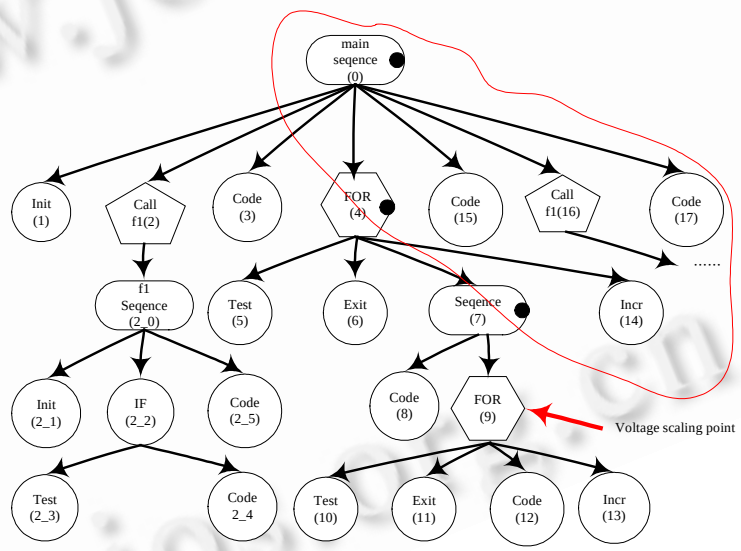


Fig.1 The representative code and the corresponding context tree for time estimate

图 1 典型代码和对应的时间估计上下文树

3 动态电压调节算法相关问题

根据基本算法的分析,完成实时动态电压调节需要估计每个电压调节点的 RWEC,并且需要考虑调节点的选择策略.电压调节点可以插入程序中的任意位置,包括线性代码结构、循环结构和函数调用.循环结构中的电压调节点 RWEC 在不同迭代存在变化,函数调用中的电压调节点在函数调用的不同调用点其 RWEC 也不同.电压调节点引入了时间开销和能量损失,必须采用恰当的策略控制调节点的位置和数目.

3.1 循环结构内的RWEC计算和估计

对于插入循环内的电压调节点,每次迭代电压调节点的 RWEC 都不同,需要联系迭代变化考虑 RWEC.本文通过参数化的方法表示循环内的 RWEC:

$$RWEC = RWEC_{base} + \sum_{i=0}^{n}(Loop^{i}_{\max\_iteration} - Loop^{i}_{cur\_iteration} - 1)\cdot WCEC^{i}_{loop} \quad (4)$$

其中 $RWEC_{base}$ 为电压调节点在所有循环最后一次迭代的最差剩余周期, $WCEC^{i}_{loop}$ 为第 i 重循环的最差运行周期, $Loop^{i}_{\max\_iteration}$ 为第 i 重循环的最大迭代次数, $Loop^{i}_{cur\_iteration}$ 为第 i 重循环的当前迭代次数.于是,上面的公式给出了不存在函数调用情况下的 RWEC 计算公式.

式(4)中的 $Loop^{i}_{\max\_iteration}$ 参数在程序的循环注释信息中能够获得,而参数 $Loop^{i}_{cur\_iteration}$ 需要更改程序的代码,增加指导信息.我们基于程序的语法树,插入指导代码得到 $Loop^{i}_{cur\_iteration}$ 参数.下面给出了指导的代码,循环的开始增加了对应循环的索引,每次进入一层循环对应索引递增;退出循环,对应索引清零.

指导前的代码结构:

```
Loop1::for(i1...)
    Loop2::for(i2...)

        LoopN::for(iN...)
        电压调节点
        LoopN end

    Loop2 end
  Loop1 end
```

指导后的代码结构:

```
__index1…__indexN=0
Loop1::for(i1...)
   Loop2::for(i2...)

       LoopN::for(iN...)
       电压调节点
       __indexN++
       LoopN end
       __indexN =0
   Loop2 end
   __index2=0
   __index1++
Loop1 end
__index1=0
```

$WCEC^{i}_{loop}$ 可以通过上下文树由叶节点到根节点的计算过程计算得到.由于上下文树的计算方法, $RWEC_{base}$ 不能直接通过上下文树计算获得.为了得到 $RWEC_{base}$ 信息,我们采用了剪枝上下文树的方法.如图 1 所示,假定 For(9)语句被选定为电压调节点,那么只有自由曲线内的节点对计算 $RWEC_{base}$ 有贡献,用黑点标记的节点只需要估计部分子节点的时间.为了使用上下文树估计 $RWEC_{base}$,从调节点开始向上依次遍历父节点,搜索上下文树上父节点的子节点,每次剪枝当前节点逻辑上前面的兄弟节点,保留当前节点和逻辑上后面的兄弟节点,对于循环节点只计算最后一次迭代,然后对剪枝后的上下文树由下而上计算得到 $RWEC_{base}$.

