

线程划分中的寄存器依赖问题分析

李美蓉, 韩 博, 赵银亮, 赵永刚

(西安交通大学 计算机科学与技术系, 陕西 西安 710049)

摘 要: 推测多线程主要针对编译器生成的指令进行线程划分, 在控制流和数据流分析基础上, 实现串行程序的自动并行化。模拟器作为检验线程划分算法的有效手段, 不仅能验证程序执行结果的正确性, 而且可以评估程序并发执行的加速比性能, 进一步也可以反映线程划分算法的合理性。针对 Olden Suite 程序在模拟器上的运行时统计信息, 分析线程划分中所存在的寄存器依赖问题。同时, 结合实例详细讨论造成寄存器依赖的主要原因。最后, 针对寄存器依赖问题提出一种改进的线程划分方法。

关键词: 自动并行化; 线程划分算法; 寄存器依赖; 模拟器

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2011)03-0006-07

The Issue of Register Dependency Analysis in the Thread Partitioning

LI Mei-rong, HAN Bo, ZHAO Yin-liang, ZHAO Yong-gang

(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: Speculative Multithreading (SpMT) mainly processes thread partitioning on the instructions generated by the compiler, and realizes the whole program auto-parallelization by using the CFG and DFA techniques. Simulator as an available means of checking thread partitioning algorithm, not only does verify the correctness of the partitioned program, but also can assess the program's speedup performance in the concurrent execution. What's more, it may reflect the effectiveness of thread partitioning algorithm. This paper focuses on the Olden Suite statistical information at runtime, and analyzes the issue of register dependency that exists in the thread partitioning. At the same time, we take an example to discuss the main reasons to cause the register dependencies in detail. At last, we propose an improved thread partitioning algorithm strategy for solving the register dependency issue.

Key words: auto-parallelization; thread partitioning algorithm; register dependency; simulator

1 引言

线程划分主要以程序执行的加速比性能为衡量标准。通过在模拟器上执行编译器端产生的线程划分结果, 不仅能够反映程序并发执行的加速比性能, 而且在分析模拟器运行时的日志信息基础上, 可以进一步评估线程划分算法的合理性。

目前, 国内外在串行程序的自动并行化研究中, 已经取得了相当的成果。Hans Vandierendonck 等^[1]提出针对循环域分解有序代码片段的线程划分算法, 通过将循环域中的有序代码抽取出来, 转换为

基于上下文环境的子任务, 实现程序的自动并行化。Sean Rul 等^[2]提出基于 profile 信息的线程划分算法。在控制流和数据流依赖分析的基础上, 允许并发线程存在一定的数据依赖关系, 并通过硬件辅助检测和撤销发生依赖的线程。Carlos Madriles 等^[3]采用硬件机制通过检测线程违规, 改造程序代码序列, 将程序分解为多个并行执行的子线程, 从而改善单个线程在多核上执行性能。Zhao-Hui Du 等^[4]提出使用代价模型评估循环域的线程划分, 通过计算循环迭代中错误推测的代价, 选择合理的划分方式, 实现提高程序并发执行的性能。本文分析和研究的线

程划分算法基础是 Chen Zhen 等^[5]提出的基于预计算片段的循环域和非循环域的线程划分算法. 其主要实现方法,在控制流和数据流分析的基础上,利用 pslice(预计算片段)程序切片技术,对 profile(程序剖析)信息反映的最有可能路径进行线程划分. 同时,线程划分算法所依赖的模拟器和执行模型是 Dong zhaoyu 等^[6]提出的基于 CMP 的支持推测多线程的执行模型的 Prophet 框架.

本文主要针对模拟器的运行时统计日志信息分析,采用典型实例进一步分析线程划分中存在的寄存器依赖问题. 在分析基础上,提出采用过程间数据流分析技术,寻找过程内满足并发执行的被调过程的思想,从而实现在细粒度线程划分的基础上,有效改善线程划分算法,解决寄存器依赖问题.

