软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(8):1761−1774 [doi: 10.3724/SP.J.1001.2013.04363]

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

面向高性能业务应用的基于剖视信息的系统能耗优化*

易会战, 罗兆成

(国防科学技术大学 计算机学院,湖南 长沙 410073)

通讯作者: 易会战, E-mail: huizhanyi@nudt.edu.cn, http://www.nudt.edu.cn

摘 要: 当前,很多部门使用高性能计算机周期性地进行业务性的数值计算.维护这些业务系统的主要代价是每天消耗的大量电能,降低能量消耗能够极大地降低维护业务系统的成本.高性能业务系统的核心是微处理器,当前,微处理器普遍支持动态电压调节技术.该技术通过降低微处理器的电压和频率减小微处理器的能耗,但是一般会导致系统性能的下降.提出了一种面向高性能业务应用的能量优化技术.该技术利用系统支持的多个频率层次,建立性能约束下的能量优化模型,优化业务应用的能耗.根据程序信息获取方式的差别,提出了 SEOM 和 CEOM 两种能量优化模型,SEOM 模型的程序信息可以直接测试获取,CEOM 的程序信息采用编译器插桩方法获取.使用典型平台对能耗优化效果进行了验证,最多可节省 12%的能耗.

关键词: 业务系统;动态电压频率调节;剖视信息;能量模型

中图法分类号: TP314 **文献标识码**: A

中文引用格式: 易会战,罗兆成.面向高性能业务应用的基于剖视信息的系统能耗优化.软件学报,2013,24(8):1761−1774. http://www.jos.org.cn/1000-9825/4363.htm

英文引用格式: Yi HZ, Luo ZC. Profile-Guided optimization of system energy consumption for high-performance operational applications. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1761−1774 (in Chinese). http://www.jos.org.cn/1000-9825/4363.htm

Profile-Guided Optimization of System Energy Consumption for High-Performance Operational Applications

YI Hui-Zhan, LUO Zhao-Cheng

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

Corresponding author: YI Hui-Zhan, E-mail: huizhanyi@nudt.edu.cn, http://www.nudt.edu.cn

Abstract: Currently many high-performance computers are used to finish operational numerical computing cyclically. The main maintenance cost originates from the large amount of electric energy, and reducing energy consumption can reduce the maintenance cost significantly. The core units for operational systems are microprocessors, and the current microprocessors prevalently support the low power technique of the dynamic voltage and frequency scaling (DVFS). DVFS reduces the energy consumption by decreasing the supply voltage and execution frequency, which generally leads to performance reduction. This paper models energy consumption of operational applications confined by time constraints, and present energy optimization techniques by DVFS in the operational systems. Differing on the way to obtain the program execution information, two energy optimization models, SEOM and CEOM are evolved. The execution information of SEOM is obtained directly from testing, and the execution information of CEOM is obtained from compiler-directed program profile. The models have been investigated in representative computer platforms, and the results show that they can save 12% the largest reduction of energy consumption.

Key words: operational system; dynamic voltage and frequency scaling; profile information; energy model

* 基金项目: 国家自然科学基金(60903059, 61003087, 61170049); 国家高技术研究发展计划(863)(2012AA01A301, 2012AA010903); 国家科技重大专项(2009ZX01036-001-003-001)

收稿时间: 2012-05-28; 修改时间: 2012-09-29; 2012-11-06; 定稿时间: 2012-12-27

随着高性能计算机系统的规模越来越庞大,计算密度越来越高,系统的功耗随之呈指数趋势飞速增长.在TOP500高性能计算机系统中排名第一的K computer系统的功耗为9.89MW,第二名Tianhe-1A系统的功耗为4.04MW,维护这样规模庞大的系统需要高额的费用.高温环境下运行增加了芯片的失效率,温度每升高10°C,系统失效率将提高1倍[1],这将导致计算机系统的稳定性下降.目前,高性能计算机系统的能耗已经成为制约大型计算机系统发展的主要瓶颈之一[2].

当前,很多部门使用高性能计算机系统周期性地进行业务性的数值计算,最典型的例子是气象部门安装了业务主机,每天使用数值气象模式进行数值天气预报.维护这些业务系统的主要代价是每天消耗大量电能,降低能量消耗能够极大地降低维护业务系统的成本.但是,业务系统的两个特征加剧了系统的能耗问题:一方面,为了保证业务系统的服务质量,业务主机一般都预留了部分计算能力,保证业务应用能够按时完成,同时保证满足可预见的将来的计算需求,这无疑将导致能耗的增加;另一方面,高性能计算系统由于规模庞大、维护复杂,为了保证系统部件可靠运行,业务应用在两次计算之间的空闲时间,系统一般不会采用关闭电源的方式节电,而是处于空转状态,业务系统在空转状态下的能耗也相当可观,是当前硬件技术尚不能解决的问题.

高性能业务系统的核心是微处理器,当前,微处理器普遍支持动态电压调节(dynamic voltage and frequency scaling,简称DVFS)技术.该技术通过降低微处理器的电压和频率减少微处理器的能耗,但是一般会导致程序性能下降.以前,DVFS技术大多应用于嵌入式领域,主要针对嵌入式CPU和存储器的功耗优化.嵌入式领域的一个重要特点是,CPU、存储器这些部件往往是嵌入式系统的主要能耗源,因此,DVFS这样的低功耗方法可以有效发挥作用.

相比之下,在高性能计算领域,虽然微处理器的能耗在系统整体能耗中占有相当大的比例,但其他部件的能耗仍然对系统整体能耗具有重要影响.DVFS技术以延长执行时间为代价来达到减少处理器能耗的目的,但时间延长将导致系统其他部件的能耗上升,此时带来的后果可能是程序运行期间系统整体运行能耗的上升.本文的工作基于高性能业务应用的实际运行情况这一背景:首先,许多业务系统预留了部分计算能力,满足业务运行的截止时间远大于业务应用的实际运行时间;其次,每次在业务应用完成后系统不会关闭电源,而是转入空转状态.本文研究在保证业务应用能够满足截止时间限制的前提下,采用DVFS技术是否能够节省高性能系统的能耗.因为空转状态的存在,必须考虑业务应用完成后系统仍然在产生空转能耗,因此在进行DVFS技术节能效果分析时,考虑相等运行时间内系统的能耗变化情况是更加公平的尺度.本文考虑系统空转能耗的影响,利用系统支持的多个频率层次建立性能约束下的能耗优化模型,优化业务系统的能耗.根据程序信息获取方式的差别,本文提出了SEOM(simplified energy optimization model)和CEOM(complex energy optimization model)两种能耗优化模型,SEOM模型的程序信息可以直接测试获取,CEOM的程序信息采用编译器插桩方法获取.然后,根据测试数据和系统能耗优化模型对能耗优化方法进行了实验验证.实验结果显示,对于不同的时间约束条件和程序,最多可节省12%的能耗.

功耗管理是当前研究的热点.操作系统的功耗管理[3]和任务调度[4,5]是重要的研究方向,编译技术广泛用于功耗优化[6].大型数据中心和Web服务器产生了大量能量消耗,因此也出现了面向相关系统的优化工作[7−9].科学计算具有非交互式、分布式的特征,这与非交互式的受限于内存的单处理器应用不同,也不同于分布式、交互式Web应用的特征,因此应该采用不同的功耗管理技术[10−13].

