计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140485
52(1):94-104,2015

面向嵌入式系统绿色需求的数据分配方法

何炎祥[1,2] 喻 涛[1] 陈 勇[3] 李清安[2] 江 南[1] 徐 超[1] 文卫东[1]
[1](武汉大学计算机学院 武汉 430072)
[2](软件工程国家重点实验室(武汉大学) 武汉 430072)
[3](中国电子科技集团公司第14研究所 南京 210013)
(yxhe@whu.edu.cn)

Green Demands Oriented Data Allocation for Embedded Systems

He Yanxiang[1,2], Yu Tao[1], Chen Yong[3], Li Qingan[2], Jiang Nan[1], Xu Chao[1], and Wen Weidong[1]
[1](*School of Computer*, *Wuhan University*, *Wuhan* 430072)
[2](*State Key Laboratory of Software Engineering* (*Wuhan University*), *Wuhan* 430072)
[3](*The* 14*th Research Institute*, *China Electronics Technology Group Corporation*, *Nanjing* 210013)

Abstract Green demands, such as energy efficiency and resource utilization, have become critical issues during the design of embedded systems. Data allocation, one of the most important back-end optimization methods of compiler, can largely influence the utilization of energy and resources. This paper proposes a data allocation approach to improve the effective utilization of resources and energy. First, a green evaluation model for data allocation is proposed in this paper. In this model, green indicators are proposed to represent both energy efficiency and resource utilization. Second, based on the evaluation model, an iterative-style multi-objective data allocation approach is proposed to reduce the energy consumption and to balance the resource utilization. This data allocation approach resorts to two common compilation optimization techniques, i.e., exchangeable instructions rearrangement and register reallocation, to improve the green indicators. In addition, an iterative framework is employed to synthesize the exchangeable instructions rearrangement and register reallocation techniques smoothly to improve the green indicators further. Simulation experiment results show that the proposed method can obtain about 23% improvement on average when GCC compiler is the baseline. Therefore, the proposed method can significantly improve the green indicators.

Key words green demands; energy; balanced utilized resources; data assignment; register reallocation

摘 要 能耗和资源等绿色需求是嵌入式系统发展不容忽视的因素.数据分配作为编译后端的重要优化手段,对能耗以及资源的利用率有着重要影响.为提高资源和能源的有效利用率,构建了数据分配过程的绿色评估模型,并以此为指导,提出了一种迭代式多目标分配优化方法,从能源消耗和资源的均衡使用度2个方面出发,利用可交换类指令重排优化和寄存器重分配优化,对总线和存储系统的绿色指标进行改进.模拟实验表明,该方法相对于GCC编译器,能够获得23%左右的绿色指标提升值,为满足更高的绿色需求提供了保障.

关键词 绿色需求;能耗;资源均衡使用;数据分配;寄存器重分配

中图法分类号 TP311; TP314

收稿日期:2014-05-31;**修回日期**:2014-11-05

基金项目:国家自然科学基金重大研究计划重点项目(91118003);国家自然科学基金面上项目(61170022,61373039,61373160);高等学校博士学科点专项科研基金项目(20130141110025l2)

通信作者:文卫东(wwd@whu.edu.cn)

为满足空间和能源的要求,能耗和资源的有效利用率等绿色需求[1]已经成为制约嵌入式系统发展的重要因素.而在嵌入式体统中,除处理器外,总线系统(本文指 CPU 的内部总线)和存储系统是其最为重要的组成模块,解决其能耗和资源的有效利用率对促进嵌入式系统的进一步发展具有重要意义.随着芯片朝着小型化、高集成化方向的快速发展,总线和存储单元的布局越来越密集,各部件直接的交叉影响如总线之间的串扰、密集操作导致局部温度过高等问题越来越明显.同时,随着各种低能耗的非易失性存储技术的出现[2],各存储单元只能进行有限次数的写操作,频繁的操作某个存储单元将严重影响整个存储块的有效利用率.因此,如何均衡使用各总线资源和存储资源,减少其因新的工艺和技术的应用而产生的不利影响是嵌入式系统进一步发展过程中不容忽视的问题.

编译过程的数据分配策略不但能够有效地控制存储资源的使用,决定哪些数据分配到哪些存储资源中.而且不同的数据分配方案也将改变程序生成的最终指令,从而也影响着指令数据总线的传输能耗以及传输耗损等总线资源的使用.特别是编译过程中的寄存器分配优化,不同的优化方案将会使得指令执行过程中指令数据总线上反正频度的大幅度改变,而其对数据溢出方案的不同将对存储系统的读写产生重要影响,从而影响系统的总体能耗以及资源的均衡实用度.同时,寄存器本身也是存储系统的一部分,而且其靠近 CPU 内部,其不均衡实用产生的局部温度过高将大幅度提升系统泄露能耗,降低 CPU 的工作效率.因此,如何在数据分配优化过程中综合考虑存储系统以及总线系统的这些绿色指标是提高系统总体绿色需求的有效途径.

为统一分析数据分配对总线系统和存储系统能耗以及资源使用等绿色指标的影响,本文首先分析了数据分配过程中影响的相关因素,并结合现有的能耗以及均衡度计量方法,构建了一个数据分配过程的绿色评估模型,以指导和评估后续的研究成果.由于寄存器是影响数据分配过程中的主要因素,本文以绿色评估模型为指导,设计了可交换类指令重排优化方法和基于扩展图着色的寄存器重分配优化方法,对编译器后端的寄存器数据进行重新调整,以获得较均衡的寄存器访问频度,减少指令数据总线的动态翻转能耗.同时,为进一步提升系统的绿色指标,本文对以上 2 种优化方法进行迭代分析,使得寄存器和总线使用更加均衡,相邻指令间总线的动态能耗更小.不同于传统的数据分配实施阶段,本文的数据分配方案将在指令调度后进行,以避免指令调度对该优化方法的影响.最后与目前广泛使用的编译器 GCC 进行对比实验.实验结果表明本文方法能够获得 23%左右的绿色指标提升值,从而证明了该方法在提高嵌入式系统整体绿色指标方面的有效性.