2 寄存器依赖在模拟器上的结果分析

父线程在激发子线程后,需要将自己的寄存器值拷贝给子线程,子线程只有利用父线程所传递的寄存器值,才能进行预计算处理以及推测执行. 若父线程从子线程的激发点到验证过程中,修改了某个寄存器的值,那么为了保证执行模型正确性,父线程需要将更新后的寄存器值传递给后继线程. 在此之前,如果后继线程读取或更新了这个寄存器的值,那么就发生寄存器依赖违规,后继线程需要撤销重启^[6].

通过在模拟器上运行 Olden 测试程序的线程划分结果,可以获得程序在推测执行过程中的运行时信息,如执行加速比,激发线程总数,失败线程总数,造成线程失败的数据依赖、寄存器依赖、控制流错误以及程序异常等信息. 这些信息通常在线程划分时难以获得,但却对分析线程划分算法有着重要的作用.

针对当前的线程划分算法,在模拟器上的运行结果如图 1 所示,图中显示了线程划分后的 Olden 测试程序在模拟器上运行时的加速比和线程的发起成功率信息. 其中,加速比平均达到 1.5 以上,线程发起的成功率达到 60% 以上. 但是,从某种程度上来说,线程发起失败率还是占相当的比例. 分析引发线程失败的原因,仍有十分重要意义.

为了更好地分析引发线程失败的原因,需要综合分析模拟器运行时统计的日志信息. 图 2 显示了模拟器统计的线程发起失败的五种原因:数据依赖验证错误 (TOnDDV), 寄存器依赖验证 (TOnRegV), 发起线程低优先级撤销 (TOnLowPriority), 程序控制依赖推测错误 (TOnControl) 和程序异常 (TOnException). 从图中显示的各种引发因素所占比例可以看出,寄存器依赖是引发线程失败的最主要原因,其次是线程推测级优先问题.

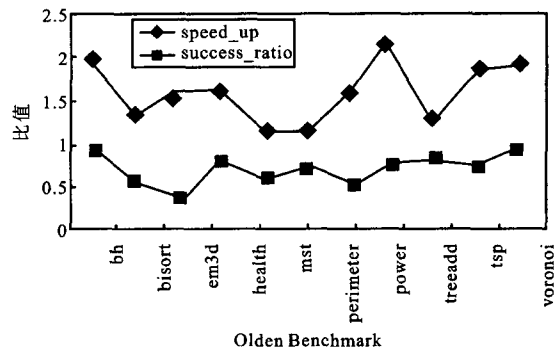


图1 程序执行加速比和线程激发的成功率

ty), 程序控制依赖推测错误 (TOnControl) 和程序异常 (TOnException). 从图中显示的各种引发因素所占比例可以看出,寄存器依赖是引发线程失败的最主要原因,其次是线程推测级优先问题.

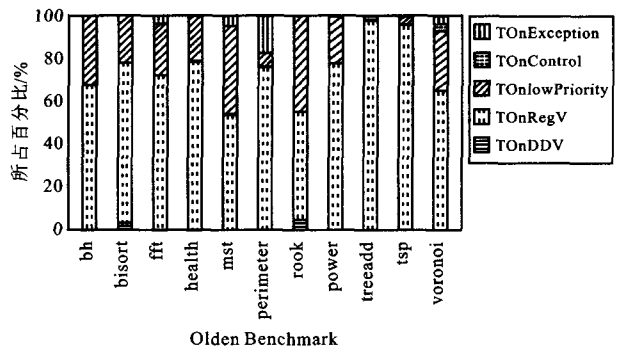


图2 失败的线程的各种情况分析

图 3 中详细给出模拟器上测试程序运行时的寄存器依赖问题,从图中可以看出,寄存器依赖较为集中的是 \$2, \$4, \$5 等寄存器(MIPS 体系结构寄存器). 为了清晰地给出不同寄存器间发生依赖的频率统计信息,图 4 中从模拟器的统计信息,计算出每个寄存器的依赖信息统计. 这样可以更直观地看出发生依赖比较集中的寄存器.