本文第1节阐述能耗优化模型.第2节基于能耗优化模型介绍编译器实现框架.第3节是实验设置.第4节对实验结果进行分析.第5节给出结论和未来的工作.

1 能耗优化模型

以CMOS电路为基础的处理器功耗与电压和频率有紧密的关系,DVFS通过在系统运行期间动态调整处理器的电压和频率,减少系统能耗.在当前典型计算机系统下对大量程序的测试表明:计算机程序的性能受处理器运行频率的影响,其能耗也与处理器运行频率紧密有关;某些程序在系统频率降低时,运行时间延长,运行期间的能耗减少;而其他程序运行时间延长,序运行期间的能耗增加.

对于高性能业务系统,业务应用运行期间的能耗并不能代表系统的整体能耗,需要考虑其运行特点,必须考虑空转能耗的影响.尽管 CPU、内存等部件使用了一些先进的节能技术,但是系统空转能耗仍然是相当可观的.除了 CPU 的功耗之外,系统功耗的很大一部分消耗在电源、存储、磁盘等部件上.对于很多系统来说,因为缺乏能量管理能力或者某些情况下不能实施能量管理,系统空转情况下的能耗是不可忽略的,甚至可能占系统能耗的很大比例.例如,考虑一个典型的 Linux/x86 平台的系统,系统空转功耗为 94W,系统满负载运行情况下功耗为 138W,空转功耗占总功耗的 68%以上.

当对不同频率下系统运行能耗进行比较时,业务应用采用低频率运行导致的延长时间段内对于高频率运行时的系统则是保持在空转状态,仍然需要消耗能量,这部分能耗必须在比较时考虑进来.也就是说,对于高性能业务系统,用业务应用运行期间的能耗来衡量 DVFS 技术的节能效果并不合适.

如图 1 所示,业务应用在高频率 f_1 下的能耗为 E_1,运行时间为 T_1,运行平均功率为 P_1;在较低频率 f_2 下的能耗为 E_2,运行时间为 T_2,运行平均功率为 P_2.T_1 到 T_2 期间,系统空转时的功率为 P_{idle}.我们在进行能耗对比时,图中阴影部分的 E_{idle} 也要作为系统在高频率 f_1 下运行的能耗考虑进来.也就是说,应该比较 E_2 和(E_1+E_{idle})的大小.

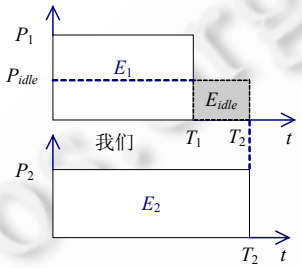

Fig.1 System energy consumptions in different frequency

图 1 不同频率下系统运行能耗

1.1 考虑空转能耗的动态电压调节技术的能耗优化模型

我们将一个业务应用程序理解为由一系列程序区域组合而成,如果知道了每个区域在不同频率的运行时间以及每个区域在不同频率下运行的能耗,设置其中的一个区域在低频率运行,其余区域保持在高频率运行,就可以计算出程序的运行时间和能耗.下面给出问题的形式化描述.

假定某频率可调节系统可以运行在频率 f 和最高频率 $f_{\max}$.在其上运行应用程序 PR,其中的区域 RE 为进行频率调节的程序区域,$N(RE)$为区域 RE 执行的次数,以 $T(RE,f)$表示在频率 f 下区域 RE 运行的时间,以 $E(RE,f)$表示在频率 f 下区域 RE 运行的能耗.系统进行一次动态电压调节的能耗开销为 E_{trans},时间开销为 T_{trans}.

进行频率调节后,程序运行时间为

$$T_{DVFS}=T(RE,f)+T(PR-RE,f_{\max})+T_{trans}\cdot 2\cdot N(RE) \tag{1}$$

即 T_{DVFS} 为在频率 f 下运行区域 RE 的时间加上剩余区域在最高频率运行时间,再加上进行 DVFS 的时间开销.这里使用 PR 代表整个程序区域,因此,$T(PR-RE,f_{\max})$表示整个程序区域减去区域 RE 后的区域的运行时间,后面公式中的 PR 含义相同.

进行频率调节后,程序运行能耗为

$$E_{DVFS}=E(RE,f)+E(PR-RE,f_{\max})+E_{trans}\cdot 2\cdot N(RE) \tag{2}$$

即 E_{DVFS} 为在频率 f 下运行区域 RE 的能耗加上剩余区域在最高频率运行的能耗,再加上进行 DVFS 的能耗开销.

为了保证进行频率调节后的程序性能仍然能够满足业务程序的时间要求,对性能损失进行限定.该限定为对时间的约束条件.假定允许最大性能降低幅度比率为 r,称 r 为时间松弛系数,则该性能约束条件描述为

$$T_{DVFS}\leqslant(1+r)\cdot T(PR,f_{\max}) \tag{3}$$

即频率调节后,程序运行时间 T_{DVFS} 不能超过整个程序在最高频率 $f_{\max}$ 下运行时间的(1+r)倍.

对于业务系统,可以设置业务应用的运行周期时间为 T_{cycle},于是可以恰当设置 r,令

$$T_{cycle}\equiv(1+r)\cdot T(PR,f_{\max}) \tag{4}$$

如前所述,对于高性能业务计算机系统的能耗优化,不能只考虑业务程序的运行能耗,而是要考虑业务周期内系统的整体能耗.因此,能耗优化时必须要考虑 T_{cycle} 内程序运行完成后的系统空转能耗.设定时间约束条件 $(1+r)\cdot T(PR,f_{\max})$,则其间的系统能耗如公式(5)所示,其中,假定系统的空转功耗为 P_{idle}.

$$E=E(RE,f)+E(PR-RE,f_{\max})+E_{trans}\cdot 2\cdot N(RE)+P_{idle}\cdot[(1+r)\cdot T(PR,f_{\max})-T_{DVFS}] \tag{5}$$

结合公式(1)、公式(3)和公式(5),得出最小化问题的形式化描述如公式(6)所示:

$$\begin{cases}\min_{RE,f} E=E(RE,f)+E(PR-RE,f_{\max})+E_{trans}\cdot 2\cdot N(RE)+P_{idle}\cdot[(1+r)\cdot T(PR,f_{\max})-T_{DVFS}]\\ \quad 受限于:T_{DVFS}\leqslant(1+r)\cdot T(PR,f_{\max})\\ \quad 其中,T_{DVFS}=T(RE,f)+T(PR-RE,f_{\max})+T_{trans}\cdot 2\cdot N(RE)\end{cases} \tag{6}$$

通常,程序运行期间的系统功耗会根据负载情况发生变化,是程序时间和频率的函数,即功耗为 $P(RE(t),f)$,而相应的能耗为

$$E(RE,f)=\int P(RE(t),f)\mathrm{d}t \tag{7}$$

结合公式(6)和公式(7),最小化问题描述为

$$\begin{cases}\min_{RE,f} E=\int P(RE(t),f)\mathrm{d}t+\int P((PR-RE)(t),f_{\max})\mathrm{d}t+E_{trans}\cdot 2\cdot N(RE)+P_{idle}\cdot[(1+r)\cdot T(PR,f_{\max})-T_{DVFS}]\\ \quad 受限于:T_{DVFS}\leqslant(1+r)\cdot T(PR,f_{\max})\\ \quad 其中,T_{DVFS}=T(RE,f)+T(PR-RE,f_{\max})+T_{trans}\cdot 2\cdot N(RE)\end{cases} \tag{8}$$

