

软件流水的低功耗编译技术研究*

赵荣彩^{1,2}, 唐志敏², 张兆庆², Guang R. Gao³

¹(信息工程大学 计算机科学与技术系,河南 郑州 450002)

²(中国科学院 计算技术研究所,北京 100080)

³(特拉华大学 电子与计算机工程系,美国)

Study on the Low Power Technology of Software Pipeline

ZHAO Rong-Cai^{1,2}, TANG Zhi-Min², ZHANG Zhao-Qing², Guang R. Gao³

¹(Department of Computer Science and Technology, University of Information Engineering, Zhengzhou 450002, China)

²(Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

³(Department of Electronic and Computer Engineering, University of Delaware, Newark, USA)

+ Corresponding author: Phn: 86-371-3531190, Fax: 86-371-3531761, E-mail: rczhao@ict.ac.cn

<http://www.ict.ac.cn>

Received 2002-09-10; Accepted 2003-03-24

Zhao RC, Tang ZM, Zhang ZQ, Gao GR. Study on the low power technology of software pipeline. *Journal of Software*, 2003,14(8):1357~1363.

<http://www.jos.org.cn/1000-9825/14/1357.htm>

Abstract: Based on loop-carried dependence of global schedule, the compiler optimization technology of low power software pipeline scheduling is studied as a kind of ILP formulation for a given loop \mathcal{L} under the configuration \mathcal{M} with dynamic frequency/voltage scalable multi-function units. A reasonable and effective method of power-aware optimal software pipeline scheduling is proposed. It can make the loop run with minimized power/energy and without performance penalty.

Key words: software pipeline; low power; compiler optimization; dynamic frequency scalable; parallel processing

摘要: 对具有可动态独立调整运行频率/电压的多功能部件配置结构 \mathcal{M} , 基于全局调度的循环依赖关系, 使用 ILP 形式化框架, 研究了对给定循环 \mathcal{L} 进行动态频率/电压调整的低功耗软件流水调度的编译优化技术. 提出了一种合理而有效的低功耗最优化软件流水调度方法, 使其在运行时保持性能不变而消耗的功耗/能量最小.

关键词: 软件流水; 低功耗; 编译优化; 动态频率调整; 并行处理

中图法分类号: TP314 文献标识码: A

软件流水是一种非常重要的细粒度循环调度方法, 它产生一种周期模式的并行调度, 使得相邻轮次循环的指令尽可能地重叠执行, 以充分开发指令级的并行性. 它可以应用于高性能流水线处理结构、超标量结构和

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA111070 (国家高技术研究发展计划(863))

第一作者简介: 赵荣彩(1957—), 男, 河南洛阳人, 教授, 博士生导师, 主要研究领域为体系结构, 先进编译技术.

VLIW 结构^[1-4]。为了成功地在实际机器上实现,必须要考虑具体机器的功能部件资源限制。目前已经提出了很多种考虑资源限制的软件流水调度算法,其中基于整数线性规划(integer linear programming,简称 ILP)的方法^[1,3]是对具有简单有限资源机器寻求最优化周期调度的方法,文献[4]对软件流水方面的算法作了很好的综述。

近 10 年来,我们已经看到通用微处理器性能的巨大提高,然而,这种高性能通常都伴随着过高的功耗。功耗问题将成为高性能微处理器发展的主要限制因素之一^[5]。因此,结构设计者必须在发展处理器的设计时考虑功耗/性能的权衡问题。

近期对于低功耗设计的研究提出了具有可以动态调整运行时主要功能部件的执行频率的体系结构。在这样的处理器中,一些功能部件可以在程序运行时动态调整和控制它们的执行速度/电压,在软件运行时通过对这些特性的合理应用,得到有效的功耗/能量节省^[5]。

产生这种设计的动因是在指令调度时存在有指令调度空隙,即一些指令不处在关键路径上,因此,延长它们的执行时间不会对性能造成明显的影响。通过对关键路径上的指令以全速运行,而对处在非关键路径上的指令按降低的频率/电压运行,可获得显著的功耗/能量节省^[5]。