3.2 函数调用的RWEC计算和估计

如果在函数调用中存在电压调节点,每次函数调用的 RWEC 是不同的,需要区分不同的调用上下文.本文采用函数的调用点提示信息计算 RWEC:

$$RWEC = RWEC_{func} + RWEC_{base} + \sum_{i=0}^{n}(Loop^{i}_{\max\_iteration} - Loop^{i}_{cur\_iteration} - 1)\cdot WCEC^{i}_{loop} \quad (5)$$

其中 $WCEC^{i}_{loop}$, $Loop^{i}_{\max\_iteration}$, $Loop^{i}_{cur\_iteration}$ 意义和前面相同, $RWEC_{base}$ 为电压调节点到当前函数结束之间的最差剩余周期, $RWEC_{func}$ 为当前函数调用结束的最差剩余周期.通过 $RWEC_{func}$ 可以区分出不同调用点的 RWEC.以下是基于语法树对函数调用进行的修改方法.

指导前调用结构:

```
f1()
{
电压调节点
}
f2()
{
f1()
f1()
}
```

指导后调用结构:

```
f1(float RWEC)
{
电压调节点
}
f2(float RWEC)
{
RWEC1 = RWEC+RWECa
f1(RWEC1)
RWEC2 = RWEC+RWECb
f1(RWEC2)
}
```

函数 $f1$ 内部存在电压调节点,给函数增加一个参数 RWEC 表示 $f1$ 的 $RWEC_{func}$,当调用 $f1$ 函数时,通过函数参数获得函数 $f1$ 调用结束的最差执行周期.函数 $f2$ 因为包含了对 $f1$ 的调用,也需要相应的参数代表 $f2$ 调用结束的最差执行周期,同时利用 $f2$ 函数内的时间估计得到 $RWECa$ 和 $RWECb$,计算得到两个 $f1$ 调用点的 $RWEC_{func}$ 信息,通过参数传递给 $f1$.

对于选择的每个调节点,算法基于程序的语法树进行搜索,获得调节点所属的函数,遍历语法树,查找函数的调用点;调用点作为新的调节点,继续搜索新的调用点,直到调节点数目不发生变化为止.根据是选择的调节点还是查找到的函数调用点来插入相应信息,完成电压调节点和函数调用点的 RWEC 计算.下面给出上述查找过程的伪码算法.

```
1 listDVSPoint=取初始电压调节点
2 While(listDVSPoint大小变化)
3     For(取listDVSPoint一点dvsPoint)
4         遍历语法树查找dvsPoint所属函数f
5         查找f的所有调用点,生成listCallPoint
6     将listCallPoint中所有的listDVSPoint不存在的点插入listDVSPoint;
7 For(逐点处理listDVSPoint)
8     If(电压调节点)
9         生成RWEC计算信息和电压调节信息
10    Else If(函数调用点)
11        生成RWEC计算信息,更改函数参数和函数调用参数
```

在带有函数调用的 RWEC 计算公式中,各参数的估计与前面循环结构的类似.对 $RWEC_{base}$ 的估计要求在剪枝上下文树时不要跨越调用点,如果函数存在多处调用,需要求解每处调用上下文的 $RWEC_{base}$,使用最大值,这样就能保证所有点的电压调节都满足截止时间.函数调用点的 RWEC 计算和电压调节点的 RWEC 计算是相同的,但是函数调用点不进行电压调节.

3.3 电压调节点的选择策略

电压调节不一定导致功耗降低,过多的调节点会延长计算时间,增加能量消耗.文献[8]为了降低调节点开销,仅仅在出现大的时间变化时才进行电压调节;另外一种策略是固定的中断时间调节电压,需要编译和操作系统相互配合,文献[12]即采用这种方法.