1.2 动态电压调节技术的简化能耗优化模型SEOM

最小问题的目标是找到 E 最小时其对应的区域 RE 和频率 f.一般来说,程序区域的选择方法多种多样,实际上很难穷举所有情况.考虑一种特殊区域设置的情况,即考虑整个程序作为一个区域进行降频的情况,此时,DVFS 的开销可以忽略,可以通过公式(6)的最小化问题得到简化的能耗优化模型 SEOM:

$$\begin{cases}\min_{f} E=E(PR,f)+P_{idle}\cdot[(1+r)\cdot T(PR,f_{\max})-T(PR,f)]\\ \quad 受限于:T(PR,f)\leqslant(1+r)\cdot T(PR,f_{\max})\end{cases} \tag{9}$$

SEOM 的目标就是找到一个频率,在该频率下,程序运行时间不超过约束条件时间 $(1+r)\cdot T(PR,f_{\max})$,同时,在该频率下,系统运行能耗 E 与其他频率下的运行能耗相比最小.

1.3 动态电压调节技术的复杂能耗优化模型CEOM

SEOM 使用简单,但是因为只能设置单个区域,而系统只能在有限的不连续频率上运行,可能在时间约束情况下无法有效采用低频运行降低能耗.下面考虑对公式(8)的最小化问题进行简化,扩展可供选择降频的区域数目.根据公式(8),要对程序区域进行分析,需要获得程序中每个区域的运行能耗,这需要测量和记录区域运行时刻的瞬时功耗,测量的代价很大.对典型串行程序运行期间的功耗数据进行采样分析,发现在程序运行过程中,其瞬时功耗变化较小;同时,测量程序区域的运行时间相对简单.根据上述特点,考虑用不同频率下程序运行的平均功耗与区域运行时间的乘积来代替区域运行能耗.假定 $P(PR,f)$ 是程序 PR 在频率 f 下运行的系统平均功耗,因此通过公式(8)的最小化问题得到复杂能耗优化模型 CEOM:

$$\begin{cases}\min_{RE,f} E=P(PR,f)\cdot T(RE,f)+P(PR,f_{\max})\cdot T(PR-RE,f_{\max})+\\ \qquad E_{trans}\cdot 2\cdot N(RE)+P_{idle}\cdot[(1+r)\cdot T(PR,f_{\max})-T_{DVFS}]\\ \quad 受限于:T_{DVFS}\leqslant(1+r)\cdot T(PR,f_{\max})\\ \quad 其中,T_{DVFS}=T(RE,f)+T(PR-RE,f_{\max})+T_{trans}\cdot 2\cdot N(RE)\end{cases} \tag{10}$$

1.4 实现方法

SEOM 求解需要测量的参数较少,包括不同频率 f 下程序 PR 的运行时间 $T(PR,f)$、不同频率 f 下程序 PR

的系统能耗 $E(PR,f)$和系统空转功耗 P_{idle}.获得这些参数后,依据设定的 r 值,使用公式(9)找出运行时间在$(1+r)\cdot T(PR,f_{\max})$内、系统能耗最小时所对应的频率.SEOM 情况下时间和能耗的测量都很简单,运行时间 $T(PR,f)$的测量可直接通过程序不同频率下运行获得,系统能耗 $E(PR,f)$的测量可以在时间测量时同时完成,本文采用功率计测试计算机应用程序的能耗,系统空转功耗 P_{idle} 也可以直接测量获取.

CEOM 求解需要获取更多的参数,包括各程序区域 RE 不同频率 f 下的运行时间 $T(RE,f)$、各程序区域 RE 的运行次数 $N(RE)$、程序在不同频率下运行的平均功率 $P(PR,f)$、DVFS 切换的时间开销 T_{trans} 和能耗开销 E_{trans}、系统空转功耗 P_{idle}.程序区域的相关参数测量依赖于区域的选择方法,我们将函数和外层循环作为候选区域,采用程序插桩的方法获取各程序区域 RE 不同频率 f 下的运行时间 $T(RE,f)$和各程序区域 RE 的运行次数 $N(RE)$.通过在 GCC 的 profile 遍增加对区域的运行时间 $T(RE,f)$和运行次数 $N(RE)$的测试函数,我们实现了对串行程序的插桩测试,具体实现方法将在第 2 节详述.DVFS 的过程是分级进行的,相邻两级的频率调节时间越短,频率跨度越大,调节时间越长.本文采用最低到峰值频率切换时间来代替各频率之间相互切换的时间 T_{trans}.与切换时间相比,DVFS 的切换能耗 E_{trans} 对整个程序的能耗影响并不明显,实现时一般将其忽略.程序在不同频率下运行的平均功率 $P(PR,f)$和系统空转功耗 P_{idle} 可以直接测量获取.

2 基于 GCC 的能耗优化模型 CEOM 实现

我们基于 GCC 编译器的插桩框架实现了能耗优化模型 CEOM,如图 2 所示.在第 1 次和第 2 次插桩编译时,将测量代码插入程序中以测量程序区域的时间和能耗参数,通过模型分析计算找出适合进行 DVFS 操作的区域;在第 3 次插桩编译时,将 DVFS 指令插入适合进行 DVFS 操作的区域.各步骤之间通过*.gcda 格式文件进行信息传递.T_{trans} 和 P_{idle} 为通用参数,每个系统只需要测试 1 次.

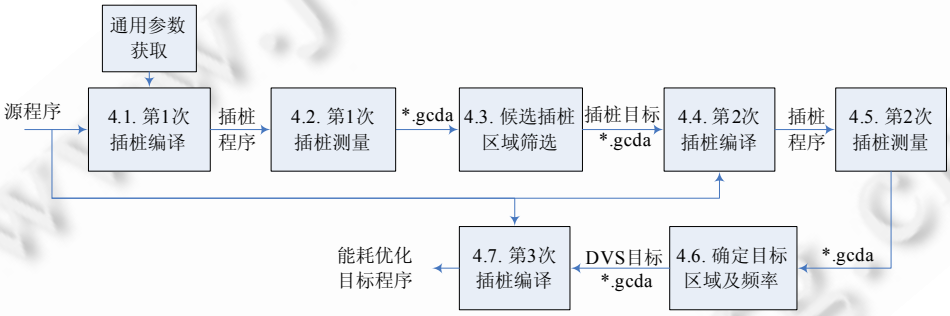

Fig.2 Flow chart of the low-power compilation algorithm

图 2 低功耗编译算法流程

2.1 第1次插桩编译

第 1 次插桩编译的目标是实现对所有候选程序区域的运行时间和运行次数的测量.编译器插桩过程中需要考虑将什么样的代码区域作为候选插桩区域,我们考虑的候选区域是函数结构和最外层循环结构,内层循环不作为候选区域,是为了避免插桩的次数太多而带来巨大的插桩开销.这两类结构是程序的典型结构,其运行时间占了整个程序运行时间的绝大部分,并且这两类结构在编译过程中构建区域比较简单.