在给定时间和资源限制时,面向性能的软件流水可被看成是 ILP 问题^[1,6],一些优秀的软件流水算法已经在产品编译器中得到了很好的应用^[2,7]。但是在低功耗方面,既最优化性能又最小化功耗/能量消耗的有效软件流水算法还很少^[8]。本文的主要工作是对具有可动态独立调整运行频率/电压的多功能部件配置结构 \mathcal{M} ,基于全局调度的循环依赖关系,使用 ILP 形式化框架,对给定循环 \mathcal{L} 进行动态频率/电压调整的低功耗软件流水最优化调度的编译技术研究。提出了一种合理而有效的低功耗最优化软件流水调度方法,使其在运行时保持性能不变而消耗的功耗/能量最小。

1 问题求解示例

本节我们用一个启示性的示例来说明在非流水(non-pipeline)执行部件情况下具有低功耗优化的软件流水调度问题,执行部件为流水部件且具有结构冒险(hazard)时具有低功耗优化的软件流水调度问题可以在此基础上进行发展。

为了对比,我们引用与文献[1]相同的一个简单的循环例子,如图 1 所示。图 1(b)给出了该例子的 C 程序和指令级表示,图 1(a)是它的数据依赖关系图(DDG)。

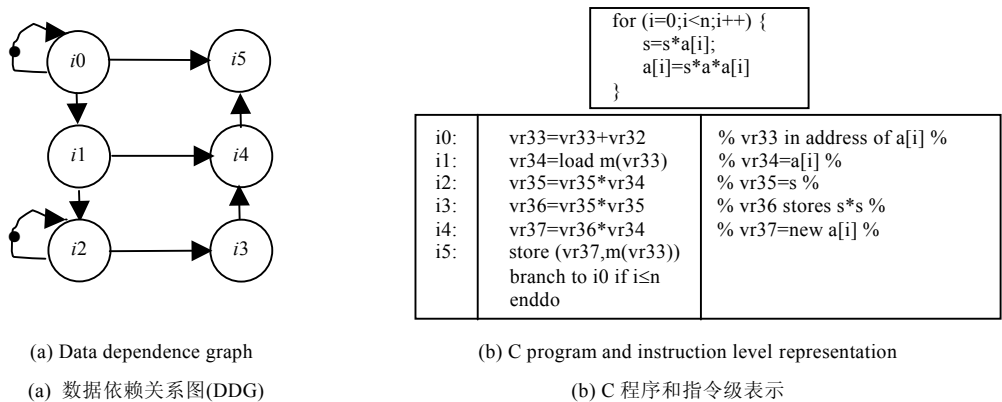


Fig.1 An example for a loop

图 1 一个循环的例子

为了便于讨论,考虑该体系结构具有 3 个整数部件、两个浮点部件(FP)和一个存储(load/store)部件。整数部件执行所有的整数操作,浮点部件执行所有的浮点操作,整数加操作(i0)的执行时间为 1 个节拍(cycle),浮点乘以操作(i2,i3,i4)的执行时间为 2 个节拍,读操作(i1)为 2 个节拍,存操作(i5)为 1 个节拍。所有的部件为非流水部件,即在同一个部件上前一个操作执行完后才能开始执行下一个操作。我们知道,在软件流水中相继循环的最小启动间隔 T_{min} 是受循环(loop-carried)依赖和可用资源的限制的^[1-3]。循环依赖决定了 T 的一个下界 T_{dep} , T_{dep} 的值则

是由循环中的关键循环依赖的距离而定^[1].特别是在关键依赖循环上时,

$$T_{dep} = \frac{\text{关键循环上的指令执行时间总和}}{\text{关键循环的依赖距离总和}}$$

对图 1(a)中的 DDG, T_{dep} 为 2 是对应指令 i_2 的自循环.