一般来说,由于电压调节存在时间和能量开销,需要在恰当的粒度调节电压,编译器控制调节粒度存在困难.我们提供了在条件语句或者循环语句后插入电压调节点的选择策略,这种方法不能有效控制调节点数.程序

结构是复杂的,存在各种粒度的条件和循环语句,有些运算量小于电压调节代码的运算量,电压调节代码极大地增加了程序的运行周期和能量消耗.

语句数可以作为程序计算量的粗略估计.我们提出了根据语句权重阈值,通过语法树遍历选择调节点的选择策略,下面给出基于语句权重阈值的选择策略.

```
1 stCounter=语句阈值(用户提供)
2 accuCounter = 0  语句数累计计数器
3  选择策略  (叶节点)
4        根据语句权重 accuCounter
5  选择策略  (非叶节点)
6        initCounter = accuCounter
7        accuCounter = 0
8        For (取当前节点子节点遍历)
9           If (accuCounter > stCounter)
10             找到电压调节点,标记当前语句之后为电压调节点
11             accuCounter = 0
12       If (当前语句为循环语句)
            accuCounter*最大迭代次数
13       根据当前节点语句权重递增 accuCounter
14       返回  (initCounter+accuCounter)
```

语句权重近似地表示语句的运行时间,用户根据调节点的需求,估计语句权重阈值 stCounter,提供给系统;系统以深度优先的方法遍历语法树,在叶节点累积语句权重,在非叶节点判断累积的权重是否达到权重阈值,确定是否增加电压调节点.如果达到给定阈值,增加调节点,并清除累积数,否则将自己连同子节点的权重与初始累积权重求和返回父节点.

4 算法实现与模拟

本文的算法在实时低功耗模拟环境 RTLPower(real-time low-power)中进行了实现.RTLPower 实时低功耗模拟环境前端是更改的 HEPTANE 最差时间分析工具[18],后端是更改的 Sim-Panalyzer 性能能量模拟器[19],前端生成的 ARM 可执行程序可以由后端模拟执行.

我们从韩国 SNU(Seoul National University)实时研究组的 SNU-RT 测试程序集[20]中挑选了 3 个典型的程序,对算法进行了分析.这 3 个程序为 adpcm,fft1k 和 matmul 程序.adpcm 是适应差分脉冲编码调制算法的实现,包括数据初始化、数据编码和数据解码 3 个大的阶段,包含 800 行代码,大量循环语句、条件语句和函数调用,数据长度为 2 000.fftlk 是长度为 1 024 的快速傅里叶变换算法.matmul 是矩阵乘程序,初始数据大小为 5×5,我们更改为 20×20 的数据.

下面给出 Sim-Panalyzer 模拟器参数.

取指宽度	1	解码宽度	1	流出宽度	1	提交宽度	1
Ruu 大小	2	Lsq 大小	2	整数 alu	1	整数乘	1
浮点 alu	1	浮点乘	1	内存端口	1	顺序流出	true
存储器	首次访问延迟 64 周期,其他 1 周期,内存宽度 4 字节						
L1 数据 cache	16 组,32 字节块,32 路,1 个周期延迟						
L1 指令 cache	16 组,32 字节块,32 路,1 个周期延迟						
TLB	32 组,4096 字节页大小,32 路,30 周期失效延迟						
时钟频率	频率 50Mhz~200Mhz,核心最高电压 1.8V,I/O 最高电压 3.3V						

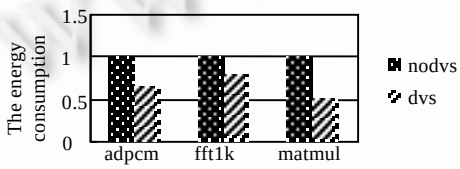
我们根据不同程序的特点选择不同的语句权重阈值,表 1 给出了电压调节的数据.adpcm 包含了两个调节点,分别在数据初始化结束和编码结束,由于初始化阶段在时间分析中占大部分时间,但是分析并不准确,第 2 次电压调节已经超过最低电压.fft1k 和 matmul 均有一个调节点在程序的最主要循环内,进行了多次电压调节.