从 MIPS 的寄存器约定可知, \$2 用于过程调用返回值寄存器 \$v0; \$4, \$5 分别对应于过程调用时的参数寄存器 \$a0, \$a1; \$25 是临时寄存器 \$t9; \$29 是栈指针寄存器 \$sp.

模拟器的运行时统计信息表明 \$2, \$4 寄存器的依赖最为严重. 也就是说,子线程过程调用时相互的参数传递导致了严重的寄存器依赖问题. 同时,在统计信息中,也显示出占有相当比例的过程调用返回值依赖问题.

本文采用基于编译后端目标代码的线程划分算法^[5]. 算法将串行程序划分为循环域和非循环域两个部分. 首先,根据编译前端收集的 profile 信息,找出程序的最有可能执行路径. 然后,在最有可能路径

上对循环域进行线程划分,利用所构造的超级块技术将循环域归结为基本块,进行非循环域的线程划分.最终,构成整个程序的划分.每一个被划分的子线程用<spawn,cqip>点对标记,spawn 代表子线程激发点,cqip 既是子线程的结束点又是新的子线

程的执行起始点.在子线程推测执行(cqip 点之后)时,采用程序切片技术,构造<pslice.entry,pslice.exit>点对用于计算子线程中的活跃变量的值,整个推测执行过程的正确性,由模拟器上的执行模型保证.

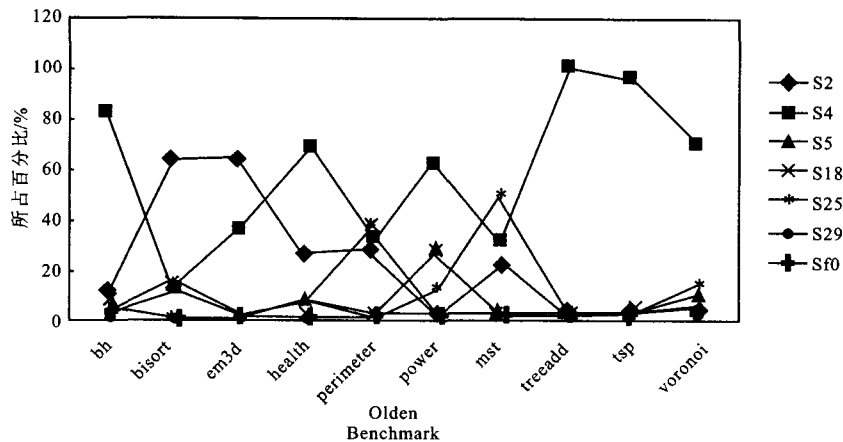


图 3 Olden Suite 程序集在各个寄存器上的依赖分析

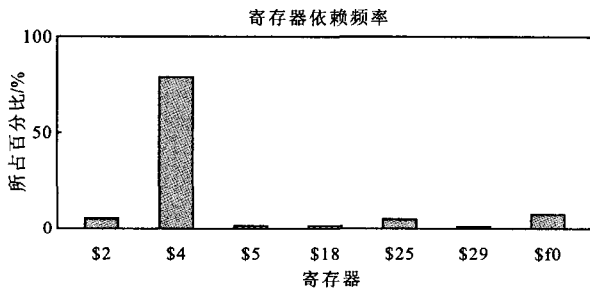


图 4 寄存器发生依赖的频率

3 tsp 测试程序的寄存器依赖分析

基于线程划分算法和寄存器依赖的定义,本节针对测试程序 tsp 的线程划分结果考虑寄存器依赖问题.图 4 中显示 tsp 程序的 \$4 寄存器依赖所占比例最大,因此,将它作为分析寄存器依赖的典型例子.