另外,候选区域的大小(运行时间长短)也需要仔细考虑,即什么规模的函数结构和循环结构能够作为候选区域.若区域运行时间太长,则因为可调节频率限制和约束条件限制,降频这样的区域可能导致无法在截止时间完成计算,不能作为优化问题的解区域,导致满足性能限制要求的区域太少;若区域运行时间太短,则一方面不能通过 DVFS 获得能量节省,另一方面插桩测量带来的额外开销又太大.因此,第 1 次插桩编译采用语句阈值来控制区域大小,即估算函数或循环的语句数,将其与语句阈值进行比较,大于者列为候选区域,否则不列为候选区域.具体的插桩过程实现如下:

首先,标记或构造出每个候选区域需要插桩的边.测量区域的执行时间和执行次数,需要在区域的开始位置和结束位置插入测量函数.通过分析在 GCC 编译器中间阶段生成的控制流图,找到区域的入口边和出口边将其作为插桩测量的开始边和结束边,采用边链表的方式存储这些边的集合,测量函数将插桩在这些边的集合上.因为将最外层循环结构和函数作为候选区域,因此区域的入口边只有 1 个,而区域的出口可能有多个.对于循环结构,如图 3 所示,循环结构的 PERHEADER 基本块只有 1 个后继块 *B*1,则插桩开始边即是 PERHEADER 指向基本块 *B*1 的边.而插桩结束边对应循环的所有出口边,即 *B*4 指向 EXIT 的边以及 *B*2 指向 EXIT 的边.对于函数结构,则需要进行基本块的切分操作以构造出需要插桩的边.如图 4(a)所示为某函数的初始控制流图结构,当 ENTRY BLOCK 的后继只有 1 个时,则将其后继边作为插桩开始边,即 ENTRY BLOCK 指向基本块 *B*1 的边为插桩开始边.为了找到插桩结束边,需要将 EXIT BLOCK 的前趋边指向的所有基本块中包含的 return 语句切分出来,构造出新的出口边,如图 4(b)所示.

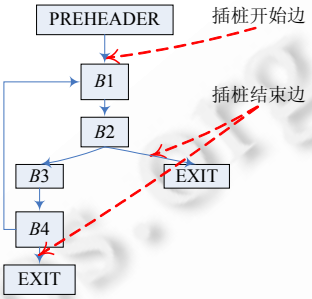

Fig.3 Instrumentation of loop structures

图 3 循环结构插桩

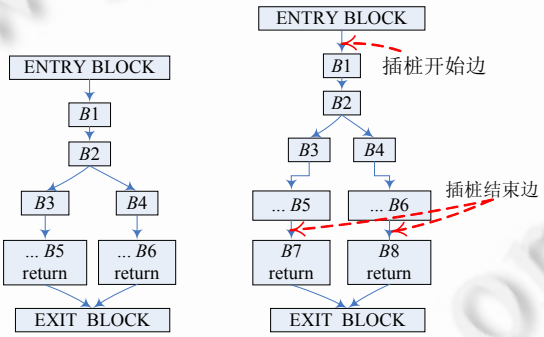

Fig.4 Instrumentation of function structures

图 4 函数结构插桩

其次,判断区域所包含的语句数是否满足语句阈值的要求:若满足,则将测量函数插入边链表中记录的插桩开始边和结束边;否则,忽略该区域,即不对其进行插桩操作.对程序中的所有循环结构和函数进行插桩会导致巨大的时间开销,采用 GCC 提供的代价函数评估函数和循环的执行时间,控制进行插桩的粒度.GCC 提供的循环和函数的代价函数能够以语句数为单位估计区域的执行时间,这种估计只能给出十分粗略的估计结果,因为其既没有考虑体系结构对区域运行时间的影响,同时没有考虑循环结构运行次数对区域运行时间的影响,因此,这里的语句阈值仅仅用于插桩过程中对候选区域的粗略控制.我们设置了一个最小代价阈值 *MIN_COST*,当区域语句数目大于 *MIN_COST* 时,才考虑作为候选插桩区域,否则不会作为候选区域.为了保证具有足够多的区域提供给能耗优化模型进行求解,*MIN_COST* 不能设置得过大(过大就没有满足要求的区域供选择);同时,又不能

引入太多的插桩开销,*MIN_COST* 也不能设置得太小,后面实验部分分析了 *MIN_COST* 不同取值的结果.

```
1:  void func1()          19:    …
2:  {                     20:
3:  …                     21:    return
4:    return              22:  }
5:  }                     23:  int main()
6:  void func2()          24:  {
7:  {                     25:
8:                        26:    …
9:    …                   27:   func1();
10:                       28:
11:   for ( )             29:   for( )
12:     {…}               30:    {…}
13:                       31:
14:   …                   32:    func2();
15:                       33:    …
16:   for ( )             34:
17:     {…}               35:    return 1;
18:                       36:  }
```

(a) 初始伪代码

```
1:  void func1()          19:    …
2:  {                     20:    end_profiler(1);
3:  …                     21:    return
4:    return              22:  }
5:  }                     23:  int main()
6:  void func2()          24:  {
7:  {                     25:    start_profiler(4);
8:    start_profiler(1);  26:      …
9:    …                   27:    func1();
10:   start_profiler(2);  28:    start_profiler(5); for( )
11:   for( )              29:      {…}
12:     {…}               30:    end_profiler(5); func2();
13:   end_profiler(2);    31:    …
14:   ...                 32:    end_profiler(4);
15:   start_profiler(3);  33:    return 1;
16:   for( )              34:  }
17:     {…}               35:
18:   end_profiler(3);    36:
```

(b) 插桩后的代码

Fig.5 Initial pseudocode and the instrumented pseudocode

图 5 初始伪代码和插桩后的代码

图 5(a)所示是一段类 C 语言伪代码,包含了 2 个子函数和若干循环.其中,函数 func1 的语句数少于语句阈值 *MIN_COST*,其余函数和循环所含语句数均大于语句阈值.因此,需要进行插桩的区域有 5 个,包括函数 func2、函数 main、第 11 行、第 12 行对应的循环(以下用 loop1 表示)、第 16 行、第 17 行对应的循环(以下用 loop2 表示)以及第 29 行、第 30 行对应的循环(以下用 loop3 表示).对其进行插桩后,程序的伪代码如图 5(b)所示.图 5(b)代码中,start_profiler 是插桩测量开始函数,用来获取区域上一次运行时的计数器数据;end_profiler 是插桩测量结束函数,用来累计计数器数据.函数的参数是计数器编号,一个区域对应一个计数器变量,图 5 中程序共有 5 个计数器变量.

2.2 第1次插桩测量

将第 1 次插桩编译生成的程序在计算机的峰值频率下运行,此时植入的插桩代码会将区域的累计运行时钟周期数和运行次数测量出来,测量的结果保存在*.gcda 文件中.

插桩区域存在区域嵌套的情况.我们采用计数器栈解决区域嵌套时计数器变量的维护问题.例如图 5 所示程序中的函数 func2,该函数包括了 3 个插桩区域,除其本身外,该函数内的两个循环也同时进行插桩,3 个插桩区域发生重叠.