1 相关工作

为提高总线系统的能效,减少各总线之间的串扰,研究者们均提出了一些有效的方法.体系结构上主要利用编码[3],选择屏蔽线(shielding)[4-5]、门电路缩放(gate sizing)[6-7]、非均匀布线[8]等技术.在软件方面,特别是在编译优化方面,研究者们主要根据硬件结构的特点和软件优化技术对其进行分析.Lee 等人[9]针对超长指令字(very long instruction word, VLIW)体系结构,提出了水平调度和垂直调度 2 种调度方法来减少总线翻转次数,达到降低总线能耗的目的.Shao 等人[10]证明低功耗指令调度问题是 NP 完全问题,并在 VLIW 体系结构中针对总线翻转次数和调度长度,提出了 3 种启发式调度算法,以尽可能获得较低的功耗.Chabini 等人[11]针对于汇编级基本块内的指令调度,将最小化基本块内总线翻转次数的指令调度问题转化为一个整数线性规划问题,并设计了 2 种启发式调度算法,以获得总线的翻转总次数较少的调度方案.Weng 等人[12]根据选择屏蔽线的特点,利用编译器后端的寄存器重命名的技术减少总线串扰.Kuo 等人[13]综合使用指令调度、寄存器重命名以及空指令填充 3 种后端编译优化方法,以最大程度消除总线之间的串扰.

而在存储系统方面,研究者们一方面使用缓存行移位技术、段交换技术、仅写回修改位技术、数据编码、耗损均衡技术(wear-leveling)等体系结构技术降低其能耗,均衡各资源的使用[13-19].另一方面采用软件的手段对寄存器堆进行优化.Zhou 等人[20]针对分块寄存器堆,首先利用寄存器重命名,以平衡各个分块中寄存器堆的访问频率.然后通过交换变量不同生命期被分配的寄存器,即在生命期不相交的寄存器中调整该生命期所属的寄存器,以平衡对单个寄存器的访问频度.Yang 等人[21]利用编译器对变量生命期的分析,在频繁使用的循环结构入口和出口处保存和恢复循环内不被访问的寄存器,以增加循环内可用的寄存器数,从而可以通过硬件的

移位函数,在循环的不同迭代中循环使用各个寄存器,以均衡使用各个寄存器,达到减少单个寄存器的过度使用.最近,Liu 等人[22]结合硬件移位器,在寄存器分配时针对移位后的结果进行优化,从而能够既减少指令数据传输过程中总线以及解码部件翻转的次数,达到降低能耗的目的,又能通过该移位器从逻辑地址到物理地址的映射,尽可能均匀地使用各个物理寄存器.

现有的编译优化手段虽然可以在一定程度上提升总线系统和存储系统的绿色需求,但大部分手段均是分开考虑,较少综合考虑这 2 个因素的影响.同时,现有的软件优化方法仍然存在一定的改进空间,如基于寄存器重命名技术优化虽然能够通过简单的改变寄存器的名字以达到减少总线串扰的目的,但它并没有充分发挥编译器中丰富的数据流信息及具体指令的特点,损失了对能耗以及寄存器资源均衡访问的更深层优化手段[1].本文将主要针对这些改进空间,结合编译中的数据流分析和具体指令的特点,对存储系统和总线系统的绿色需求进行进一步优化,以最大程度提升软件的绿色指标.

2 数据分配过程绿色指标的评估模型

数据分配是编译器优化过程的重要组成部分,它通过对程序中使用的数据以及系统可用资源的分析,确定将哪些数据存放到哪些寄存器,将哪些数据存放到内存,以保证程序在有限的存储资源中能够顺利地运行.不同的优化目标需要有不同的数据分配方案,如中断较多的嵌入式程序希望使用尽可能少的寄存器,以减少中断保护和中断恢复的开销;而性能优先的优化希望充分使用所有可用寄存器,使其尽可能少地访问内存,提高执行效率.针对于绿色评估模型中的 2 个指标——能耗指标和资源使用均衡度指标,不同的数据分配优化方法对系统的绿色指标将产生以下 2 个方面的影响.

1) 不同的数据分配方案将产生不同的指令序列,而不同的指令执行序列将会影响指令数据总线的翻转频度以及翻转的均衡度,进而影响总线能耗和单根总线的负载.如何调整数据分配方案、减少总线翻转频度、均衡各根总线负载,以提高总线绿色指标,是数据分配方案不容忽视的问题.

2) 由于程序中数据访问模式的方式不同,不同的数据分配方案将使得不同存储单元访问频度、访问能耗有很大不同.即使是相同的读数据操作,由于温度因素的影响,访问频度较高的存储模块将需要较高的读操作能耗.因此,如何根据具体的数据访问模式合理地调整被访问的存储单元,以均衡各存储单位的访问频度,减少存储单元因过度密集访问而导致的额外能耗以及设备损耗.

综合以上分析,我们将数据分配过程绿色指标的评估模型表示为式(1):

$$W=\alpha(E_{\mathrm{Bus}}+E_{\mathrm{M}})+\beta(\eta_1 S_{\mathrm{Bus}}+\eta_2 S_{\mathrm{M}}), \quad (1)$$

其中,E_{Bus} 表示总线的能耗指标,其值可使用文献[1]所述的总线能耗模型,根据总线翻转模式进行计算,具体如式(2)所示(各符号含义参见文献[1]);E_{M} 表示存储单元的能耗指标,由于存储单元的动态能耗主要是由于读写操作产生的,读写次数越频繁,其能耗也将越大,因此本文将主要根据单次读(写)平均能耗 $\bar{E}_{\mathrm{R}}(\bar{E}_{\mathrm{W}})$ 以及读(写)总次数 $r(w)$,使用式(3)计算;S_{Bus} 和 S_{M} 分别表示总线和存储单元的访问均衡度指标,其值将利用标准差进行计算,如式(4)和式(5)所示,$\bar{r}$,$\bar{w}$ 和 $\bar{b}$ 分别表示单个存储单元平均读次数、平均写次数和单根总线平均翻转次数;r_j,w_j 和 b_i 分别表示第 j 个存储单元的读写次数和第 i 根总线的翻转次数;α 和 β 分别表示能耗以及均衡度这 2 个指标对整个系统的影响程度,其值根据具体应用环境对这 2 个指标的需求程度进行确定;η_1 和 η_2 分别表示总线均衡度以及存储单元均衡度对系统的影响,其值可以根据两者失效率的比值确定.