图 5 所示为 tsp 程序的 main 函数代码,经过编译过程和线程划分后,所生成的 MIPS 汇编代码如图 6 所示.在图 6 中,<spawn,cqip>点对(带有下划线部分)代表激发的一个线程体.其中,spawn 为线程的激发点,cqip 点为线程执行部分.<pslice_entry,pslice_exit>点对为激发线程构建的预计算片段标志,用于线程划分中计算所激发子线程的活跃变量的值,一般插入到 cqip 点之后,推测执行指令之前.斜体部分是过程调用指令.

在图 6 中,父线程 entry.headerspawn 激发子线程 entry.L6 后,继续执行了 dealwithargs 函数,此时,entry.L6 线程在执行指令 move \$s0, \$v0

```
int main(int argc,char*argv[])
{
    Treet;
    int num;
    num=dealwithargs(argc,argv);
    printf("Building tree of size %d\n",num);
    t=build_tree(num,0,0,NumNodes,0.0,1.0,0.0,1.0)
    if(!flag)printf("Past build\n");
    if(flag)printf("newgraph\n");
    if(flag)printf("newcurve pts\n");
    printf("Call tsp(t,%d,%d)\n",150,NumNodes);
    tsp(t,150,NumNodes);
    if(flag)print_list(t);
    if(flag)printf("linetype solid\n");
}
```

图 5 tsp 测试程序的 main 函数

时,读取了 entry.headerspawn 的 dealwithargs 的函数调用返回值.因此,因没有预测新的返回值而使后继线程在执行过程中,读取错误的 entry.headerargs 因与 entry.L6 并发执行没有完成 dealwithargs 函数调用,使后继线程 entry.L6 不能获得正确的返回值 \$v0,从而导致线程 entry.L6 推测失败而撤销.

同时,父线程 entry.L7 在激发子线程 entry.L8 后,假定当前处理器的数目足够多,父线程 entry.L7 拷贝自己的寄存器值传递给后继线程 entry.L8,然后将 entry.L8 分配到一个空闲核上推测执行.在 entry.L7 和 entry.L8 并行执行的过程中,\$a0(即 \$4)寄存器因 lw \$a0,flag 被修改,此时,父线程 entry.L7 将 \$a0 的值传递给后继线程 entry.L8,但

综上所述,本文以改进线程划分算法为目标,采用过程间数据流分析技术,提前激发彼此没有数据依赖关系的被调过程,使寄存器依赖问题得到解决。Aleen Farhana 等^[11]提出抽取程序中的代码片段,即所谓的 commutative functions,它在任意执行顺序下不影响程序的最终执行结果,实现了程序的自动并行化。借鉴此研究基础上,本文提出一种改进的线程划分方法。在线程划分使用过程内数据流分析结果的基础上,对被调过程展开分析,将每一个被调过程对全局变量和主调过程变量的读写作用记录下来。然后,结合前端 profile 信息提取不受过程调用约束的变量信息。最后,在综合过程内数据依赖分析的基础上,将有返回值依赖关系的过程进行顺序激发,而对于条件控制的过程,根据过程间数据流分析的基础,实现提前激发,尽量采取以每个过程调用为一个线程体的细粒度线程划分策略。