图 6 显示了进入函数 func2 后计数器栈的变化情况:

- 首先获取区域计数器 1 和当前时刻 P1_0,将其压入计数器栈中(由函数 start_profiler(1)完成).
- 随后进入 loop1 循环区域,获取区域计数器 2 的数据和当前时刻 P2_0,将其压入计数器栈中(由函数 start_profiler (2)完成),此时,动态栈中有两个计数器的相关数据,计数器 2 的数据在栈顶.
- 之后,loop1 循环区域运行完成,程序读取栈顶的计数器 2 的数据并计算完成得到该区域的执行周期和累计执行周期,同时将栈顶数据弹出(由函数 end_profiler(2)完成).
- 随后的 loop2 循环区域与之类似.
- 当函数 func2 运行到结尾时,读取栈顶的计数器 1 的数据并将其弹出(由函数 end_profiler(1)完成).

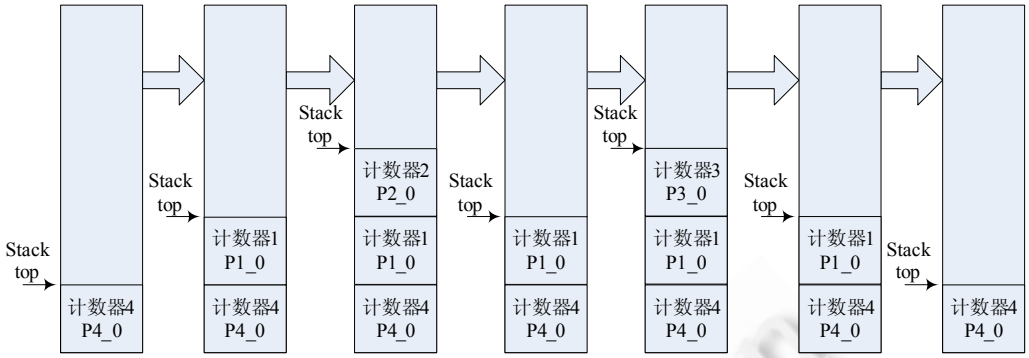

Fig.6 Variation of the counter stack when func2 are executed

图 6 func2 运行过程中计数器栈的变化过程

2.3 候选插桩区域筛选

因为第 1 次插桩采用的函数和循环的代价函数只能粗略地估计函数和循环的执行时间,因此,经过 *MIN_COST* 静态过滤后的区域个数仍然较多,插桩的开销也较大.这些区域并非都有进行降频调节的价值,因此,使用第 2.2 节所述步骤生成的数据文件,提取所有候选区域的时钟周期数和执行次数后对程序区域进行筛选.采用下面的条件剔除插桩区域:

$$T_{trans}\cdot 2\cdot N(RE)\leqslant \alpha\cdot T(RE,f_{\max}) \tag{11}$$

$$T_{trans}\cdot 2\cdot N(RE)\leqslant \beta\cdot T(PR,f_{\max}) \tag{12}$$

即进行 DVFS 所花费的总时间不能超过调节区域运行时间的α倍或程序总运行时间的β倍,不满足公式(11)或者公式(12)的区域被剔除.我们设置α=5%,β=1%.α和β的取值是考虑到频率切换开销的影响而设置的,一方面保证筛选后保留下足够进行能耗优化模型求解的区域集;另一方面,将调频明显影响整个程序性能的区域去除.

完成筛选后,修改*.gcda 文件,对不符合条件的程序区域做标记,标记方法是将该区域的运行次数设置为 0,表示不要将该区域作为候选调频区域.修改后的*.gcda 文件将在后续编译时被读入,用于指导后续插桩.

2.4 第2次插桩编译

第 2 次插桩编译的目的仍然是实现对程序区域的运行时间和运行次数的测量,但本次编译测量的对象是经过上一步骤筛选后剩余的候选区域.GCC 编译应用程序时,首先读取上一步骤生成的*.gcda 文件,然后在插入测量函数这一步骤时,对照该区域与测量数据文件中对应的区域运行次数,如果为零,则不进行插桩操作.重新编译后,程序中的插桩位置会明显减少,从而极大地降低了插桩对生成的目标程序性能的影响,获取的插桩测试数据将更加精确.

2.5 第2次插桩测量

在每个 CPU 支持的频率下运行一遍第 2 次插桩编译生成的程序,即可测量出各程序区域在每个频率下的运行周期数和运行次数.在每个频率下程序执行完成后,生成对应频率的*.gcda 文件,用于后续求解最小化问题.

第 2 次插桩测量的另一个作用是,在运行插桩程序的同时测量程序在不同频率下的平均功率参数 $T(PR,f)$.

2.6 确定可降频运行的区域与对应的频率

通过上述步骤,我们可以得到求解能耗优化模型 CEOM 所需的参数.分别将不同区域的数据参数代入优化模型,获得最小能耗对应的区域、频率以及最佳调频次数.

采用插桩方法获得的区域的运行时间是被测区域多次运行累计的运行时间,区域的运行时间是作为一个整体考虑的.也就是说,假设求解出的区域 *RE* 共运行了 1 000 次,则最终的调频次数也是 1 000 次,即每运行到该区域就进行降频和恢复操作.某些情况下,如图 7(b)所示,假设区域 *RE* 的降频节能效果最佳,但其运行 1 000 次时

增加的时间超过时间松弛比例$(1+r)\cdot T(PR,f_{\max})$的要求,此时就不得不选择其他区域,甚至可能找不出适合降频的区域.为了扩大最小化问题的候选区域,我们将区域 *RE* 的每次运行都单独进行考虑.这样,本来只是 1 个区域 *RE*,现在就可以理解为 1 000 个区域,也就是说,将 1 次,2 次,…,1 000 次分别作为候选区域考虑以获得最优解.例如,可以获得如下解:区域 *RE* 在频率 *f* 下运行 600 次时对应的时间松弛满足要求,同时其能耗最小.这样,在最终编译生成的低能耗程序中,如图 7(c)所示,区域 *RE* 只做 600 次降频运行,而剩余的 400 次仍然保持在峰值频率运行,满足了时间限制.因为编译器插桩只获得了区域多次运行的累计时间,这里实际上将区域代码每次运行的时间做近似相等的处理.

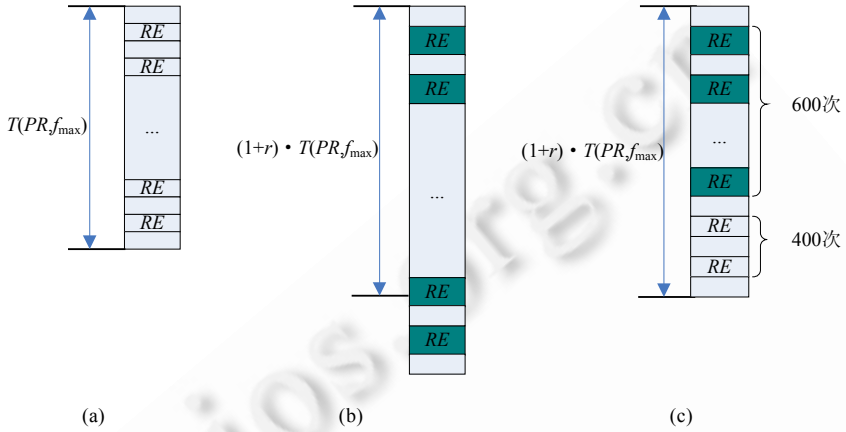

Fig.7 Used regions when solving the optimization problem

图 7 优化问题求解使用的区域

根据最优化问题获得的解,修改*.gcda 文件,对应解区域的执行周期修改为频率层次数,解区域的执行次数修改为区域降频运行的次数,其他区域的执行次数置 0.