$$E_{\mathrm{Bus}}=\sum_{j=1}^{n}(1+C_{\mathrm{eff},j}\times\lambda)\times C_{\mathrm{L}}\times\Delta V_j\times V_j; \quad (2)$$

$$E_{\mathrm{M}}=r\times\bar{E}_{\mathrm{R}}+w\times\bar{E}_{\mathrm{W}}; \quad (3)$$

$$S_{\mathrm{M}}=\sqrt{\sum_{j=1}^{n}(r_j-\bar{r})^2\Big/n}+\sqrt{\sum_{j=1}^{n}(w_j-\bar{w})^2\Big/n}; \quad (4)$$

$$S_{\mathrm{Bus}}=\sqrt{\sum_{i=1}^{n}(b_i-\bar{b})^2\Big/n}. \quad (5)$$

3 多目标数据分配优化方法

3.1 总体优化框架

为提高总线以及存储系统的绿色需求,本文分 2 个层次对编译器后端的数据分配进行迭代式优化.首先,利用程序执行的 profiling 信息(包括寄存器及其邻近寄存器访问频度、指令执行次数等信息),对可交换指令类的指令进行操作数重排.其中,

可交换指令类是指这样一类多操作数指令,当交换其操作的某 2 个或 2 个以上的操作数时,指令执行的结果仍然保持不变. 如大多数目标指令集中的加法指令、乘法指令、逻辑运算指令等均属于这类指令,而如减法指令、移位指令等则不属于该类指令.

在获得待优化数据后我们先对可交换类指令的操作数进行重排,使其指令切换代价尽可能小. 如图 1 所示,我们将图 1(a)中"i2"指令的后 2 个操作数进行交换,将原来的 6 次翻转减少到 4 次翻转,如图 1(b)所示.

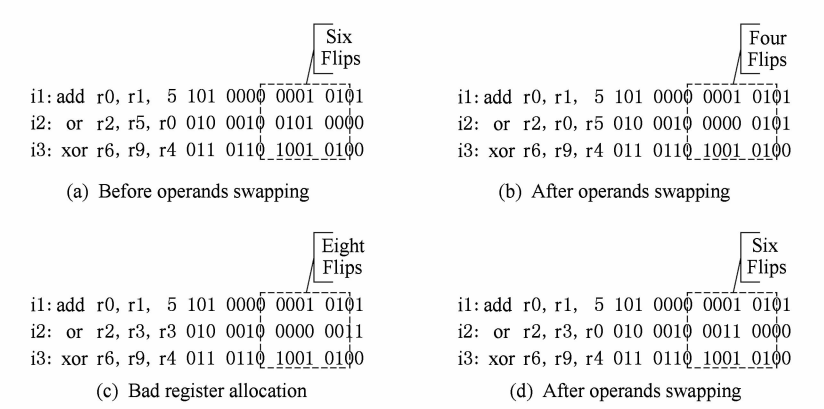


Fig. 1 Examples of optimization.

图 1 优化示例

接着,在考虑寄存器访问模式(即寄存器本身访问频度及其临近寄存器访问频度)的情况下,对重排过可交换类指令集的程序进行寄存器重分配,使得在保证溢出数据不大量增加(本文对于溢出数据不作优化)的前提下,一方面使每个寄存器访问尽可能均衡,另一方面使指令间切换时的总线翻转尽可能少而均衡.

此外,随着寄存器分配方案的不同,可交换指令类操作数的其他重排方式可能要优于现有的重排方式. 如图 1 所示,当由于程序其他部分的影响,将寄存器"r5"与寄存器"r3"互换后,图 1(b)中"i2"指令处虚框内的翻转次数将增加为 8 次(如图 1(c)所示),此时若将"r0"与"r3"再次交换,则可将其翻转次数降低为 6 次(如图 1 (d)所示). 因此,为进一步加强优化效果,我们采用迭代编译优化的方法,直到满足连续迭代次数达到某个阈值 n(其中 n 根据具体的应用需求而定,为保证算法效率,取值通常不超过 30)时结束优化,转入其他优化或生成相应优化后代码. 数据分配过程总体优化框架如图 2 所示.

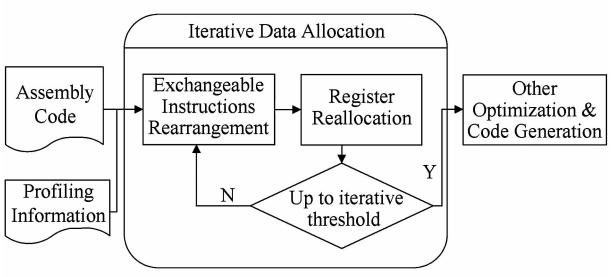


Fig. 2 The framework of data allocation optimization.

图 2 迭代式数据分配优化框架

3.2 可交换类指令操作数重排优化

在对交换类指令操作数进行重排的过程中,由于只是调整单条指令操作数的分布,不能变换使用的寄存器,因此,我们将主要使用总线绿色评估模型的构建指令操作数重排的策略,具体计算方法如式(6)所示:

$$Cal_{b_i}=\begin{cases}\sum\limits_{pB\in pre_b}[Eval(b_1,pB_{\text{last}})]\times f(pB,b), & \text{if } i=1;\\ Eval(b_{i-1},b_i), & \text{if } i>1.\end{cases} \tag{6}$$

在式(6)中,b_i 表示基本块 b 中的第 i 条指令;pre_b 表示基本块 b 的前驱基本块的集合;pB_{last} 表示基本块 pB 的最后一条指令;$f(pB,b)$ 表示从基本块 pB 流入基本块 b 的频度;$Eval(i,j)$ 表示指令 i 与指令 j 相对于总线的绿色评估值,其计算方法如式(7)所示:

$$Eval(i,j)=\alpha\times E_{Bus}(i,j)+\beta\times S_{Bus}(i,j). \tag{7}$$

在构建了操作数重排策略后,采用如算法 1 所示的操作数重排算法依次对每个基本块进行处理.

算法 1. 指令操作数重排算法.

输入:基本块 b、基本块间分支访问频度 $f:B\rightarrow B\rightarrow R$;

输出:基本块 b 中指令操作数的排列方式.

