低功耗多线程编译优化技术*

赵荣彩[1], 唐志敏[1], 张兆庆[1], Guang R. Gao[2]

[1](中国科学院 计算技术研究所,北京 100080);
[2](Delaware 大学 电子与计算机工程系, Newark, DE 19716-3130,美国)
E-mail: rczhao@ict.ac.cn
http://www.ict.ac.cn

摘要: 提出了在多线程体系结构中通过降低执行频率有效减小功耗的理论模型和方法.首先研究识别可降频运行的线程的计算模型和降频因子的计算,然后给出在编译过程中基于对应用程序行为的分析,结合线程划分的低功耗编译优化算法和实现策略.该模型和方法可用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,既可开发 TLP(thread level parallelism),又可有效减小功率消耗.

关 键 词: 多线程;低功耗;编译优化;并行处理;计算机体系结构
中图法分类号: TP311 文献标识码: A

在现代的计算机系统中,编译器和处理器结构是密切相关的,因此,大部分后端编译优化都是特定于目标系统的具体结构和实现特征的.这种处理的首要目标就是提高系统的效率和性能.多年来的研究工作都是致力于最优化系统功能部件的利用率和充分开发应用的并行性,如指令级并行(instruction level parallelism,简称 ILP)、线程级并行(thread level parallelism,简称 TLP)、任务级并行等.随着近年来支持多现场(context)执行的不同体系结构的研究和发展,开发 TLP 的潜力和研究意义更为突出[1].开发 TLP 主要是通过提高多现场同时执行的并行度以及用一个线程的并行执行来容忍(torlerate)或隐藏(hiding)另一线程的长延迟操作来提高整个系统的性能[2,3].尤其是,细粒度线程级并行的开发是目前体系结构和编译研究努力追求的主要目标之一.它要求硬件结构在不同现场间线程切换的开销减至最小甚至零开销[4],要求编译对体系结构的特征更有针对性,对应用程序的行为分析更为精细和准确.

编译技术研究除了追求应用程序运行效率的最优化之外,通过编译对应用程序行为的充分分析,对不显著响程序运行性能的条件下最小化系统或处理器的运行功耗的研究工作在近期倍受关注[5~8].这是电路级设计多年来一直追求但又不能涉足的有效降低功耗的一个潜在领域.因此,低功耗研究必须走出电路技术和低层设计的限制.处理器的能量和功率消耗在移动计算、嵌入应用和高性能处理器设计中已成为明显的发展瓶颈,当最大功率消耗成为关键的设计限制时,最大化性能/功耗比也就是最大化性能了.

多线程处理器因其吞吐率高,使得它在很多低功耗和功率受限设备的设计和应用中具有很大的吸引力.这是由于多线程结构允许线程选择机制和指令动态分配机制有更大更灵活的调度和权衡空间,因此,在低功耗优化的研究上更有潜力.

目前,在单线程、超标量结构上已有通过编译指导的频率动态调整降低功耗的相关研究[5,7],也有不少通过其他编译优化或体系结构级硬件实现降低功耗的相关研究[6,8].在基于不同结构的多线程划分方面已有很多研

* 收稿日期: 2001-12-25; 修改日期: 2002-03-11
基金项目: 国家高技术研究发展计划资助项目(2001AA111070)

作者简介: 赵荣彩(1957－),男,河南洛阳人,教授,主要研究领域为体系结构,先进编译技术;唐志敏(1966－),男,江苏苏州人,博士,研究员,博士生导师,主要研究领域为计算机体系结构,并行计算,VLSI 设计;张兆庆(1938－),女,北京人,研究员,博士生导师,主要研究领域为先进编译及相关工具环境;**Guang R. Gao**(1945－),男,北京人,教授,博士生导师,主要研究领域为计算机体系结构,并行计算,先进编译.