2.7 第3次优化编译

第 3 次编译的目的是将频率调节代码插入能耗优化模型求解得到的区域中,此次插桩区域只有 1 个.首先,读取上一步修改后得出的*.gcda 文件,此时,*.gcda 文件中除了一个区域以外,其他区域的运行次数都为 0,找到该区域,在原来插桩开始边和结束边插入开始和结束频率调节函数,生成满足算法要求的低能耗程序.频率调节接口函数如下:

start_freq_adjuster(*freq_level*,*max_adjust_num*),

end_freq_adjuster(*freq_level*,*max_adjust_num*).

其两个参数的含义是:第 1 个 *freq_level* 代表目标频率层次,第 2 个 *max_adjust_num* 代表最大调频次数.我们降低能耗的方法是让指定的程序区域在低频率下运行,而其他区域仍然在峰值频率下运行.因此,start_freq_adjuster 函数的作用是将处理器频率降到指定的频率值,同时累计该区域的执行次数,当达到最佳调频次数后停止频率调节.而 end_freq_adjuster 函数的作用是在该区域执行完后恢复处理器的频率到峰值频率.

2.8 各步骤间的*.gcda数据文件变化

在上述的实现步骤中,3 次编译之间都通过 gcda 文件进行数据传递,运行第 1 次、第 2 次编译生成的程序,测量数据写入到 gcda 文件中,而第 2 次、第 3 次编译时又需要读入处理后的 gcda 文件.

下面以图 5 所示程序的 gcda 数据文件为例来说明算法实现过程中区域测量数据的变化及其作用.表 1 是第 2.2 节第 1 次插桩测量后生成的 gcda 数据文件的区域信息,第 1 列表示区域的标识,每个区域有一个独立的标识,第 2 列表示该区域的累计运行周期数(即运行时间),采用 PAPI 性能分析接口获得,第 3 列表示该区域的累

计运行次数,这里,单次插桩开销设置为 50 周期.表 2 是经过第 2.2 节候选区域筛选后的结果数据,由表中数据可以看出,只有标识为 1,3,4 的区域的运行次数保持不变,其余区域的运行次数都被设置为 0,这些区域将在第 2 次编译插桩时被剔除.

Table 1 Measurement data after the first instrumentation

表 1 第 1 次插桩测量数据

区域标识	运行周期数	运行次数
1	1 000	1
2	400	1
3	500	1
4	1 400	1
5	200	1

Table 2 Selected regions

表 2 筛选后的区域

区域标识	运行周期数	运行次数
1	1 000	1
2	400	0
3	500	1
4	1 400	1
5	200	0

表 3 是第 2.4 节第 2 次编译生成的程序在第 2.5 节不同频率下运行后测量得出的数据,假定系统的处理器支持 f_0 和 f_1 两个频率,由数据可以看出,只有标识为 1,3,4 的区域还有测量数据,其余区域的测量数据为 0,即不再对这些区域进行插桩测量,因此也不再存在插桩开销.表 3 的数据代入模型公式(10),根据第 2.6 节的方法即可求得最佳的调频区域、调频频率和调频次数.表 4 是根据求解结果修改的 gcda 文件内容,可知求解得出的调频区域为标识为 1 的区域,对应的频率层次值为 0,即在频率 f_0 下运行区域 1,调频次数为 1 次,其余区域的运行次数都被设置为 0.第 2.7 节根据表 4 的结果,在区域 1 的开始和结束位置插入调频代码.

Table 3 Measurement data after the second instrumentation

表 3 第 2 次插桩测量数据

区域标识	f_0 下插桩测量数据		f_1 下插桩测量数据	
	运行周期数	运行次数	运行周期数	运行次数
1	950	1	950	1
2	0	0	0	0
3	500	1	500	1
4	1 300	1	1 300	1
5	0	0	0	0

Table 4 Solution region

表 4 解区域

区域标识	运行周期数	运行次数
1	0	1
2	0	0
3	500	0
4	1 300	0
5	0	0

3 实验设置

采用功率计测试计算机应用程序的能耗,功率计的能耗测试精度为 1/10 瓦时.测试选择的目标计算机见表 5.

Table 5 System configuration of target computers

表 5 目标机系统配置

	T1			T2
CPU 型号	Intel i5 750 2667			Intel Core 2 Duo P8800
CPU 数	1			1
每个 CPU 核数	4			2
CPU 支持频率 (GHz)	2.668 2.667 2.533 2.400	2.267 2.133 2.000 1.867	1.733 1.600 1.467 1.333 1.200	2.667 2.666 2.133 1.600 0.800
主存(GB)	2			2
操作系统	Fedora 14 Linux			Fedora 14 Linux
空转功耗(W)	49.5			25.5
最大功耗(W)	<120			<60

目标机的处理器都是多核处理器,可以进行并行程序的测试.T1 为 Intel 商用服务器,空转功耗高.T2 为 Sony VGN-Z55F 笔记本,空转功耗低.所有目标机的 CPU 都支持 DVFS,操作系统加载了 cpufreq 模块支持动态电压调节.这里的目标计算机系统没有采用大规模的并行计算机系统,但是基于本文的优化模型,可以直接扩展到对大

规模业务主机的能耗测量和能耗优化.实验中使用了目标机支持的全部频率,T1 目标机处理器的可调节频率有13 个,T2 目标机支持 5 个频率层次.

实验使用测试程序集 SPEC2006.SPEC2006 是由 29 个测试程序构成的国际标准测试程序集.SPEC2006 可以运行在两种模式下.在标准运行模式下,程序每次运行一个副本,用于测试 CPU 的串行性能.SPEC2006 提供了3 种数据集:test,train,ref.正式测试都使用 ref 数据集.本文实验使用了 ref 数据集.

SEOM 可以简单地扩展到对并行程序的能耗优化,但是因为 CEOM 当前不支持并行程序,无法提供对比数据,我们没有给出 SEOM 并行程序的测试结果.

4 实验结果分析

本文第 2 节的 GCC 编译器支持 C 与 Fortran 语言,针对串行程序的性能和能耗进行优化,因此,我们从SPEC2006 测试程序集中选择了 C 与 Fortran 语言程序各 3 个,分别是 410.bwaves,429.mcf,433.milc,437.leslie3d,459.GemsFDTD 和 462.libquantum,对两种能耗优化模型 SEOM 和 CEOM 进行了实验分析.