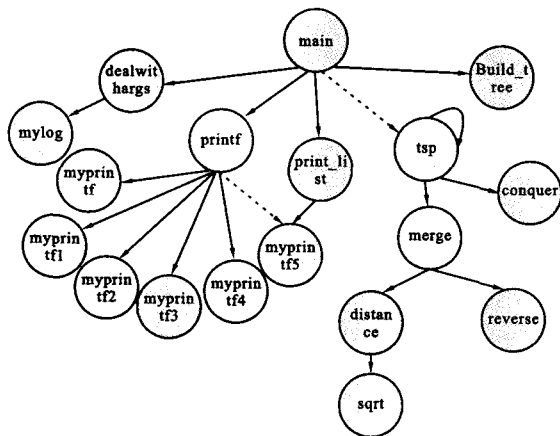


图 7 main 函数过程调用图

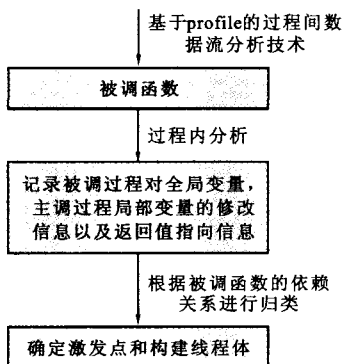


图 8 调用过程的数据流分析

一般情况下,被调过程在进行线程划分时,往往因其不可预测的执行时间而被忽略。但被调过程常常是使一个串行程序顺序执行的关键环节。如过程调用的返回值信息通常作为下一条指令的输入信息。因此,尽量提前激发这些被调过程,能有效地提

高程序执行的加速比性能。对程序进行过程间分析,需要构建程序的过程调用图,图 7 为 tsp 程序的过程调用图。过程间分析指在分析程序间的调用关系的基础上实现过程间的数据流分析技术。

采用过程间的数据流分析,其设计方案如图 8 所示。本文采用基于 profile 信息的过程间数据流分析技术对被调过程进行数据流分析。首先,考虑被调过程的调用参数,分析参数的来源,即是否是前面调用函数的返回值,以及能否从其他被调过程中提取出来。其次,对被调过程进行过程内的数据流分析,记录参数的修改信息以及全局变量的使用情况。再者,将获得的全部过程调用分析结果根据控制和数据依赖关系,进行归类。最后,结合循环域和非循环域的线程划分,给出一个合理的被调过程乱序激发方案,插入合适的 spawn, cqip 激发点对。

对于部分实际存在返回值依赖的前驱和后继关系的调用过程,如图 9 中的 build_tree 和 tsp 被调过程,由于 build_tree 函数返回值作为 tsp 过程调用的输入参数。因此,将采取顺序激发的策略实现。在分析过程中,基于 profile 信息的最有可能路径划分会产生推测错误的情况。此时,依据所设计的执行模型,在推测过程中出现推测失败时,执行模型将采取一定的措施对后继线程进行撤销和重启,特别对于被调过程的乱序激发,需要执行模型对已经提前执行的过程进行逻辑排序和验证等。特别地,在推测失败时,能及时撤销和改正程序的执行,从而保证程序执行的正确性^[6]。

下面仍以 tsp 程序的 main 函数为例,采用增加过程调用数据流分析的技术实现线程划分。图 7 为 tsp 程序的过程调用图,为了细粒度实现过程调用的分析,这里,将 printf 系统函数按照具有输出形式

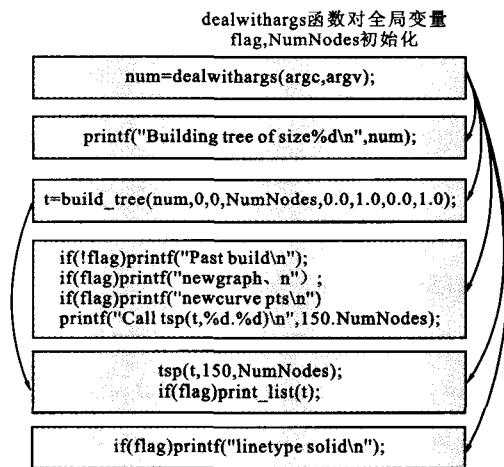


图 9 tsp 程序 main 函数中过程间变量依赖关系

划分为 printf, printf1, printf2, printf3, printf4, printf5 等. 按照所设计的过程间数据流分析的方法, 对 tsp 程序的所有被调过程进行数据流分析. 最后, 从图 9 可以看到, dealwithargs 函数实现全局变量 flag 和 NumNodes 的初始化. 而 printf 函数直接引用 dealwithargs 的过程调用返回值作为输入参数, 两者之间存在直接的前驱和后继关系. build_tree, tsp 函数则间接引用 dealwithargs 函数中初始化的全局变量信息. 它们是间接的依赖关系, 但是这种关系可以通过过程间的分析得到解决. 这里, 通过提取 dealwithargs 过程内关于得到全局变量 flag 和 NumNodes 的指令切片程序^[12], 获得最有可能的执行结果. 同样地, 程序中受全局变量 flag 控制执行的被调过程也可以在提取 flag 值的情况下, 根据 profile 计算的分支概率信息, 在 dealwithargs 过程执行前, 实现乱序激发. 而对于受返回值依赖的被调过程, 如 dealwithargs 和紧跟其后的 printf 函数, build_tree 和 tsp, print_list 函数, 此时, 采用顺序激发策略. 程序的执行的正确性由模拟器上的执行模型来保证.