究,由 Guang R. Gao 领导的 CAPSL 实验室在 EARTH 项目中,对粗、中、细粒度的多线程划分已有深入的研究[3].但对多线程模型的低功耗编译优化技术的研究此前还未见报道.本文重点提出了一种基于频率动态调整的结合细粒度多线程划分的低功耗优化模型,设计了相关的算法和编译实现策略,旨在不影响充分开发应用程序 TLP 的同时,尽可能有效地减少处理器的运行功耗.

1 多线程划分模型与低功耗优化的基本思想

1.1 频率调整功耗模型的思想基础

一个程序的总执行时间(T)和能量消耗(E)可以通过以下公式来估计:

$$T \approx W.1/f, \quad E \approx C.W.V^2.$$

W 是总的执行时间(cycles),f 是时钟频率,C 是容量交换效率,V 是提供的电压.假定 C 和 W 与 f 不相关.由于在动态电压调整(DVS)系统中,电压 V 的变化与频率 f 为线形关系($V \propto f$).对频率调整的性能/能耗的权衡关系可由以下公式表示:

$$T \propto 1/f, \quad E \propto f^2.$$

降低时钟频率就会减少能量消耗,但有可能以性能损失为代价.

我们知道,由于应用程序对系统各功能部件利用的不均衡性(如存储受限或 I/O 受限等),使得某段代码或某些功能部件成为关键段或瓶颈,而其他部分则经常处于等待状态.如果保证在不超过合理的限定时间 d,对非关键代码或非瓶颈部件有可能进行降低频率执行,达到降低功耗的效果.采用这一方法,经过在超标量单线程处理器上对大量典型的 SPEC95 进行模拟研究,相应的功耗可降低 23%~75%[7].由于能耗与频率的平方成正比,因此,通过动态调整电压/执行频率是显著降低 CPU 功耗的主要措施之一.

1.2 线程划分的基本思想

为了支持低功耗编译优化,本文主要考虑基于支持多线程的单片多 EU(execution unit)结构模型.设定每一个 EU 的执行频率在一定范围内,程序通过特定的指令可以对其进行独立的动态调整.基于这种多线程结构模型,图 1 给出了一个启示性例子的简单数据依赖代价图(data dependency graph,简称 DDG),所有的程序都可以表示为一个 DDG[2,3],每一个节点 v_i 附上一个执行代价 c_i.我们的目的是把 DDG 的节点划分为多个线程,在满足数据依赖关系并最小化执行时间同时最小化功率消耗.

对于具有两个 EU 的结构,若线程间切换代价 δ=4,可有划分:$TA=\{v_a\}$,$TB=\{v_b\}$,$TC=\{v_c,v_d,v_e\}$,$TD=\{v_f\}$,$TE=\{v_g\}$以及如图 2 所示的调度.线程 TA 和 TC 后有阴影的扩展部分,说明这两个线程是非关键代码,可以降频运行,在不影响整个程序执行性能的同时,它们可以分别降频 1/2 和 1/9 来执行,相应线程的执行功耗按第 1.1 节的计算关系则可相应降低 75%和 21%,进而达到显著降低功耗的目的.当一个 EU 结束一个线程,且同一应用中又没有在该 EU 上执行的后续线程时也可关闭该 EU,以降低功耗(如在 TC 之后可关闭该 EU).在并发线程比较多的应用中,线程的分配策略不同对功耗的影响也会有所不同,详见第 2.3 节给出的线程调度算法.