T 的另一个下界是有资源限制所决定的 T_{res} . 考虑:(1) 循环中使用类型 r 的功能部件 FU 的所有操作(如浮点部件 FP);(2) 有 F_r 个 r 类型的部件;(3) 在类型 r 的部件 FU 上所有指令的执行时间和为 T_r , 则

$$T_{res} = \max_x \left\{ \frac{T_r}{F_r} \right\}$$

在该例子中, T_{res} 的值是由 FP 和存取部件决定的, 即

$$\begin{aligned} T_{res} &= \max(T_{Int}, T_{FP}, T_{load/store}) \\ &= \max\left(\left\lceil \frac{1}{3} \right\rceil, \left\lceil \frac{2+2+2}{2} \right\rceil, \left\lceil \frac{2+1}{1} \right\rceil\right) \\ &= \max(1, 3, 3) \\ &= 3. \end{aligned}$$

最短的启动间隔 T_{lb} 是 T_{res} 和 T_{dep} 的最大值. 也可能存在或不存在满足给定资源限制的周期为 T_{lb} 的调度. T_{min} 是调度存在的最小周期, 因此, $T_{min} \geq T_{lb}$.

图 1(a)中 DDG 的调度可表示为线性调度 $T \times i + t_s$, 其中 $T=3, t_{i0}=0, t_{i1}=1, t_{i2}=3, t_{i3}=5, t_{i4}=7$, 和 $t_{i5}=9$. 用 $_i1$ 表示指令 i_1 在当前时间节拍继续执行. 表 1 说明了调度 A 的指令执行情况. 由于假定 FU 是非流水部件, 指令就会在当前节拍继续占用 FU. 调度 A 由 3 部分组成: 前奏(prolog)(节拍[0,6])、重复模式(repetitive pattern)(节拍[7,9])和后续(epilog). 在重复体中是满足资源限制的, 即在任何时刻对资源的使用都不多于 3 个整数部件、2 个浮点部件和 1 个存取部件. 换言之, 就是说调度 A 既满足依赖限制又满足体系结构的资源限制. 表中两条黑粗线之间的部分为软件流水调度每一轮循环的重复模式.

Table 1 The schedule A of the example

表 1 例子的调度 A 方案

Iteration	Time steps															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	i_0	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4	$_i4$	i_5						
1				i_0	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4	$_i4$	i_5			
2							i_0	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4	$_i4$	i_5
3										i_0	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$
4													i_0	i_1	$_i1$	i_2

从调度 A 可以看出, 在整个循环体中, 整数部件只有一条指令 i_0 , 它的执行时间仅需要 1 个节拍, 其余 2 个节拍都是空闲的, 如果不降低性能的情况下, 降低它的运行频率/电压, 则可有显著的功耗/能量节省空间. 若使其降低频率后执行所占用的时间为 3 个节拍的时间(最慢也只能是 3 个节拍, 再慢就会使 T 增加), 则有表 2 所示的调度 B. 调度 B 与调度 A 相比, 循环体的执行时间相同, 相继循环的启动间隔也相同, 只是调度 B 的前奏和后续要比调度 A 分别多 2 个节拍, 这对性能没有产生明显的影响(即 T 不变), 但整数部件的功耗理论值应降低 2/3.

Table 2 The schedule B of the example

表 2 例子的调度 B 方案

Iteration	Time steps															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	i_0	$_i0$	$_i0$	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4	$_i4$	i_5				
1				i_0	$_i0$	$_i0$	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4	$_i4$	i_5	
2							i_0	$_i0$	$_i0$	i_1	$_i1$	i_2	$_i2$	i_3	$_i3$	i_4
3										i_0	$_i0$	$_i0$	i_1	$_i1$	i_2	$_i2$
4													i_0	$_i0$	$_i0$	i_1

因此, 我们要考虑的主要问题是把功耗优化问题与软件流水优化统一在形式化的 ILP 框架中, 即给定一个循环 \mathcal{L} 和具有功能部件可独立进行动态频率调整的处理器配置 \mathcal{M} , 找出一种合理的最优化的软件流水调度, 使其消耗的功耗/能量最小.