通过考虑过程调用函数对主调过程的影响作用, 采用数据流分析技术实现过程间的数据依赖分析, 改进后的线程划分算法使 flag 条件控制的输出函数和 build_tree 函数在 dealwitharg 函数执行之前得到激发, 由于同时激发的过程调用间彼此不存在寄存器依赖关系, 含有过程调用的激发线程通常采用在过程调用点后激发子线程, 如存在返回值依赖的 build_tree 和 tsp 函数, 即顺序激发的策略, 从而有效避免了过程返回值寄存器的依赖情况. 为了能更好地解决参数寄存器依赖问题, 需要在编译器后端寄存器分配阶段, 结合线程划分的思想改进寄存器分配算法, 使指令序列中的虚拟寄存器和临时寄存器对应到合理的空闲参数和返回值寄存器. 按照所设计的过程调用数据流分析方法在线程划分算法中实现后, 程序的预期划分效果如图 10 所示. 从图中可以看到, 此时激发的线程更多, 每个线程体指令数更少, 是细粒度的线程划分方法.

5 结束语

本文通过对模拟器的运行时日志统计信息分析, 探讨导致线程发起失败的主要因素, 即线程间的寄存器依赖问题. 本文对寄存器依赖问题进行了讨论和实例分析, 观察到当前线程划分算法的不足之处. 在此基础上, 提出采用过程间数据流分析技术对

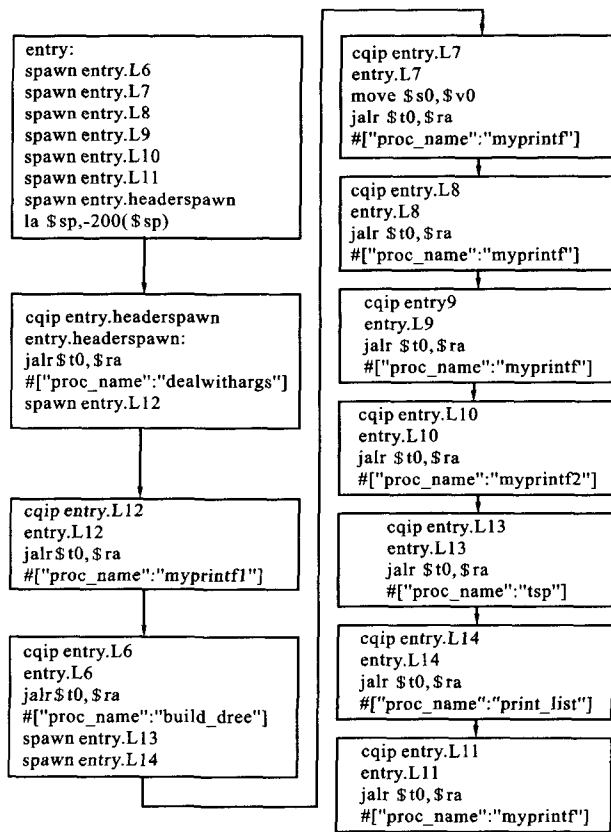


图 10 基于过程调用分析的线程划分结果

过程内的所有被调过程进行数据依赖分析, 并将所有被调过程按照数据依赖程度分类, 根据合适的激发策略, 实现程序的自动并行化. 最后, 本文结合实例分析, 给出改进的线程划分方案的预期划分效果.