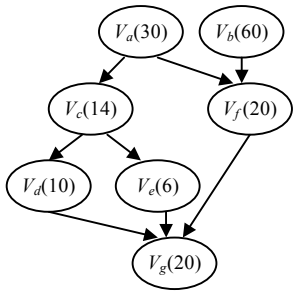

Fig.1 An example by using DDG with cost
图 1 表示程序数据依赖代价图的例子

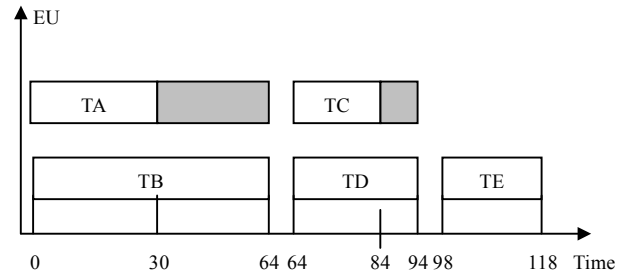

Fig.2 Thread execution schedule
图 2 线程的执行调度

2 低功耗模型与线程划分的编译优化策略研究

2.1 问题的形式化

2.1.1 基本定义

任意一个程序都可表示为它的数据依赖图 G,G 中的每个节点 v_i 都附有它的执行代价 c_i.用大写字母 P 表示对 G 的线程划分,T_K 为划分 P 中的第 K 个线程.与线程划分相关的概念采用与文献[2]基本一致的描述.

开始节点和结束节点.先把两个节点 v_{start} 和 v_{end} 附加到数据依赖图 G 中,再从 v_{start} 到每一个没有先辈的节点之间附加上相应的边,再从每一个没有后继的节点到 v_{end} 之间附加上相应的边.经过这样的变换之后,G 就可表示为一个单入口和单出口的图.

节点高度.一个节点 v_i 的高度 h_i 定义如下:

$$h_i=c_i+\max h_j,\quad \{j\,|C_{ij}=1\}.$$

这里,C 是 G 的邻接矩阵.定义 $h_{\text{end}}=0$,则一个节点的高度就代表从该节点开始执行到程序结束的关键路径的时间度量.节点的高度用从该节点 v_i 到结束节点 v_{end} 的执行节拍(cycles)来度量.

节点的开始时间.节点 v_i 的开始时间 s_i 就是该节点在 G 的划分 P 的调度中执行开始的时间.

节点的完成时间.节点 v_i 的完成时间 f_i 是该节点操作执行完成的时间,并且所有 v_i 产生的结果对其后继都是可用的.

$$F_i=s_i+c_i.$$

节点准备好时间.节点 v_i 的准备好时间 r_i 是节点 v_i 需要的所有数据都已经是可用的时间,亦即 G 中 v_i 的所有先辈都已经执行结束了.即

$$r_i=\max f_j,\quad \{j\,|C_{ji}=1\}.$$

应该看到,对于 G 的所有节点 v_i,应有 $r_i\le s_i$.

线程的开始时间.线程 T_K 的开始时间 S_K 是在划分 P 的调度中该线程开始执行的时间.

线程的完成时间.线程 T_K 的完成时间 F_K 是该线程中所有操作完成的时间加上线程的切换代价 δ.即

$$f_K=S_K+\delta+\sum c_i,\quad \{i|v_i\in T_K\}.$$

父线程集.节点 v_i 的父线程集 $PT(v_i)$的构成:

$$PT(v_i)=\{T_K|T_K\in P,(\exists j)(v_j\in T_K,C_{ji}=1)\}.$$

密切关系.G 中节点 v_i 与 P 中线程 T_K 的密切关系 $A(v_i,T_K)$定义为 T_K 中各节点到 v_i 的总边数与 v_i 的总入边数之比:

$$A(v_i,T_K)=\left(\sum_{v_j\in T_K}C_{ji}\right)\Big/\left(\sum_{v_j\in G}C_{ji}\right).$$

可以看出,如果 v_i 的所有入边都来自 T_K,则 $A(v_i,T_K)=1$;另一方面,如果 v_i 没有一条来自 T_K 的入边,则 $A(v_i,T_K)=0$.

一个节点 v_i 加入 T_K 之后的结果线程不可能在 T_K 或 v_i 的准备好时间之前开始执行,因此,由于是 v_i 加入到 T_K 中,则可能产生延迟执行,其延迟时间 $d(v_i,T_K)$可计算如下:

$$d(v_i,T_K)=\begin{cases}F_K-r_i & \text{if } PT(v_i)==\{T_K\}\\ \max(r_i,S_K)+(F_K-S_K)-r_i & \text{if } PT(v_i)!=\{T_K\}\end{cases}.$$

同时,再定义一个损耗函数 $P(v_i,T_K)$来决定节点 v_i 加入 T_K 的合适程度:

$$P(v_i,T_K)=D*d(v_i,T_K)+L*(1-A(v_i,T_K)).$$

D 和 L 是常数系数,D 表示节点 v_i 的执行对整个执行时间的延迟的影响参数,L 表示把 v_i 加入到已经包含了很多 v_i 先辈的这个线程中之后可获得的数据局部性的重要程度.线程的节点划分算法将把节点 v_i 分配到产生最小损耗的线程 T_K 之中.