这里使用术语“合理的最优化调度”,意指对 \mathcal{L} 不存在其他的调度对处理器配置 \mathcal{M} 有更小的启动间隔 Π (initiation interval,即对应于该例子中的启动间隔周期 T);使用术语“最小功耗/能量调度”,意指在处理器配置 \mathcal{M} 上对循环 \mathcal{L} 不存在具有给定 Π 的其他功耗/能量更小的调度。

2 对问题的 ILP 形式化描述

在本节我们把对软件流水的整数线性规划框架^[7]扩展为具有低功耗优化的形式描述。

2.1 软件流水的 ILP 形式化描述

令 DDG 中的节点数为 N , E 为节点集合, Π 为相继循环的启动间隔^[2,3]。本节只考虑重复(repetitive)调度和模(modulo)调度。软件流水方法尝试从 MII(minimum initiation interval)^[2,3]开始,对每一个 Π 的值寻找一个调度,直到找到存在软件流水调度的第 1 个大于或等于 MII 的 Π 就停止。

在模调度中,在第 j 轮循环(iteration)中操作 i 的调度时间为

$$t_{ij} = t_i + j \times \Pi,$$

t_i 是操作 i 在重复模式第 1 轮循环时的调度时间。我们用一个 N 元向量 Γ 代表在重复模式第 1 轮循环中所有操作的调度时间,向量 Γ 由变量 t_0, \dots, t_{N-1} 组成,用一个 $\Pi \times N$ 的矩阵 A 来表示在模调度中指令的重复模式。矩阵 $A = [a_{t,i}]$, 当且仅当在模调度中节点(指令) i 在时刻(节拍) t 被调度执行时, $a_{t,i} = 1$, 否则 $a_{t,i} = 0$, 而 $t \in [0, \dots, \Pi - 1]$ 。显然:

$$a_{t,i} = 1, \text{ 若 } t_i \bmod \Pi = t.$$

进而 Γ 和 A 存在以下关系:

$$\Gamma = \Pi \times \kappa + A^T \times [0, 1, \dots, \Pi - 1]^T, \quad (1)$$

向量 κ 是一个由 k_0, \dots, k_{N-1} 组成的 N 元向量。式(1)隐含定义了:

$$k_i = \left\lfloor \frac{t_i}{\Pi} \right\rfloor.$$

由于每一条指令 i 在重复模式中只被调度 1 次,因此,一个正确的调度必须满足以下条件:

$$\sum_{t=0}^{\Pi-1} a_{t,i} = 1 \quad \forall i, i \in [0, N-1]. \quad (2)$$

注意到 A 矩阵仅描述了在重复模式中每条指令的开始运行时刻的节拍点。为了确定非流水功能部件的需求,我们不仅要知道每条指令的启动时间,而且也需要知道每条指令要执行多长时间。

如果一条延迟(latency)为 d_i 的指令在重复模式中的 t 时刻的节拍点开始运行,它将一直执行到时间节拍点 $(t + d_i - 1) \bmod \Pi$ 结束。因为 FU 是非流水部件,在这一整段时间不会有其他指令使用该部件。我们可以根据仅描述每条指令起始点的模保留表(modulo reservation table) A 得出可以描述使用非流水 FU 的整个指令执行区间的矩阵 U 。

形式上 U 的计算如下:

$$u_{t,i} = \sum_{k=0}^{d_i-1} a_{(t-k) \bmod \Pi, i} \quad \forall t, t \in [0, \Pi - 1], i \in [0, N - 1], \quad (3)$$

如果所有指令的延迟都为 1 个节拍($d_i = 1$),则 $u_{t,i} = a_{t,i}$ 。根据式(2),则有 $u_{t,i} \in \{0, 1\}$ 。