① for each $i\in b$ do

② if $i \in SInst$ then
③ $minWeight=\infty$;
④ for each $k \in swaps_i$ do
⑤ $weight=Cal_k$;
⑥ if $weight<minWeight$ do
⑦ $minWeight=weight$;
⑧ $iIns=k$;
⑨ endif
⑩ endfor
⑪ 更新 b 中指令 i 为 $iIns$;
⑫ endif
⑬ endfor
⑭ return b.

该算法自上而下依次遍历基本块内的每条指令(行①).每当遇到可交换指令(行②)时,则遍历其每种操作数重排方案(行④),根据策略评估式,选择评估值最小的操作数重排方案(行⑥～⑨)作为该指令最终的操作数重排方案(行⑪).在该算法中,$SInst$ 表示可交换指令集,$swaps_i$ 表示指令 i 操作数所有功能等价的排列方式组合,如对于加法指令"add $r1$, $r2$,$r3$",$swaps_i$={"add $r1$,$r2$,$r3$","add $r1$,$r3$,$r2$"}.

通过该指令操作数重排优化,可以获得在当前数据分配方案下绿色指标尽可能好的操作指令.

3.3 基于 APIG 图的寄存器重分配方法

为提高存储系统以及总线的绿色指标,本文将结合数据访问模式信息,对编译器后端生成的目标机器指令进行寄存器重分配,其基本步骤如图 3 所示:

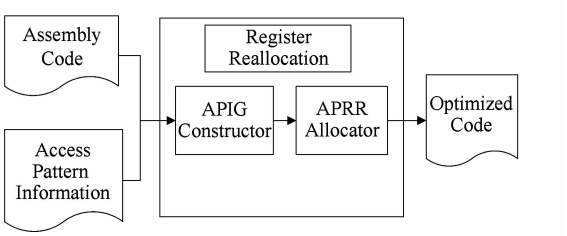


Fig. 3 Register reallocation optimization.
图 3 寄存器重分配优化

在图 3 所示的步骤中主要包括 2 个部分:访问模式信息图(access pattern information graph, APIG)的构建和基于访问模式的寄存器重分配方法(access pattern based register allocation, APRR).APIG 主要用于描述寄存器访问模式信息,包括访问频度、访问次序等信息.为提高该信息的精确度,本文将结合程序执行时的动态 profiling 信息进行统计.在 APRR 分析中,我们将根据 APIG 图中数据之间的关系以及绿色评估指标对寄存器进行重分配,使得最终的程序运行过程中减少总线以及存储器的能耗,提高其有效使用时间,获得较大绿色指标的提升.下面我们将分别介绍 APIG 图的构建方法以及寄存器重分配的 APRR 方法.

3.3.1 访问模式信息图(APIG)

根据总线以及存储系统的绿色评估模型可知,为提高存储系统的绿色指标,我们需要将数据访问操作尽可能平均地分布在寄存器层次,使得各寄存器的访问频度尽可能均衡;而为了提高总线系统的绿色指标,我们需要考虑相邻指令间寄存器的使用情况,尽可能减少相邻寄存器访问时的总线翻转.此外,寄存器分配仍然需要满足数据流依赖关系,不能将生命期相交的变量存放在相同的寄存器中,以保证程序的正确性.因此,为能够有效地显示以上 3 类信息,指导面向绿色需求的寄存器分配方案,本文对寄存器分配过程中使用较为广泛的冲突图进行了扩展,提出了一种带数据访问模式信息的扩展冲突图模型——访问模式信息图.

访问模式信息图是一个加权的无向图,可形式化地表示为一个四元组 $APIG=(V,E_{\mathrm{I}},E_{\mathrm{N}},W_{\mathrm{E}})$,其中节点集合 V 中的每个节点 $v \in V$ 表示一个待分配数据;E_{I} 表示所有冲突边的集合,其中每条边 $e(u,v) \in E_{\mathrm{I}}$ 表示节点 u 和节点 v 不能分配给同一个寄存器;E_{N} 表示所有相邻访问数据边的集合,其中每条边 $e'(u',v') \in E_{\mathrm{N}}$,表示节点 u' 和节点 v' 有可能在相邻时钟周期内在指令数据总线的相邻位置进行访问,$w(e') \in W_{\mathrm{E}}$ 表示这 2 个节点相邻访问的频度.

根据 APIG 图的定义,由于其源自于冲突图,因此能够很好地表达节点的数据依赖关系,从而保证数据分配的正确性.其次,由于 E_{N} 以及 W_{E} 的引入,各变量之间的依次访问频度关系也能够得到有效的表达.而对于每个节点 v 的访问频度本身的访问频度,则可以利用其相邻边的访问频度,通过式(8)很方便地求得.综合以上分析,该图能够较好地适应绿色需求的寄存器重分配方法的表达:

$$f_v=\left\lceil \frac{1}{2}\sum_{e(v,u)\in E_{\mathrm{N}},u\in V} w(e) \right\rceil. \tag{8}$$

在构建 APIG 时,本文以目标指令集表示的汇编程序为分析对象,对每条指令进行分析,以静态单一赋值[23](static single assignment, SSA)的形式,构建基本块为单位的控制流图,确保每个数据在不

考虑循环的情况下只被写入一次,并将该程序中使用的寄存器作为节点加入到 APIG 图的节点集中(算法 2 行①～⑧).接着,根据文献[23]中所描述的数据流分析方法,对程序进行数据流分析,获得每个待重分配的寄存器的生命期(行⑨～⑩).然后,以数据流分析获得的每个寄存器的生命期为依据,添加冲突边(行⑪～⑮).在构建完冲突边后我们依次遍历程序执行的动态 profiling 信息,添加邻近访问边 E_N 及其访问频度权值 W_E(行⑯～⑲).最后返回构建的 APIG 图供寄存器重分配使用. APIG 图的详细构建方法如算法 2 所示.

算法 2. APIG 构建算法.

输入:目标机器指令集表示的汇编程序 S、数据访问模式信息 P:$V\rightarrow V\rightarrow W_E$;

输出:$APIG=(V,E_I,E_N,W_E)$.