4.1 插桩开销分析

插桩影响了程序的运行时间,进一步影响模型的优化结果,*MIN_COST* 参数的作用是:首先对候选区域进行一次粗略筛选,将明显不具备降频调节价值的区域剔除掉.表 6 是 433.milc 和 459.GemsFDTD 设定不同 *MIN_COST* 后得到的插桩个数以及插桩程序的运行时间对比.显然,随着 *MIN_COST* 的减小,插桩区域也不断增加.同时,编译后程序运行时间明显增加,插桩测量函数对程序性能的影响随之变大.此时,各区域测试的运行时间误差随之增加,越靠近外层的区域,插桩的误差就越大.后面的实验中,*MIN_COST* 取值 50.

Table 6 Instrumentation information in different *MIN_COST*

表 6 不同 *MIN_COST* 对应的插桩信息

MIN_COST	100		50		10	
	插桩数	运行时间(s)	插桩数	运行时间(s)	插桩数	运行时间(s)
433	125	574	210	1 385	405	17 397
459	124	805	178	887	400	928

第 1 次插桩测量后,会根据测试结果进行一次候选区域筛选,经过筛选后,插桩开销已经很小.表 7 是 T1 目标机上对程序插桩情况的统计.第 2 列和第 3 列是各程序在第 1 次编译后插桩的个数以及筛选后插桩的个数,筛选后的插桩区域明显减少.第 4 列是按照第 2.6 节的区域运行次数扩展后计算得出的区域数.第 5 列是以最高频率运行引入时间开销的比率,开销最大的 437.leslie3d 的插桩开销为 8.2%.候选区域筛选保证了合理的插桩开销.

Table 7 Number of instrumented regions

表 7 插桩区域个数

程序	初始插桩数	筛选后插桩数	扩展后区域数	插桩开销比率(%)
410.bwaves	17	14	5 218	0.8
429.mcf	42	19	66	0.1
433.milc	210	62	3 052	0.6
437.leslie3d	178	17	53 030	8.2
459.GemsFDTD	178	29	20 043	2.8
462.libquantum	72	29	15 720	2.4

4.2 T1目标机

为了分析 CEOM 模型的优化效果,我们设定时间松弛系数 r 从 0.05~0.5 取值.根据求解结果生成了对应不同 r 值的能耗优化程序,测量各程序的运行时间和能耗,计算运行时间比 r_T 和运行能耗比 r_E,计算方法如下:

$$r_T=T_{\mathrm{DVFS}}/T(PR,f_{\max}) \tag{13}$$

其中,T_{DVFS} 表示能耗优化程序的运行时间,$T(PR,f_{\max})$表示原始程序在峰值频率下的运行时间.

$$r_E = \frac{E_{\mathrm{DVFS}} + P_{idle} \cdot [1.5 \cdot T(PR, f_{\max}) - T_{\mathrm{DVFS}}]}{E(PR, f_{\max}) + P_{idle} \cdot 0.50 \cdot T(PR, f_{\max})} \tag{14}$$

其中,E_{DVFS} 表示能耗优化程序的运行能耗,$E(PR,f_{\max})$表示原始程序在峰值频率下的程序运行能耗,P_{idle} 表示系统的空转功率,所有能耗数据相对于 r=0.50 情况下的能耗进行了归一化.实验分析结果见表 8.部分程序运行时间超出 r 值范围的主要原因是区域测量误差和频率切换次数较多.使用 CEOM 模型,程序 429.mcf 在 T1 目标机上最多可节省 12%的能耗.

Table 8 Time and energy consumption of different r in the target computer T1

表 8 T1 目标机在不同 r 值下的时间和能耗

程序	r=0.05		r=0.10		r=0.20		r=0.30		r=0.40		r=0.50	
	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E
410	1.04	0.99	1.10	0.96	1.19	0.91	1.30	0.90	1.30	0.90	1.30	0.90
429	1.02	0.95	1.10	0.92	1.20	0.88	1.30	0.88	1.35	0.88	1.35	0.88
433	1.06	0.95	1.07	0.92	1.16	0.91	1.27	0.90	1.38	0.89	1.38	0.89
437	1.05	0.97	1.09	0.96	1.16	0.90	1.23	0.89	1.36	0.88	1.42	0.88
459	1.07	0.99	1.13	0.99	1.14	0.92	1.14	0.92	1.32	0.91	1.38	0.91
462	1.01	0.98	1.05	0.96	1.14	0.91	1.24	0.90	1.34	0.90	1.48	0.90

与 SEOM 相比,CEOM 有更多的候选可调频区域,可以更好地对程序性能进行控制.理论上,CEOM 可以根据任意的 r 值对程序运行时间进行调节以获得能耗优化程序.图 8 是 T1 目标机 SEOM 和 CEOM 节能效果对比,从对比数据可以看出,采用 CEOM 时,大部分程序在 r 值大于 0.20 以后也基本上是全程序段降频运行,故两种模型节能效果十分接近.对于 r 值小于 0.20 的情况,大部分程序 CEOM 的节能效果要好于 SEOM,这是由于 CEOM 控制粒度更细,能够找到满足要求的降频区域.

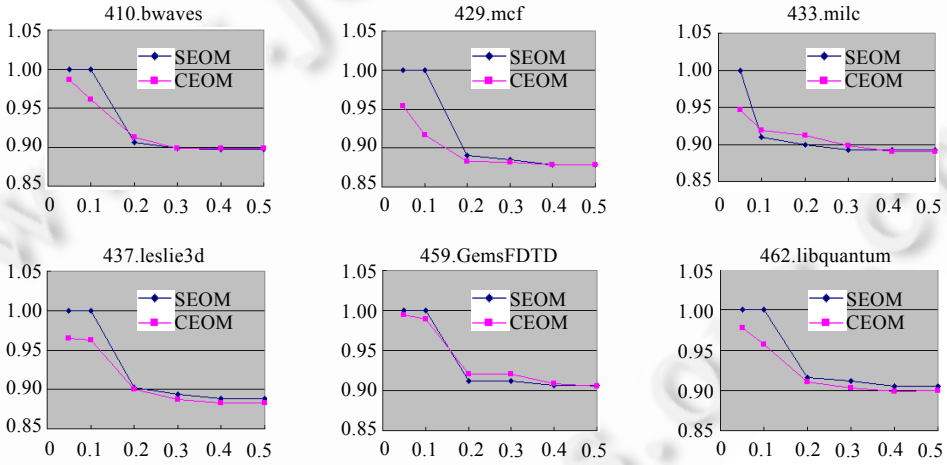

Fig.8 Energy consumption of SEOM and CEOM in the target computer T1

图 8 T1 目标机上 SEOM 和 CEOM 的能耗

4.3 T2目标机

使用 CEOM 模型,T2 目标机程序运行时间比和系统运行能耗比结果见表 9.在 T2 目标机上,CEOM 最多可以节省 12%的能耗.

图 9 对比了 T2 目标机上 SEOM 和 CEOM 的节能效果.在 T2 系统上,CEOM 和 SEOM 的节能效果相当,没有显示出 CEOM 在较小 r 值时的优势.从 T1 和 T2 平台的优化结果可以看出,CEOM 一般要优于 SEOM,但实际优化效果与目标平台相关.