所以,我们在计算节拍点 t 的资源需求时,必须对所有 i 的 $u_{t,i}$ 求和。又由于不同的指令可能使用不同类的功能部件,因此,要按不同类型的功能部件分别计算资源需求。我们用 $\zeta(r)$ 表示需要类型 r 功能部件的所有指令的集合,则对于一个合理的调度,资源需求必须满足以下限制:

$$\sum_{i \in \zeta(r)} u_{t,i} \leq R. \quad (4)$$

这里为叙述简单起见,我们只考虑降频时以资源类为单位,比如,要对整数操作进行降频处理,则所有的整数运算部件都降频运行,这时对资源使用的限制条件是式(3)完全一致的。如果进一步考虑同一类型部件中有的降频有的不降频或各自降频比不同时,可采用与文献[8]类似的资源限制来考虑,把同类中的资源再按不同的降频比划分为更细的子类,再按式(3)进行限制。

最后,在调度中通过以下约束来强调 DDG 中描述的依赖限制:

$$t_j - t_i \geq d_i - II \times m_{ij} \quad \forall i, j \in [0, N-1], \quad (5)$$

这里 d_i 是指令 i 的延迟, m_{ij} 是 i 到 j 的依赖距离(dependence distance). 进而, 为了保证每一条指令在重复模式中只调度一次, 对矩阵 A 除了要满足限制式(2)以外, 还要满足以下附加限制:

$$t_i \geq 0, k_i \geq 0, a_{t,i} \geq 0, u_{t,i} \geq 0, \text{且均为整数}; i \in [0, N-1], t \in [0, II-1]. \quad (6)$$

最终, 整个软件流水调度问题可归结为: 在式(1)~式(6)的限制条件之下, ILP 力求最小化各类功能部件的加权和. 若设有 h 类功能部件, 且与类型 r 的功能部件 FU 相关的权值为 C_r , 则它可以反映循环中所使用的功能部件的关键性. 可以用这种方法对 ILP 进行启发求解, 其目标函数为

$$\min \sum_{r=0}^{h-1} C_r \times R_r.$$

2.2 软件流水的低功耗优化调度

在进行软件流水的低功耗调度时, 主要是要考虑每条被降频执行的指令延迟变化, 受降频的影响, 各指令在重复模式中的起始执行时刻节拍会有所改变, 因此, 第 2.1 节中的有关计算也都会有相应的改变. 下面主要讨论: (1) 在什么样的条件下可以实现降频执行; (2) 降频后各指令延迟如何计算; (3) 各功能部件的降频比应如何计算; (4) 对相应的 ILP 计算如何进行调整等.

首先, 对

$$\sum_{i \in \zeta(r)} u_{t,i} < R_r, \quad t \in [0, II-1], \quad (7)$$

或

$$\left\lfloor \frac{T_r}{F_r} \right\rfloor < II \quad (8)$$

的指令集 $\zeta(r)$ 所使用的资源可以考虑进行降频运行, 因为这些资源是有空闲节拍存在的. 由于降频后的指令 i 的延迟, dd_i 相对于降频之前的延迟 d_i 是延长了. dd_i 可由以下计算得到:

$$dd_i = \left\lceil \frac{II}{\left\lfloor \frac{N_r}{R_r} \right\rfloor} \right\rceil, \quad i \in \zeta(r), \quad (9)$$

式中 N_r 为集合 $\zeta(r)$ 中的指令条数, 这里假定在同一类部件上执行的指令延迟相同. 对于不在降频部件上执行的指令的延迟仍然不变(即 $dd_i = d_i$), 且 r 类功能部件的降频比为

$$\delta_r = \frac{\sum_{i \in \zeta(r)} d_i}{\sum_{i \in \zeta(r)} dd_i}. \quad (10)$$