① 根据 S 构建程序的控制流图 $CFG(V',E')$
② for each $v\in V'$ do
③ 将 v 转成 SSA 形式;
④ for each $i\in v$ do
⑤ 获得 i 中使用的寄存器集合 Reg_i;
⑥ 将 Reg_i 中寄存器加入到集合 V 中;
⑦ endfor
⑧ endfor
⑨ 获得控制流图 CFG 的数据流分析结果 DS;
⑩ 根据 DS 获得每个寄存器的生命期 $Livereg$;
⑪ for each $i,j\in Livereg$, $i\neq j$ do
⑫ if $Livereg_i\cap Livereg_j\neq\varnothing$ then
⑬ $E_I\leftarrow e(i,j)$;
⑭ endif
⑮ endfor
⑯ for each $\langle i,j,w_{i,j}\rangle\in P$ do
⑰ $E_N\leftarrow e'(i,j)$;
⑱ $W_E\leftarrow w_{i,j}$;
⑲ endfor
⑳ return $APIG=(V,E_I,E_N,W_E)$.

在算法 2 中,$Livereg_i$ 表示寄存器 i 的生命期,输入的访问模式函数 $P(i,j,w_{i,j})$表示寄存器 i 与寄存器 j 在相邻一个时钟周期内访问的次数 $w_{i,j}$.

3.3.2 APRR 寄存器重分配算法

为使得寄存器分配过程能够朝着提高总线和存储系统绿色需求的方向发展,我们需要设计一套合理的编译器可控的启发式分配策略指导寄存器分配.根据前面部分对数据分配过程中编译器可控因子的分析,对于当前需要分配的 APIG 图中的 2 个节点 i 和 j,本文设计了如式(9)所示的启发式策略.从当前可选寄存器集合 $Regs$ 中选择 2 个寄存器 r_i 和 r_j 分别分配给这 2 个节点,使得该启发式评估策略值 $H(i,j)$最小:

$$H(r_i,r_j)=(\alpha\times Ham(r_i,r_j)+\beta\times CT(r_i,r_j))\times w(i,j)+\eta\times Trans_{vB}(r_i,r_j)\times w(i,j)+\gamma\times\sum_{r\in\{r_i,r_j\}}F_r\times\mathrm{abs}(F_r),\tag{9}$$

其中,$Ham(r_i,r_j)$表示寄存器 r_i 和 r_j 编号的海明距离,如 $Ham(r_i,r_j)=2$;$Crosstalk(r_i,r_j)$表示寄存器 r_i 和 r_j 之间的总线串扰评估值,其具体计算方法如文献[1]所示;$w(i,j)\in W_E$ 即为 $APIG$ 图中邻近访问边 E_N 的访问频度,表示节点 i 和 j 邻近在相同总线位置顺次访问的总次数;$Trans_{vB}(r_i,r_j)$表示 r_i 和 r_j 是否有可能引起当前翻转频度最大的总线 vB 发生翻转,如式(10)和式(11)所示,该因子主要用于控制单根总线的访问频度;$r_i(k)$表示用二进制表示的寄存器编号 r_i 的第 k 个比特位的值;F_r 表示已分配给寄存器 r 的所有节点的访问频度之和与节点平均访问次数之间的差值,主要用于平衡各寄存器的访问频度.α,β,η,γ 是对于因子的平衡参数,用于控制各成分对总体绿色指标的影响程度,可以根据具体应用对各成分的敏感程度确定取值.

$$\delta_{i,j}^{k}=r_i(k)-r_j(k);\tag{10}$$

$$Trans_{vB}(r_i,r_j)=\begin{cases}0, & \delta_{i,j}^{vB\%m}=0;\\ 1, & \text{others.}\end{cases}\tag{11}$$

在获得 APIG 图构建方法以及以上启发式寄存器绿色分配策略后,本文设计了如算法 3 所示的面向绿色需求的寄存器重分配算法 APRR.

算法 3. APRR 算法.

输入:每个函数的 $APIG(V,E_I,E_N,W_E)$,可用寄存器集合 $Regs=\{r_1,r_2,\cdots,r_n\}$;

输出:寄存器重分配策略 M:$V\rightarrow Regs$.

① $listE=sortDec(E_N,W_E)$;
② for each $v\in V$ do
③ $Reg_v=Regs$;
④ endfor
⑤ while $listE\neq\varnothing$ do
⑥ $e(u,v)=listE_0$;
⑦ if 有且仅有一个节点 u 赋值给寄存器 r_i then
⑧ $H_{\min}=\infty$;
⑨ for each $r\in Reg_v$ do
⑩ if $H_{\min}>EvlO(r_i,r)$ then

⑪ $r_j=r, H_{\min}=EvlO(r_i,r)$;
⑫ endif
⑬ endfor
⑭ $M\leftarrow(v,r_j)$;
⑮ for each $e'(v,u')\in E_I$ do
⑯ 从 $Reg_{u'}$ 中删除 r_j;
⑰ endfor
⑱ else if 这 2 个节点均未被赋值 then
⑲ $H_{\min}=\infty$
⑳ for each $r_i\in Reg_v$ do
㉑ for each $r_j\in Reg_u$ do
㉒ if $H_{\min}>EvlT(r_i,r_j)$ then
㉓ $r_m=r_i, r_n=r_j, H_{\min}=EvlT(r_i,r_j)$;
㉔ endif
㉕ endfor
㉖ endfor
㉗ $M\leftarrow(u,r_m), M\leftarrow(u,r_n)$;
㉘ for each $e'(v,u')\in E_I, e''(u,v')\in E_I$ do
㉙ 从 $Reg_{v'}$ 中删除 r_m, 从 $Reg_{u'}$ 中删除 r_n;
㉚ endfor
㉛ endif
㉜ endwhile
㉝ return M.

由于邻近访问边的权值越大,表明该边的 2 个节点依次访问越频繁,因此为减少总线翻转次数,我们希望这样的 2 个节点尽可能放在相同的或翻转次数极少的寄存器对中. 同时,由于溢出代码的引入不但会增加额外的存储层次访问开销,而且会改变顺次执行的指令序列,导致重构 APIG 图,为尽可能提高寄存器层次的访问,减少低存储层次的高能耗的访问以及不必要的算法开销,我们希望不产生新的溢出代码. 为此,本文在传统编译器已成功分配寄存器的基础上进行相关寄存器重分配,以避免新的溢出代码的产生. 在获得按边权值从大到小排列的邻近访问边的集合后,我们先将每个节点的可选寄存器集合设定为该体系结构中所有可用寄存器的集合 $Regs$,然后依次遍历每条边,并依次分配每个节点.