2.1.2 低功耗优化模型

假定 Q 是在对 n 个 EU 的线程划分 P 的一种低功耗调度,可将 Q 定义为

$$Q=\{Q_1,Q_2,\ldots,Q_n\},$$

$$Q_i=\langle T_{i1},T_{i2},\ldots,T_{im}\rangle,\quad i=1,\ldots,n.$$

调度 Q 定义为由 n 个线程队列组成,每个线程队列 Q_i 定义为被调度在第 i 个 EU 上运行的 m(对不同的 Q_i,m 可能是不同的)个线程的有序执行.

如果一个线程 T_{ij} 可以降低频率执行,必须满足以下条件:

$$\mathrm{Min}(S_{ij+1},\mathrm{Min}(S_{gf}))-F_{ij}>0,\quad \{g|g\neq i,(\exists v_k\exists v_h)(v_k\in T_{ij},v_h\in T_{gf},C_{kh}=1)\}.$$

线程执行的降频因子.线程 T_{ij} 执行的降频因子 TF_{ij} 是为降低功耗而把 T_{ij} 降频后执行的频率与正常频率的比值.

$$TF_{ij}=(F_{ij}-S_{ij}-\delta)/(\mathrm{Min}(S_{ij+1},\mathrm{Min}(S_{gf}))-S_{ij}-\delta).$$

$$\{g|g\neq i,(\exists v_k\exists v_h)(v_k\in T_{ij},v_h\in T_{gf},C_{kh}=1)\}.$$

一个线程被降频后运行所节省的能量 E_{decrease} 与正常运行所用能量 E_{normal} 的百分比 TP_{ij} 可进行如下计算:

$$TP_{ij}=(E_{\mathrm{decrease}}/E_{\mathrm{normal}})\times 100\%=(1-TF_{ij}^2)\times 100\%.$$

由此,可以很容易地计算出一个应用程序经过这种低功耗优化后运行时所节省的总能耗.

2.1.3 问题陈述

在上面定义的基础上,提出本文所要解决的关键问题:

问题陈述.假定在运行时,在单一的线程准备好队列中用有效调度器选择线程执行.现给定数据依赖关系图 G,表示其中每个节点的代价 c_i 和线程切换代价常数 δ,寻找一个满足以下目标的线程划分 P 和调度 Schedule:(1) 通过最小化结束节点 v_{end} 的开始时间最小化整个程序的执行时间;(2) 在不损耗性能的情况下,尽可能有效地降低功率消耗.

2.2 功耗优化的线程划分算法

功耗优化划分算法根据 G 中记录的计算代价和 EU 的可用情况来计算线程划分 P 和功耗优化调度.图 3 中的算法 *PowerAwarePartition*(G)在 while 循环的每一轮,从 F 中选取一个最高节点 v_i(第 7 步),或者在划分 P 中创建一个新线程,或者将 v_i 加入到 P 中已经存在的一个线程中(第 8 步、第 9 步).然后,把那些还没有被分配但其父节点已经被分配了的节点加入到 F 中(第 10 步),并调整事件表 E(第 11 步).这一过程一直重复到 F 为空为止.事件表中的每一个元素包含着一个事件的时间和类型,事件类型 S 表示是一个线程的开始事件,事件类型 F 表示是一个线程的结束事件.

```
PowerAwarePartition(G)
1.  Add v_satrt and v_end to G
2.  Calculate h_i for every node v_i in G
3.  P←∅
4.  E←∅
5.  F←{v_start}
6.  while F!=∅ {
7.          v_i←Highest node in F
8.          T_K←FittestThread(P,E,v_i)
9.          InsertIntoThread(v_i,T_K)
10.         UpdateReadyList(F,v_i)
11.         UpdateEventList(E,T_K)
12.     }
13. LowPowerSch(G,E)   //power-aware
14. end
```