Table 9 Time and energy consumption of different r in the target computer T2

表 9 T2 目标机在不同 r 值下的时间和能量消耗

程序	r=0.05		r=0.10		r=0.20		r=0.30		r=0.40		r=0.50	
	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E	r_T	r_E
410	1.03	0.97	1.08	0.96	1.19	0.94	1.22	0.92	1.22	0.92	1.22	0.92
429	1.06	0.97	1.14	0.91	1.14	0.91	1.14	0.91	1.34	0.88	1.34	0.88
433	1.09	0.93	1.09	0.93	1.24	0.92	1.24	0.92	1.24	0.92	1.24	0.92
437	1.03	0.97	1.03	0.97	1.18	0.92	1.18	0.92	1.18	0.92	1.45	0.88
459	1.03	0.99	1.03	0.99	1.17	0.93	1.17	0.93	1.17	0.93	1.46	0.91
462	1.07	0.92	1.07	0.92	1.20	0.92	1.20	0.92	1.20	0.92	1.20	0.92

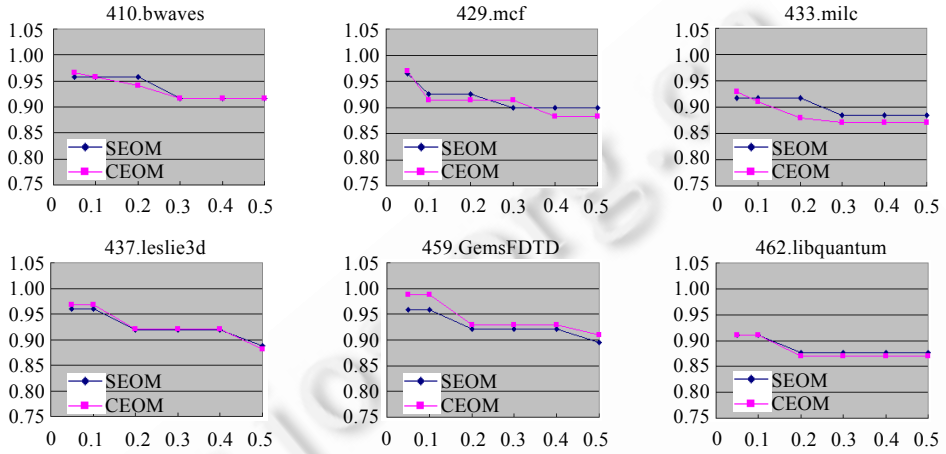

Fig.9 Energy consumption of SEOM and CEOM in the target computer T2

图 9 T2 目标机上 SEOM 和 CEOM 的能耗

5 结论和进一步的工作

本文建立了一个适用于高性能业务应用的考虑系统空转功耗影响的能耗优化模型,在时间约束条件下优化系统能耗.该方法将应用程序分成若干区域,通过程序部分区域降频运行来达到在节省能耗的同时又满足程序性能的要求.根据区域选择方法的不同,本文给出了 SEOM 和 CEOM 能耗优化模型,基于 GCC 编译器插桩实现了 CEOM 模型,在典型计算机系统上进行了测试,对比了 SEOM 和 CEOM 的能耗优化结果.实验结果显示,根据不同的时间约束条件和测试程序,本文所述方法最多可节省 12%的能耗.

SEOM 方法的测试结果表明,OpenMP 并行程序能够更好地节省能耗.本文下一步工作将针对 OpenMP 程序实现 CEOM 的优化方法,进一步减少能量消耗.

References:

[1] Feng W. Making a case for efficient supercomputing. ACM Queue, 2003,1(7):54–64. [doi: 10.1145/957717.957772]

[2] Ge R, Feng X, Cameron K. Improvement of power-performance efficiency for high-end computing. In: Siegel HJ, ed. Proc. of the 19th IEEE Int'l Parallel and Distributed Processing Symp. Washington: IEEE Computer Society, 2005. 233–234. [doi: 10.1109/IPDPS.2005.251]

[3] Marinoni M, Elastic BG. DVFS management in processors with discrete voltage/frequency modes. IEEE Trans. on Industrial Informatics, 2007,3(1):51–56. [doi: 10.1109/TII.2006.890494]

[4] Banikazemi M, Poff D, Abali B. PAM: A novel performance/power aware meta-scheduler for multi-core systems. In: Teller PJ, ed. Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2008). Washington: IEEE Computer Society, 2008. 40–54. [doi: 10.1109/SC.2008.5222643]

[5] Zhu YF, Mueller F. DVSleak: Combining leakage reduction and voltage scaling in feedback EDF scheduling. In: Pande S, Li ZY, eds. Proc. of the ACM SIGPLAN/SIGBED 2007 Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES 2007). San Diego: ACM Press, 2007. 31−40. [doi: 10.1145/1254766.1254772]

[6] Sampson A, Dietl W, Fortuna E, Gnanapragasam D, Ceze L, Grossman D. EnerJ: Approximate data types for safe and general low-power computation. In: Hall MW, Padua DA, eds. Proc. of the 32nd ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2011). San Diego: ACM Press, 2011. 164−174. [doi: 10.1145/1993498.1993518]

[7] Tolentino M, Turner J, Cameron KW. Memory MISER: Improving main memory energy efficiency in servers. IEEE Trans. on Computers, 2009,58(3):336−350. [doi: 10.1109/TC.2008.177]

[8] Shang PJ, Wang J. A novel power management for CMP systems in data-intensive environment. In: Sussman A, ed. Proc. of the 25th IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS 2011). Washington: IEEE Computer Society, 2011. 92−103. [doi: 10.1109/IPDPS.2011.19]

[9] Bianchini R, Rajamony R. Power and energy management for server systems. IEEE Computer, 2004,37(11):68−74. [doi: 10.1109/MC.2004.217]

[10] Li D, Nikolopoulos DS, Cameron KW, de Supinski BR, Schulz M. Power-Aware MPI task aggregation prediction for high-end computing systems. In: Bader DA, ed. Proc. of the 24th IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS 2010). Washington: IEEE Computer Society, 2010. 1−12. [doi: 10.1109/IPDPS.2010.5470464]

[11] Rountree B, Lowenthal DK, Funk S, Freeh VW, de Supinski BR, Schulz M. Bounding energy consumption in large-scale MPI programs. In: Verastegui B, ed. Proc. of the ACM/IEEE Conf. on High Performance Networking and Computing (SC 2007). San Diego: ACM Press, 2007. 1−9. [doi: 10.1145/1362622.1362688]

[12] Ge R, Feng XZ, Feng WC, Cameron KW. CPU MISER: A performance-directed, run-time system for power-aware clusters. In: Lai TH, ed. Proc. of the Int'l Conf. on Parallel Processing (ICPP 2007). Washington: IEEE Computer Society, 2007. 1−8. [doi: 10.1109/ICPP.2007.29]

[13] Song SW, Su CY, Ge R, Vishnu A, Cameron KW. Iso-Energy-Efficiency: An approach to power-constrained parallel computation. In: Sussman A, ed. Proc. of the 25th IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS 2011). Washington: IEEE Computer Society, 2011. 128−139. [doi: 10.1109/IPDPS.2011.22]

易会战(1976－),男,河北肃宁人,博士,副研究员,主要研究领域为低功耗优化,并行编程语言.
E-mail: huizhanyi@nudt.edu.cn

罗兆成(1974－),男,讲师,主要研究领域为低功耗优化.
E-mail: luopulzc@126.com