参考文献:

- [1] Vandierendonck Hans, Rul Sean, DeBosschere Koen. Factoring out ordered sections to expose thread-level parallelism[C]// The 2nd Workshop on Parallel Execution of Sequential Programs on Multi-core Architecture. Austin, Tx, USA; PESPMA, 2009; 12-19.
- [2] Sean Rul, Hans Vandierendonck, Koen DeBosschere. Function level parallelism driven by data dependencies [J]. ACM SIGARCH Computer Architecture News, 2007, 35(3): 55-62.
- [3] Madriles Carlos, López Pedro, Codina JosepM, et al. Boosting single-thread performance in multi-core systems through fine-grain multi-threading [C]// Proceedings of the 36th Annual International Symposium on Computer Architecture. New York: ACM, 2009, 37(3): 474-483.
- [4] Du Zhaohui, Yang Chen, Lim ChuCheow, et al. Acost-driven compilation framework for speculative parallelization of sequential programs [J]. ACM SIGPLAN

- Notice, 2004,39(6):71—81.
- [5] Chen Zheng, Zhao Yinliang, Pan Xiaoyu, et al. An overview of prophet[C]// Porceedings of the International Conference on Algorithms and Architectures for Parallel Processing. Berlin: Computer Science, 2009: 396—407.
- [6] Dong Zhaoyu, ZhaoYinliang, Wei Yuanke, et al. Prophet: A speculative multi—threading execution model with architectural support based on CMP[C]// InternationalConference on Scalable Computing and Communications—The 8th International Conference on Embedded Computing. Dalian: ScalCom-EmbeddedCom, 2009, 2009:103—108.
- [7] Taku O, Masamichi T, Shoji K, et al. Pinot: Speculative multi—threading processor architecture exploiting parallelism over a wide range of granularities[C]// Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture. Washington, DC, OSA: IEEE Computer Society, 2005:81—92.
- [8] Tullsen Dean M, Senq John S. Storageless value prediction using prior register values[C]// Conference Proceedings—Annual International Symposium on Computer Architecture Atlanta, GA, USA: ISCA, 1999: 270—279.
- [9] Lo Jack L, Parekh Sujay S, Eqgers Susan J, et al. Software— directed register deallocation for simultaneous multithreaded processors[J]. IEEE Transations on parallel and Distributed systems, 2002(8):922—933.
- [10] Lin Kelvin, Shann Jean Jyh— Jiun, ChungPing. Code compression by register operand dependency. Journal of Systems and Software[J]. 2004,72(3):295—304.
- [11] Aleen Farhana, Clark Nathan. Commutativity analysis for software parallelization: Letting program transformations see the big picture. ACM Sigplan Notices[C]// Proceeding of the 14th international conference on Architectural support for programming languages and operating systems. New Yrok: ACM, 2009, 44(3): 241—252.
- [12] Gallagher K, Binkley D. Program slicing[C]// Frontiers of Software Maintenance. Beijing: Inst of Elec and Elec Eng Computer Society, 2008: 58—67.

作者简介:

李美蓉 女,(1984—),博士研究生.研究方向为编译技术、推测多线程技术的线程划分算法和数据流分析.

(上接第 5 页)

- [5] Munoz Hernandez G A, Diaz Sanchez A, Jones D I, et al. Estimating the frequency variation of the mexican grid by Kalman filtering [C]// IEEE International Conference on Electrical, Communications, and Computers. Washington: IEEE, 2009: 245—249.
- [6] Zhang Z G, Chan S C, Tsui K M. A recursive frequency estimator using linear prediction and a kalman—filter—based iterative algorithm [J]. IEEE Transaction on Circuits and Systems, 2008, 55(6): 576—580.

作者简介:

孟 真 男,(1983—),博士生.研究方向为射频电路设计、系统设计与信号处理.

阎跃鹏 男,(1964—),研究员.研究方向为卫星导航通讯、射频微波技术与电子系统总体设计.