Fig.3 Thread partition algorithm with low power optimization

图 3 低功耗优化的线程划分算法

事件表 E 是一个按时间发生顺序的有序事件表,它在 FittestThread 和 LowPowerSch 算法中起着重要的作用.最后,根据 DDG 和事件表 E 中各线程的开始和结束事件,进行低功耗优化,识别可降频运行的线程并计算每个线程的降频因子和线程调度(第 13 步).

G 的一个重要特性是从 v_{start} 到 G 中的每一个节点之间都存在至少一条通路.因此,划分算法中的第 6 步~第 11 步的循环终能遍历 G 中的所有节点,每一轮循环调度一个节点.第 7 步选取一个最高节点 v_i,第 8 步调用的 FittestThread 算法返回一个 v_i 最合适加入的线程,第 9 步调用 InsertIntoThread 算法寻找节点 v_i 应加入该线程的最佳调度位置.

该算法的结构组织成明显的两个阶段,第 1 步~第 12 步主要完成线程的划分阶段的工作,通过第 13 步调用功耗优化算法完成低功耗线程调度阶段的优化工作.这种结构的优点是:第 1,充分体现了尽量优化性能的情况下,降低功耗的基本优化思想;第 2,结构清楚,先集中精力实现性能优化的线程划分,再进行低功耗优化,比融合在一起算法简单,易于编译结构的组织和实现;第 3,便于将该低功耗优化思想和算法与该类模型的其他更好的线程划分算法相配合的组织和实现.

2.2.1 FittestThread 算法

FittestThread(P,E,v_i)算法(如图 4 所示)是考虑到低功耗研究的需要对文献[2,3]中的 EarlistThread 算法进行适当改进的相应算法,它使用损耗函数和密切关系权衡对 v_i 的调度,并尽可能早地把它安排在已经包含有与 v_i 密切相关的一些操作的线程中.

FittestThread(P,E,v_i)算法首先调用 TestGap 算法验证 v_i 是否能在它的已经准备好时间 r_i 时启动自己的线程(第 2 行),TestGap 算法返回 True 或 False 指明是否有合适的时间间隙可容纳该线程的执行,如果为 True,则为 v_i 创建一个新的线程(第 5 行).当存在空闲 EU 时,为一个节点创建新线程是对多 EU 结构开发应用中所存在的最大 TLP 的需要.

然后,*FittestThread*(P,E,v_i)算法对 v_i 的父节点集 $PT(v_i)$中的所有线程计算最小损耗 P_m(第 4 行~第 11 行).只有那些 v_i 可以加入的线程才被列为最小损耗线程,并查找一个在 v_i 的准备好时间 r_i 之后最早结束的线程 T_f (第 12 行),第 13 行~第 18 行的 while 循环从最早结束事件的位置开始扫描事件,找出调度中足以容纳 v_i 的最早时间间隙.

```
FittestThread(P,E,vi)
1.  compute ri
2.  if TestGap(P,E,ri,ri+ci+δ)
3.  then return Tnew
4.  Pm←∞
5.  for Tk∈PT(vi) {
6.    do p←P(vi,Tk) {
7.       if p<Pm
8.       then {S'←max(ri,Sk)
                F'←S'+Fk−Sk+ci}
9.       if TestGap(P,E,S',F'+δ)
10.      then {m←k
11.             Pm← p}}}
12. f←EarliestFinishEvent(E,ri)
13.  while((Ff−ri)D<Pm and f<n) {
14.      if (P(vi,Tf)<Pm and TestGap(P,E,Ff,Ff+ci))
15.      then return Tf
16.      if TestGap(P,E,Ff,Ff+ci+δ)
17.      then return Tnew
18.      f← f+1}
19.  return Tm
```