在算法 3 所示重分配算法中,首先对程序中的每个函数进行分析,构建对应的 APIG 图,并按照该图中邻近访问边 $e\in E_N$ 权值递减的顺序对各邻近访问边进行排序(行①),并初始化每个节点可分配寄存器为所有可用寄存器集合 $Regs$(行②~④). 在对邻近访问边进行处理时,首先判断这 2 个节点是否已经被分配给寄存器. 如果这 2 个寄存器中有一个节点已经分配,则对该边的另一个节点进行搜索,查找在其中一个节点 u 已经分配给寄存器 r_i 的情况下,根据式(12)计算节点 v 可用的最优分配方案并保存(行⑧~⑭). 在式(12)中,M_K 表示 M 这个寄存器分配策略映射表中所有已分配节点的集合,$M_{u'}$ 表示节点 u' 分配的寄存器编号. 在确定节点 v 的分配方案后,修改与该节点相关联的冲突边的其他节点的可选寄存器集合(行⑮~⑰). 如果这 2 个节点 u 和 v 都未被分配,则依次搜索 2 个节点集中的可选寄存器 r_i 和 r_j,根据式(13)查找其中最优的分配方案保存(行⑲~㉗),同样修改对应的冲突边上的其他节点的可选寄存器集合(行㉘~㉚). 如果待分析边的 2 个节点均已经分配了寄存器,则不处理,直接进入下一条边的分析. 当所有邻近访问边分析完成后则将保存的分配方案 M 返回.

$$EvlO(r_i,r)=H(r_i,r)+\sum_{u'\in M_K\&\&e(u',v)\in E_N}(H(r,M_{u'})-\gamma\sum_{x\in\{r,M_{u'}\}}F_x\times\mathrm{abs}(F_r)); \tag{12}$$

$$EvlT(r_i,r_j)=EvlO(r_i,r_j)+\sum_{v'\in M_K\&\&e(u,v')\in E_N}(H(r_i,M_{v'})-\gamma\sum_{x\in\{r_i,M_{v'}\}}F_x\times\mathrm{abs}(F_r)). \tag{13}$$

4 实验与结果分析

4.1 实验构建

本文以 5 级流水线顺序执行的 StrongARM 为目标芯片,利用交叉编译器 arm-linux-gcc-3.3.2 以及修改后的模拟器 simplescalar-arm-0.2[24] 构建实验环境,然后通过对比原始编译器优化结果以及采用本文算法优化后的结果,以评估本文方法对系统绿色指标的提升能力. 其中选取的测试用例源于广泛使用的嵌入式基准测试用例集 Mibench 和 Mediabench,具体实验环境配置如表 1 所示.

之所以选择 StrongARM:一方面在于它是经典的嵌入式系统平台,已经获得广泛的应用;另一方面拥有 GCC,simplescalar 等实验环境的支持,能够较方便地进行相关模拟实验. 但数据分配是所有体系结构程序运行不可或缺的步骤,只要该体系结构存在可交换指令类或者采用基于寄存器的访问模式均可采用本文所示方法进行.

Table 1 Experimental Setup

表 1 实验环境配置

Item	Value
Compiler	arm-linux-gcc 3.3.2
Optimization Option	-O3
Target Platform	StrongARM
Simulator	simplescalar-arm-0.2
Benchmark	Mibench, Mediabench
Number of GP Registers	16
Cache Word Size/B	32

在实验过程中,本文首先使用交叉编译器 arm-linux-gcc 对源程序进行编译,生成目标平台的汇编代码和二进制代码.然后对 StrongARM 平台模拟器 simplescalar-arm 进行修改,获得该程序缓存单元访问轨迹以及对应的 profiling 信息.接着,利用这些 profiling 信息,根据本文提出的数据重分配算法对汇编代码进行处理,生成新的优化后的汇编代码.由于该实验评估的绿色指标主要集中于总线翻转、寄存器访问次数、缓存单元访问次数等信息,因此可以根据原始执行的轨迹以及 profiling 信息中每条指令的执行次数信息统计出对应的绿色指标评估值,其具体实验方案如图 4 所示:

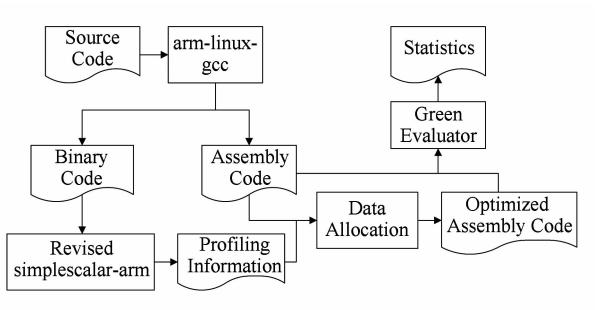


Fig. 4 Experimental scheme.

图 4 实验方案

4.2 实验结果与分析

本文对 Mibench 中的 10 个用例以及 Mediabench 中的 5 个用例进行了实验,图 5 显示了本文算法在无迭代情况迭代 10 次的情况以及迭代 20 次情况下对系统绿色指标提升值的统计结果.

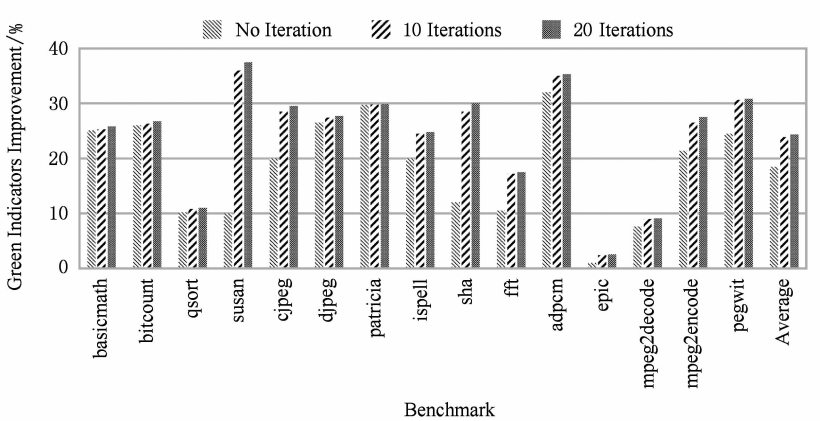


Fig. 5 Improvement of green indicators.