对 Γ 向量中每条指令 i 在第 1 次执行时的启动时间 t_i 重新计算如下:

$$t_i = \sum_{j=0}^{i-1} dd_j, \quad i \in [0, N-1]. \quad (11)$$

然后, 对同一循环程序, 以重新计算后的指令延迟和 Γ 为基础, 按第 2.1 节中相同的 ILP 求解方法进行同样软件流水调度过程, 并相应调整约束条件(3)和(5)为(3')和(5'):

$$u_{t,i} = \sum_{k=0}^{dd_i-1} a_{((t-k) \bmod II), i}, \quad \forall t \in [0, II-1], i \in [0, N-1] \quad (3')$$

和

$$t_j - t_i \geq dd_i - II \times m_{ij}, \quad \forall i, j \in [0, N-1]. \quad (5')$$

按照以上考虑降频低功耗优化条件的 ILP 计算方法, 可对表 2 中调度 B 的各矩阵进行新的计算和调整.

综上所述,我们可将以上思想和论述归结为以下编译实现过程:

- ① 对循环 L ,以式(1)~式(6)为约束条件,按 ILP 启发求解目标函数,得出合适的调度 A 和相应的 II.
- ② 根据条件(7)或(8)判定调度 A 是否可进行降频处理?
- ③ 如果调度 A 不可以进行降频处理,则结束.
- ④ 如果调度 A 可以进行降频处理,则根据式(9)~式(11)计算可降频部件的降频比 δ 、各指令的延迟 dd_i 和 I 向量.

⑤ 在第④步计算结果的基础上,以新的约束条件(即用(3')和(5')分别替换(3)和(5))和 II 重新产生调度 B.若成功,则以调度 B 为最终的软件流水调度;若不成功,则恢复调度 A 为最终的软件流水调度.

3 结束语

本文的主要工作是对具有可动态独立调整运行频率/电压的多功能部件配置结构 \mathcal{M} ,基于全局调度的循环依赖关系,使用 ILP 形式化框架,对给定循环 L 进行动态频率/电压调整的低功耗软件流水最优化调度的编译技术研究.提出了一种合理有效的低功耗最优化软件流水调度方法,使其在运行时保持 II 不变而消耗的功耗/能量最小.本研究的基本出发点是一种不降低性能的低功耗软件流水调度,而不是一种性能/功耗的权衡,但它又有很强的灵活性,如:

- (1) 若在判别一个调度是否可进行降频时,将条件(8)改为

$$\left[\frac{T_r}{F_r} \right] < II \cdot \theta, \theta \geq 1,$$

则可通过调整 θ 的值进行性能/功耗的权衡.

(2) 如果进一步考虑把同一类型部件划分为更细一级的子类,就可实现对同一类型的功能部件按不同的降频比进行调度和资源限制.

(3) 如果在为低功耗调度计算 dd_i 时,调整式(9)的计算方法和相应的调度策略,使得 r 类功能部件的指令尽量集中调度在部分部件上执行,多余的空闲部件利用动态 clock gating 支持功能将其关闭,以减少静态功耗.

(4) 若在低功耗软件流水调度时增加各功能部件资源保留表的条件约束,则可应用于具有冒险(hazard)的流水功能部件结构.

同时,程序的执行具有动态特性,只要不影响软件流水的优化,都可以按本方法进行相应的低功耗优化.但在对任何动态频率调整的低功耗优化时都要考虑到动态频率调整时硬件稳定所造成的时间开销^[5],以期达到整体效果的提高.

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,在此表示感谢.

References:

- [1] Altman ER, Govindarajan R, Gao GR. Scheduling and mapping software pipelining in the presence of structural hazards. In: Wall DW, ed. Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation. New York: ACM Press, 1995. 139~150.
- [2] Lam M. Software pipelining: An effective scheduling technique for VLIW machines. In: Wexelblat RL, ed. Proceedings of the SIGPLAN 1988 Conference on Programming Language Design and Implementation. New York: ACM Press, 1988. 318~328.
- [3] Ning Q, Gao GR. A novel framework of register allocation for software pipelining. In: Deussen MV, Lang B, eds. Proceedings of the Conference Recording of the 20th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM Press, 1993. 29~42.
- [4] Rau BR, Fisher JA. Instruction-Level parallel processing: History, overview and perspective. Journal of Supercomputing, 1993,7(1/2): 9~50.