Fig.4 Algorithm for searching the best thread

图 4 最佳线程搜索算法

2.2.2 TestGap 算法

TestGap(P,E,s,f)算法是对事件表进行搜索,验证在时间间隔[s;f]期间是否至少存在一个 EU 是可用的.直观上,它计算在时间 s 有多少个线程被调度,然后,遍历从 s 开始的事件表,当发现一个开始事件时,就将计数器加 1;当发现一个完成事件时,就将事件计数器减 1.如果在[s;f]期间的任何时候,计数器等于机器中可用的 EU 数目,该算法就返回 False.

如果它搜索到了间隔 f 的终点,同时计数器还没有超过 N–1 个线程(N 为机器中总 EU 的数目),该算法就返回 True.可参见文献[2]对 TestGap 和 InsertIntoThread 的相关描述.

2.3 低功耗线程调度和降频因子的计算

如图 5 所示,LowPowerSch 算法的工作大致分为两步:第 1 步按照事件表中的顺序给每一个线程分配 EU 号和建立每一个 EU 的线程运行队列.调度表中除了降频因子 LowPowerFactor 外的所有项目的填写均在这一步完成;第 2 步计算每一个线程的降频因子.每一个 EU 在运行完给它分配的线程之后是否关闭,可根据它的最后一个线程的结束时间与整个该应用程序的结束时间之差而定.每一个线程在调度表 *Schedule*[n]中相应的 EU 线程队列中占一项,每一项有 5 个域:线程号 ThreadNumber、该线程的开始时间 SK、正常结束时间 FK、降频执行因子 LowPowerFactor 和指向该 EU 队列的下一个线程的指针 NextThread.其中函数 *Thnumber*(*E*(*evt*))是从事件表中提取产生事件 evt 的线程号.δ是线程的切换时间.运行时(runtime)系统即可按照调度表 *Schedule*[n]的安排和降频因子来调度执行该应用的各个线程.

在图 5 中的第 5 行~第 9 行完成线程到处理机 EU 的分配,每次分配都是从 0 号处理部件开始查找,应该说是一种不均衡的分配,与传统的负载均衡分配原理相违背,但在这里是有意义的.第 1,前面所述的线程划分的机理使得这种分配方法能够保证该应用程序的执行性能不会降低;第 2,负载集中在编号小的 EU 上,较大编号 EU

长期空闲的可能性增大,它们被关闭(turn off)的时间也相应增多,因而可实现最大可能降低功耗的目的.

```
LowPowerSch(G,E)
1.  begin
2.  set all of a[n] idle
3.  Forall evt in event list E{
4.     if Type(evt)==S{
5.          for i=0 to n−1
6.              search the first i which a[i] is idle
7.          add one item on the tail of Schedle[i]
8.          SK←time(E(evt))
9.          ThreadNumber←Thnumber(E(evt))
10.         set a[i] busy}
11.    if Type(evt)==F {
12.       search the item in Schedule[i]
13.         FK←time(E(evt))
14.         set the a[i] idle}}
15. Forall T_K in Schedule[] {
16.    SE←NextThread→SK
17.    Forall v_i in T_K
18.          if ((C_ij==1 and v_j not in T_K)
                 and (v_j∈T_J and S_J<SE))
19.              SE←S_J
20.    TF←(FK−SK)/(SE−SK−δ)}
21. end
```

Fig.5 Algorithm computing thread schedule and the execution frequency factor for low power
图 5 计算线程调度和低耗降频因子的算法

3 结束语

本文重点提出了一种解决多线程低功耗的编译优化技术问题,这也是目前倍受关注的研究方向.当前,不仅主机端有大量的 multiprocessor 结构平台,而且,SMT 结构的机器很快就要面市,就连网络设备专用的处理器都采用了硬件支持多线程的多现场结构,如 Intel 最近推出的网络处理器 IXP1200 系列,采用 SoC 技术在单片中容纳了 1 个 StrongARM 处理器和 6 个 micro-engine,每个 micro-engine 都是支持 4 个硬件现场的处理器.其中 StrongARM 的下一代替换产品 XScale 的执行频率可由程序控制动态调整支持低功耗优化.这样的高集成度和多硬件现场,其现场(线程)间切换的开销几乎近于零.只有具有大量细粒度的 TLP 才能充分体现它们的优势,而且希望能够尽可能的低功耗运行,低功耗就能带来可靠,带来性能.