图 5 绿色指标提升值

其中,我们将所有影响因子均设为 0.5(此处假设各部分的影响程度相同,根据应用的不同,该值可取不同值).由图 5 可以看出,本文算法能够较大幅度提升系统的绿色指标,无迭代情况下最高提升幅度约为 32%(如 adpcm),平均提升幅度约为 18%;在迭代情况下,其优化效果获得进一步提升,平均提升幅度约为 23%,相对于无迭代情况下的平均增幅约为 5%,绿色指标最高提升幅度可达 37%左右(如 susan).而且由图 5 可以看出,迭代 10 次与迭代 20 次时其优化效果几乎相同,因而其收敛速度较快,在迭代 10 次时基本可达到温度值.此外,为进一步检测该方法对单个绿色指标的影响,本文分别对其进行了统计分析,其结果如表 2 所示.

表 2 示出采用迭代方法以及不采用迭代方法下,可交换类指令重排优化和寄存器重分配优化对总线能耗、总线访问均衡度以及寄存器访问均衡度 3 个指标的统计结果.其中第 1 列表示测试用例名,第 2~4 列为无迭代情况下本文的优化方法相对于 arm-linux-gcc 最高优化选项的优化方法下获得的总线动态能耗、总线访问均衡度以及寄存器访问均衡度的归一化值.第 5~7 列表示迭代 10 次情况下本文的优化方法相对于 arm-linux-gcc 最高优化选项的优化方法下获得的总线动态能耗、总线访问均衡度以及寄存器访问均衡度的归一化值.由表 2

可以看出迭代优化能够在提升一定程度的优化效果.相对于无迭代情况下,迭代优化下总线的动态能耗平均提升率达到68.31%,总线访问均衡度平均提升率为33.75%,寄存器访问均衡度平均提升率为27.05%(其中,提升率=迭代优化率/无迭代优化率-1).在总的情况下,由于传统编译器主要集中于如何减少需要的资源数目,较少考虑均衡度因素,因此本文算法在均衡度的提升幅度上比较大:对于单根总线访问的均衡度平均提升幅度达到15.06%,最高可达32.13%(如mpeg2encode);而对于寄存器访问的均衡度平均提升幅度达到55.94%,最高提升幅度更可达88.68%(如djpeg).但在总线动态能耗上,在无迭代情况下平均减少率仅为2.43%,在迭代优化时平均优化率也只有4.09%.其主要原因有2个方面:1)能耗问题受重视程度高、研究较多,现有编译器已经在一定程度上对其进行了考虑;2)总线的动态能耗指标同均衡度指标并不总是相辅相成的,对于某些测试用例(如测试用例djpeg,patricia,ispell等),为获得较高的总线访问均衡度以及寄存器访问均衡度(大于50%),需要牺牲较少的(少于5%)能耗指标,以获得总体绿色指标的提升.

Table 2 Comparison of Green Indicators for both Buses and Registers

表2 总线及寄存器绿色指标优化值 %

Benchmark	No Iteration			10 Iterations		
	Bus Dyn Engy	Bus Evenness	Register Evenness	Bus Dyn Engy	Bus Evenness	Register Evenness
basicmath	81.63	76.23	67.26	81.61	76.07	67.26
bitcount	85.10	82.61	56.73	85.05	82.25	56.73
qsort	97.02	97.64	71.80	98.38	94.71	71.80
susan	97.33	97.51	72.33	81.57	85.66	18.62
cjpeg	106.59	93.72	37.60	97.67	83.81	28.72
djpeg	104.57	93.60	11.32	102.17	91.48	11.32
patricia	102.15	81.26	26.38	102.15	81.26	26.38
ispell	107.33	105.40	25.55	104.17	98.82	21.62
sha	93.73	81.40	84.52	91.55	77.39	38.42
fft	98.92	92.17	75.75	98.82	92.11	55.54
adpcm	100.12	76.83	23.62	94.86	74.11	23.62
epic	99.31	95.26	99.16	98.26	91.82	96.08
mpeg2decode	99.76	92.80	84.30	103.38	93.05	75.70
mpeg2encode	92.24	73.84	67.07	102.15	67.87	44.89
pegwit	97.76	90.80	36.13	96.86	83.75	24.14
Average	97.57	88.74	55.97	95.91	84.94	44.06

5 总结与展望

数据分配不但是编译器后端的主要任务之一,而且对程序运行时存储系统和总线系统的绿色指标有着重要影响.本文主要以数据分配为主要研究点,首先分析了该过程对系统绿色指标影响的相关因素,构建了绿色评估指标.接着分别针对可交换类指令、寄存器分配这2个可优化方面,以提高总线系统和寄存器等多个目标的绿色指标,设计了可交换类指令重排优化方法和基于扩展图着色的APRR寄存器重分配优化方法,对编译器后端的寄存器以及栈数据进行重新调整,以获得较均衡的寄存器访问频度和单根总线的负载,减少总线的系统动态翻转能耗.同时,为进一步提升系统的绿色指标,本文对这2种优化进行迭代分析,获得了绿色指标的进一步提升值.与已有的寄存器分配方案不同,本文的寄存器重命名并非最小化寄存器使用数目,而是综合考虑能耗以及寄存器、总线使用的均衡度,扩展的数据依赖关系图很好地表述了资源的访问模式,为数据分配提供了更多可用信息,以指导其获得较好的优化效果.模拟实验结果表明,通过本文的数据分配优化,系统的总体绿色指标平均获得23%左右的提升值.

能耗和资源的均衡使用等绿色需求是未来计算机发展不容忽视的重要方面,数据分配为计算机最重要的资源分配方式,必然能够对其产生重要影响.在今后的工作中,我们将进一步结合新型的体系结构特征,不但针对寄存器数据,还将面向缓存甚至内存中存放的数据,进行更细粒度和更深层次的优化,已获得更有效的数据分配和使用方案,获得系统绿色指标的大幅度提升.