Table 1 The statistics of voltage adjustment

表 1 电压调节结果统计

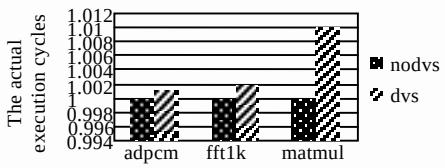
Program	Weight threshold	The number of voltage scaling point	The times of voltage adjustment	Frequencies (Mhz)
adpcm	1 000	2	1	200, 81
fft1k	10 000	1	7	200,179,159,139,119,99,79,59
matmul	400	2	16	200,196,187,178,169,…,84,74,64,54

图 2 给出了电压调节算法的测试结果,所有的结果相对于没有电压调节的结果进行了归一化,nodvs 是不进行电压调节的结果,dvs 是电压调节的结果.图 2(a)是算法的能量消耗结果,从结果可以看出,算法产生了 20%~50%的能量节省,matmul 取得了最好的节能效果.图 2(b)给出了电压调节导致的实际运行周期的变化,该参数可以衡量算法增加的运算量.电压调节的运算量增加很少,fft1k 的运算量增加不到 0.2%,matmul 导致 1%的运算量增加,adpcm 运算量增加了 0.1%.实时电压调节是通过减慢速度降低电压来节省能量的,图 2(c)是电压调节导致的实际运行时间变化,时间增加了 40%~120%,实时算法只要在规定时间内完成,能量消耗是唯一衡量的尺度.我们统计了实际运行时间和截止时间的比率,以示算法对截止时间的使用程度,结果如图 2(d)所示.程序比截止时间提前很多完成,分析发现,截止时间过于宽松是主要原因,因为即使程序完全使用 50Mhz 的最低频率运行,其运行时间也远远小于截止时间,所以算法需要更精确的最差时间分析技术.为了更准确地反映算法的效果,我们使用 50Mhz 的运行频率和实际运行周期估计程序的最长运行时间,然后与电压调节的程序运行时间进行比较,反映程序对节能等级的利用程度,结果如图 2(e)所示.从结果可以看出,算法的利用程度在 35%~50%,fftlk 最小,而 matmul 最大.



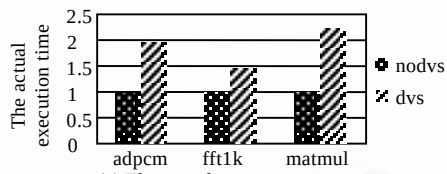
(a) The energy consumption

(a) 电压调节能量变化



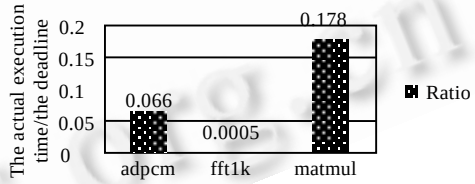
(b) The actual execution cycles

(b) 电压调节实际运行周期变化



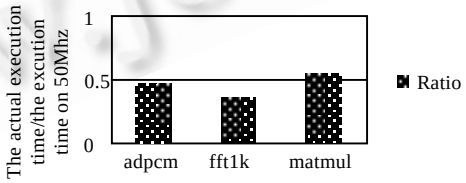
(c) The actual execution time

(c) 电压调节实际运行时间变化



(d) The ratio of the actual execution time versus the deadline

(d) 实际时间/截止时间比率



(e) The ratio of the actual execution time versus the execution time on 50 MHz

(e) 实际时间/50Mhz 运行时间比率

Fig.2 The integrated results of voltage adjustment

图 2 电压调节参数统计

5 结论和未来的工作

本文提出了一种基于语法树的实时动态电压调节低功耗算法.该算法基于程序的语法树,根据语句权重阈值的调节点选择策略,自动插入调节点.在实时条件下,根据静态时间分析方法进行实时电压调节,能够处理循

环和函数调用引入的复杂问题.由于基于程序的语法树结构,该算法具有很好的可移植性.在 RTLPower 实时低功耗系统中实现了该算法,测试结果证明,该算法能够节省 20%~50%的能量消耗.

算法的初步实现和测试分析显示,电压调节效果与程序的时间估计准确度、调节点的设置有密切的关系.寻找更加灵活的调节点选择策略,分析各种因素对电压调节的影响,优化电压调节方法是今后需要进一步研究的工作.

致谢 我们的工作基于 HEPTANE 的实时研究框架和 Sim-Panalyzer 性能功耗模拟框架,感谢 Heptane 研究组和 SimpleScalar-ARM 功耗建模研究组.