致谢 本文的工作是在中国科学院计算技术研究所与美国 Delaware 大学电子与计算机工程系的联合实验室完成的.Guang R. Gao 的 EARTH 项目研究对本文的工作提供了很好的基础[2,3,5].本文的研究工作受到了国家高技术研究发展计划的资助,在此一并表示感谢.

References:

[1] Lo, J., Eggers, S., Levy, H., *et al*. Tuning compiler optimizations for simultaneous multithreading. In: Srimani, P., Fayad, M.E., eds. Proceedings of the 30th Annual International Symposium on Microarchitecture. Los Alamitos: IEEE Computer Society, 1997. 114~124.

[2] Gao, G.R, Tang, Xinan, Wang, Jian, *et al*. Thread partitioning and scheduling based on cost model. In: Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures. New York: ACM Press, 1997. 272~281.

[3] Theobald, K.B. EARTH: an efficient architecture for running threads [Ph.D. Thesis]. Montreal: McGill University, 1999.

[4] Tullsen, D.M., Eggers, S.J., Levy, H.M. Simultaneous multithreading: maximizing on-chip parallelism. In: Sohi, Gurindar, ed. 25 Years of the International Symposium on Computer Architecture: Selected Papers. New York: ACM Press, 1998. 115~116.

[5] Yang, Hong-bo, Gao, G.R., Marquez, A., *et al*. Impacts of loop optimization on power consumption——a fresh look. In: Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP) 2001. 2001. http://www.capsl.udel.edu/~hyang/personal.html.

[6] Seng, J.S., Tullsen, D.M, Cai, G.Z.N. Power-Sensitive multithreaded architecture. In: Srimani, Pradip, ed. Proceedings of the International Conference on Computer Design 2000. Los Alamitos: IEEE Computer Society, 2000. 199~206.

[7] Hsu, C-H., Kremer, U., Hsiao, M. Compiler-Directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In: International Symposium on Low Power Electronics and Design (ISLPED'01). 2001. 275~278. http://www.cs.rutgers.edu/~uli/pubs.html.

[8] Seng, J.S., Tune, E.S., Tullsen, D.M. Reduceing power with dynamic critical path information. In: Williams, M., ed. Proceedings of the 34th Annual International Symposium on Microarchitecture. Los Alamitos: IEEE Computer Society, 2001. 114~123.

A Multithreaded Compiler Optimization Technology with Low Power*

ZHAO Rong-cai[1], TANG Zhi-min[1], ZHANG Zhao-qing[1], Guang R. Gao[2]

[1](*Institute of Computing Technology*, *The Chinese Academy of Sciences*, *Beijing* 100080, *China*);

[2](*Department of Electronic and Computer Engineering*, *University of Delaware*, *Newark*, *DE* 19716-3130, *USA*)

E-mail: rczhao@ict.ac.cn

http://www.ict.ac.cn

Abstract: A theoretic model and method is proposed to decrease power consumption effectively by reducing execution frequency on multithreaded architecture in this paper. At first, the computation model is studied to recognize the thread which can be executed at lower frequency, and the factor is computed to slow down the frequency. Then, an algorithm and policy of compiler optimization combining with thread partition for low power is given based on the analysis of application program. This model and method can be used to exploit TLP (thread level parallelism) and decrease the power consumption effectively for the multithreaded multiprocessor architecture with scalable execution frequency.

Key words: multithreading; low power; compiler optimization; parallel computing; computer architecture

* Received December 25, 2001; accepted March 11, 2002
Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA111070