参 考 文 献

[1] He Yanxiang, Chen Yong, Wu Wei, et al. Optimization for green compilations: A survey [J]. Journal of Frontiers of Computer Science & Technology, 2013, 7(8): 673-690 (in Chinese)
(何炎祥, 陈勇, 吴伟, 等. 绿色编译优化策略:研究综述[J]. 计算机科学与探索, 2013, 7(8): 673-690)

[2] Li H, Chen Y. An overview of non-volatile memory technology and the implication for tools and architectures [C] // Proc of Design, Automation and Test in Europe (DATE). Piscataway, NJ: IEEE, 2009: 731-736

[3] Wu X, Yan Z. Efficient CODEC designs for crosstalk avoidance codes based on numeral systems [J]. IEEE Trans on Very Large Scale Integration Systems, 2011, 19(4): 548-558

[4] Zhang J, Friedman E G. Effect of shield insertion on reducing crosstalk noise between coupled interconnects [C] // Proc of the 2004 Int Symp on Circuits and Systems (ISCAS). Piscataway, NJ: IEEE, 2004: II529-II532

[5] Mutyam M. Selective shielding technique to eliminate crosstalk transitions [J]. ACM Trans on Design Automation of Electronic Systems, 2009, 14(3): 43:1-43:20

[6] Gupta U, Ranganathan N. A utilitarian approach to variation aware delay, power, and crosstalk noise optimization [J]. IEEE Trans on Very Large Scale Integration Systems, 2011, 19(9): 1723-1726

[7] Hanchate N, Ranganathan N. Simultaneous interconnect delay and crosstalk noise optimization through gate sizing using game theory [J]. IEEE Trans on Computers, 2006, 55(8): 1011-1023

[8] Macii E, Poncino M, Salerno S. Combining wire swapping and spacing for low-power deep-submicron buses [C] // Proc of the IEEE Great Lakes Symp on VLSI. Piscataway, NJ: IEEE, 2003: 198-202

[9] Lee C, Lee J K, Hwang T, et al. Compiler optimization on VLIW instruction scheduling for low power [J]. ACM Trans on Design Automation of Electronic Systems, 2003, 8(2): 252-268

[10] Shao Z, Zhuge Q, Zhang Y, et al. Efficient scheduling for low-power high-performance DSP applications [C] //Proc of the 2nd Workshop on Hardware/Software Supper for High Performance Scientific and Engineering Computing. Piscataway, NJ: IEEE, 2003: 135-149

[11] Chabini N, Wolf M C. Reordering the assembly instructions in basic blocks to reduce switching activities on the instruction bus [J]. Computers & Digital Techniques, 2011, 5(5): 386-392

[12] Weng T H, Lin C H, Shann J J J, et al. Power reduction by register relabeling for crosstalk-toggling free instruction bus coding [C] //Proc of the 2010 Int Conf on Computer Symp. Piscataway, NJ: IEEE, 2010: 676-681

[13] Kuo W A, Chiang Y L, Hwang T T, et al. Performance-driven crosstalk elimination at post-compiler level [C] //Proc of IEEE Int Symp on Circuits and Systems. Piscataway, NJ: IEEE, 2006: 3041-3044

[14] Dhiman G, Ayoub R, Rosing T. PDRAM: A hybrid PRAM and DRAM main memory system [C] // Proc of the 46th Annual Design Automation Conf. Piscataway, NJ: IEEE, 2009: 664-669

[15] Ferreira A P, Zhou M, Bock S, et al. Increasing pcm main memory lifetime [C] // Proc of Design, Automation and Test in Europe. Piscataway, NJ: IEEE, 2010: 914-919

[16] Liu D, Wang T, Wang Y, et al. PCM-FTL: A write-activity-aware NAND flash memory management scheme for PCM-based embedded systems [C] // Proc of the 32nd Real-Time Systems Symp. Piscataway, NJ: IEEE, 2011: 357-366

[17] Qureshi M K, Karidis J, Franceschini M, et al. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling [C] // Proc of the 42nd Annual Int Symp on Microarchitecture (MICRO). Los Alamitos, CA: IEEE Computer Society, 2009: 14-23

[18] Qureshi M K, Srinivasan V, Rivers J A. Scalable high performance main memory system using phase-change memory technology [C] // Proc of the 36th Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2009: 24-33

[19] Xu W, Liu J, Zhang T. Data manipulation techniques to reduce phase change memory write energy [C] // Proc of the Int Symp on Low Power Electronics and Design (ISLPED). Piscataway, NJ: IEEE, 2009: 237-242

[20] Zhou X, Yu C, Petrov P. Compiler-driven register re-assignment for register file power-density and temperature reduction [C]// Proc of the 45th Annual Design Automation Conf. New York: ACM, 2008: 750-753

[21] Yang C, Orailoglu A. Processor reliability enhancement through compiler-directed register file peak temperature reduction [C] // Proc of IEEE/IFIP Int Conf on Dependable Systems & Networks. New York: ACM, 2009: 468-477

[22] Liu T, Orailoglu A, Xue C J, et al. Register allocation for simultaneous reduction of energy and peak temperature on registers [C]// Proc of Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2011: 1-6

[23] Aho A V. Compilers: Principles, Techniques, & Tools [M]. Reading, MA: Addison Wesley, 2007.

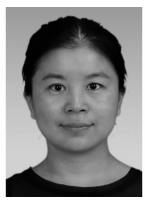
[24] Burger D, Austin T M. The SimpleScalar tool set, version 2.0 [J]. SIGARCH Computer Architecture News, 1997, 25 (3): 13-25

He Yanxiang, born in 1952. PhD, professor, PhD supervisor. Senior member of China Computer Federation. His main research interests include compiler system, trusted software, and software engineering.

Yu Tao, born in 1981. PhD candidate. His main research interests include trusted software, embedded systems, and next genaration internet architechure (yut@whu.edu.cn).

Chen Yong, born in 1986. PhD. His main research interests include trusted software and optimization for the embedded system (cyong@whu.edu.cn).

Li Qingan, born in 1986. PhD and lecturer. His main research interests include trusted software and optimization for the embedded system (qali@whu.edu.cn).



Jiang Nan, born in 1976. PhD candidate. Her main research interests include trusted software and embedded system (nanjiang@whu.edu.cn).

Xu Chao, born in 1980. PhD candidate. His main research interests include trusted software and embedded system (xuch@whu.edu.cn).

Wen Weidong, born in 1976. PhD and associate professor. His main research interests include compiler system and distributed parallel processing.