References:

[1] Semiconductor Industry Association (SIA). Int'l technology roadmap for semiconductors 2003 edition. http://public.itrs.net

[2] Lahiri K, Raghunathan A, Dey S, Panigrahi D. Battery-Driven system design: A new frontier in low power design. In: IEEE, ed. Proc. of the ASP-DAC/7th Asia and South Pacific and the 15th Int'l Conf. on VLSI Design. Washington: IEEE Computer Society Press, 2002. 261−267.

[3] Mudge T. Power: A first class design constraint for future architectures. In: Valero M, Prasanna VK, Vajapeyam S, eds. Proc. of the High Performance Computing—HiPC 2000: 7th Int'l Conf. German: Springer-Verlag GmbH, 2000. 215−224.

[4] Rabaey JM, Chandrakasan A, Nikolic B. Digital Integrated Circuits: A Design Perspective. 2004.

[5] Weiser M, Welch B, Demers A, Shenker S. Scheduling for reduced CPU energy. In: USENIX, ed. Proc. of the 1st USENIX Symp. on Operating Systems Design and Implementation. New York: The Advanced Computing Systems Association, 1994. 13−23.

[6] Lorch JR. Operating systems techniques for reducing processor energy consumption [Ph.D. Thesis]. Berkeley: University of California, 2001.

[7] Mosse D, Aydin H, Childers B, Melhem R. Compiler-Assisted dynamic power-aware scheduling for real-time applications. In: ACM ed. Proc. of the Workshop on Compilers and Operating Systems for Low-Power (COLP 2000). New York: ACM Press, 2000. 194−203.

[8] Shin DK, Kim JH, Lee SS. Intra-Task voltage scheduling for low-energy hard real-time applications. IEEE Design & Test of Computers, 2001,18(2):20−30.

[9] Saputra H, Kandemir M, Vijaykrishnan N, Irwin MJ, Hu JS. Energy-Conscious compilation based on voltage scaling. In: ACM, ed. Proc. of the Sigplan Joint Conf. on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems. New York: ACM Press, 2002. 1−10.

[10] Gruian F. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In: ACM, ed. Proc. of the Int'l Symp. on Low-Power Electronics and Design. New York: ACM Press, 2001. 46−51.

[11] Azevedo A, Issenin I, Cornea R, Guptam R, Dutt N, Veidenbaum A, Nicolau A. Profile-Based dynamic voltage scheduling using program checkpoints. In: IEEE, ed. Proc. of the Design, Automation and Test in Europe Conf. (DATE). Washington: IEEE Computer Society Press, 2002. 168−178.

[12] AbouGhazaleh N, Mosse D, Childers B, Melhem R, Craven M. Collaborative operating system and compiler power management for real-time applications. In IEEE ed. Proc. of the Real-Time Technology and Application Symp. Washington: IEEE Computer Society Press, 2003. 133−143.

[13] Hsu C-H, Kremer U. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In: ACM, ed. Proc. of the ACM Sigplan 2003 Conf. on Programming Language Design and Implementation. New York: ACM Press, 2003. 38−48.

[14] Yi HZ, Yang XJ. An effective method of low-power compilation optimization: Localizing the use of system units. Journal of Software, 2004,15(10):1451−1460 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1451.htm

[15] Burd TD, Brodersen RW. Design issues for dynamic voltage scaling. In: ACM, ed. Proc. of the 2000 Int'l Symp. on Low Power Electronics and Design. New York: ACM Press, 2000. 9−14.

[16] Puschner P, Burns A. A Review of Worst-Case Execution-Time Analysis (Editorial). Holand: Kluwer Academic Publishers, 1999.

[17] Mueller F. Static cache simulation and its applications [Ph.D. Thesis]. Department of Computer Science, Florida State University, 1994.

[18] Colin A, Puaut I. Worst case execution time analysis for a processor with branch prediction. Real-Time System, 2000,18(2/3): 249−274.

[19] Kim NS, Austin T, Mudge T. Challenges for architectural level power modeling (Power Aware Computing). Kluwer Academic Publishers, 2001.

[20] SNU. SNU real-time benchmarks. http://archi.snu.ac.kr/realtime/benchmark/

附中文参考文献:

[14] 易会战,杨学军.有效的低功耗编译优化方法:部件使用局部化.软件学报,2004,15(10):1451−1460. http://www.jos.org.cn/1000-9825/15/1451.htm