- [5] Zhao RC, Tang ZM, Zhang ZQ, Gao GR. A multithreaded compiler optimization technology with low power. Journal of Software, 2002,13(6):1123~1129 (in Chinese with English abstract).
- [6] Govindarajan R, Altman ER, Gao GR. Minimizing register requirements under resource_constrained rate-optimal software pipelining. In: Mulder H, Farrens M, eds. Proceedings of the 27th Annual International Symposium on Microarchitecture. New York: ACM Press, 1994. 85~94.
- [7] Ruttenberg J, Gao GR, Stouchinin A, Lichtenstein W. Software pipelining show-down: Optimal vs. heuristic methods in a production compiler. In: Fischer CN, ed. Proceedings of the ACM SIGPLAN'96 Conference on Programming Language Design and Implementation. New York: ACM Press, 1996. 1~11.
- [8] Yang HB, Govindarajan R, Gao GR, Theobald KB. Power-Performance trade-offs for energy-efficient architectures: A quantitative study. In: International Conference on Computer Design (ICCD) 2002. 2002. <http://www.capsl.udel.edu/~hyang/personal.html>.

附中文参考文献:

- [5] 赵荣彩,唐志敏,张兆庆,Guang R Gao.低功耗多线程编译优化技术.软件学报,2002,13(6):1123~1129.

敬告作者

《软件学报》创刊以来,蒙国内外学术界厚爱,收到许多高质量的稿件,其中不少在发表后读者反映良好,认为本刊保持了较高的学术水平.但也有一些稿件因不符合本刊的要求而未能通过审稿.为了帮助广大作者尽快地把他们的优秀研究成果发表在我刊上,特此列举一些审稿过程中经常遇到的问题,请作者投稿时尽量予以避免,以利大作的发表.

1. 读书偶有所得,即匆忙成文,未曾注意该领域或该研究课题国内外近年来的发展情况,不引用和不比较最近文献中的同类结果,有的甚至完全不列参考文献.

2. 做了一个软件系统,详尽描述该系统的各个方面,如像工作报告,但采用的基本上是成熟技术,未与国内外同类系统比较,没有指出该系统在技术上哪几点比别人先进,为什么先进.一般来说,技术上没有创新的软件系统是没有发表价值的.

3. 提出一个新的算法,认为该算法优越,但既未从数学上证明比现有的其他算法好(例如降低复杂性),也没有用实验数据来进行对比,难以令人信服.

4. 提出一个大型软件系统的总体设想,但很粗糙,而且还没有(哪怕是部分的)实现,很难证明该设想是现实的、可行的、先进的.

5. 介绍一个现有的软件开发方法,或一个现有软件产品的结构(非作者本人开发,往往是引进的,或公司产品),甚至某一软件的使用方法.本刊不登载高级科普文章,不支持在论文中引进广告色彩.

6. 提出对软件开发或软件产业的某种观点,泛泛而论,技术含量少.本刊目前暂不开办软件论坛,只发表学术文章,但也欢迎材料丰富,反映现代软件理论或技术发展,并含有作者精辟见解的某一领域的综述文章.

7. 介绍作者做的把软件技术应用于某个领域的工作,但其中软件技术含量太少,甚至微不足道,大部分内容是其他专业领域的技术细节,这类文章宜改投其他专业刊物.

8. 其主要内容已经在其他正式学术刊物上或在正式出版物中发表过的文章,一稿多投的文章,经退稿后未作本质修改换名重投的文章.

本刊热情欢迎国内外科技界对《软件学报》踊跃投稿.为了和大家一起办好本刊,特提出以上各点敬告作者.并且欢迎广大作者和读者对本刊的各个方面,尤其是对论文的质量多多提出批评建议.