计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(7):167−172 [doi: 10.15888/j.cnki.csa.005834]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于BWDSP的字符串与内存处理函数优化①

张仁高[1], 郑启龙[1], 王向前[2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国电子科技集团公司第三十八研究所, 合肥 230088)

摘　要: 面向BWDSP的体系结构分析了字符串与内存处理函数汇编优化方法, 基于向量化与软件流水的优化技术, 通过利用高效访存指令、能够提升循环执行效率的零开销循环机制、指令重排技术, 结合具体功能函数的循环特性, 展开针对字符串与内存处理函数的指令级并行性挖掘. 实验结果表明, 这些库函数的优化效率能够达到硬件平台提供函数性能理论运行时间的1.5倍以下,对BWDSP平台整体性能提升具有重要意义.

关键词: 字符串与内存处理函数; BWDSP; 函数优化; 向量化与软件流水; 特殊指令; 并行性

引用格式: 张仁高,郑启龙,王向前.基于BWDSP的字符串与内存处理函数优化.计算机系统应用,2017,26(7):167–172. http://www.c-s-a.org.cn/1003-3254/5834.html

Optimization of String and Memory Functions Based on BWDSP

ZHANG Ren-Gao[1], ZHENG Qi-Long[1], WANG Xiang-Qian[2]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](No.38 Research Institute, China Electronics Technology Group Corporation, Hefei 230088, China)

Abstract: For architecture of BWDSP, this paper analyzes the implementation of the optimizations, based on vectorization and software pipelining of parallel optimization technique, including special memory access instruction, zero-overhead looping instruction of improving efficiency in the loop, Instruction-level Parallelism(ILP), combined with the specific function of loop characteristics, expansion for the string and memory function of instruction level parallel mining. The experimental results show that the optimization rates of most functions of the theory have a running time of 1.5 times on hardware platform, which is of great importance to enhance the platform performance of BWDSP.

Key words: string and memory functions; BWDSP; optimization of function; vectorization and software pipelining; special instruction; parallelism

1　引言

字符串与内存处理函数是最基本的库函数, 它对上层函数的调用与处理提供了强大的接口调用. 这些函数使用较为频繁, 经编译系统直接生成的代码性能并不高, 有必要从汇编层面优化其性能. 本文主要对字符串与内存处理函数进行汇编优化, 提高其的执行效率.

2　BWDSP介绍

BWDSP是一款16发射, 7级流水线, 4簇结构的处理器. 四个簇分别标号为X, Y, Z, T. 每个簇有64个寄存器, 4个乘法器, 8个ALU和4个特殊功能单元. 簇与簇之间采用簇间传输总线通信. BWDSP还采用了常见的AGU结构用来加速访存, 内存访问必须经过AGU单元. AGU(Address Generation Unit)是用作访存地址计算特殊单元, BWDSP的load/store的地址操作数必须使用AGU中地址寄存器. 使用独立的AGU计算, 表示访存地址可以降低处理器设计的复杂度. BWDSP的体系结构支持多种访存模式, 支持零开销循环、向量化执行

① 基金项目: 国家核高基重大专项(2012ZX01034001-001)
收稿时间: 2016-11-19; 收到修改稿时间: 2016-11-24

© 中国科学院软件研究所 http://www.c-s-a.org.cn

等, 其主要体系结构[1]如图1所示.

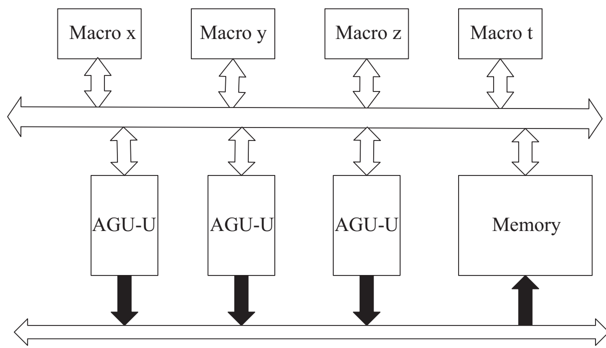

图1 BWDSP主要体系结构

2.1 BWDSP SIMD指令的特性分析

BWDSP计算指令的通用格式如下[1]:

[Macro]Rm = Rn op Rs

Macro是执行宏的代号, op是操作, 符号“||”连接多个可并行指令, 比如: xRm=Rn+Rs是Scalar指令, 表示在X宏上执行整数加法操作, xyztRm=Rn+Rs是SIMD指令, 表示在XYZT四个宏上同事执行整数加法操作, 等于xRm=Rn+Rs||yRm=Rn+Rs||zRm=Rn+Rs||tRm=Rn+Rs.

2.2 BWDSP SIMD寄存器分配策略

BWDSP的编译器OCC是以Open64为基础开发的, Open64编译器的寄存器分配分为两类: 全局寄存器分配(GRA)和局部寄存器分配(LRA). 全局寄存器分配在一个函数范围内, 为活跃范围超出一个基本块的LR分配寄存器; 局部寄存器分配在一个基本范围内, 为只活跃在基本块内部的LR分配寄存器.

2.3 通用DSP优化策略

使用循环计数器是通用DSP软件流水代码的优化方法. 最有效的软件流水循环一般按递减形式对循环进行计数. 通常即使源代码中没有按这种形式编写, 编译器也能转换成递减形式.

DSP的指令流水线存在着不可避免的阻塞现象, MAC单元指令也一样. 尽管在硬件设计时已经采用了专用模块来减少阻塞, 但有些阻塞是不可避免的, 从程序优化的角度来说, 可以充分利用指令流水线阻塞现象, 通过重排指令流水线上的指令, 消除阻塞, 以使得程序的运行时间缩短, 从而达到优化的目的.

3 BWDSP优化分析

3.1 特殊指令

双字寻址指令和sigma指令.

双字寻址指令功能是在1个时钟周期内读取多个字, 进一步减少函数的访存次数[2]. 指令形式如下:

{x, y, z, t}Rs+1, s = [Un+=Um, Uk]

[Un+=Um, Uk] = {x, y, z, t}Rs+1, s

第一条指令是读访存指令, Un是基地址, Uk是在Un地址基础上的调整量, Um则是指令执行后基地址Un的修正量. 指令采用双字节读数, 故偏移地址为[Un]和[Un+1]、[Un+2Uk]和[Un+2Uk+1]、[Un+2×2Uk]和[Un+2×2Uk+1]、[Un+3×2Uk]和[Un+3×2Uk+1], 将这8个地址的数据依次送到4个宏里的同名寄存器{x, y, z, t}Rs:s+1. 第二条指令是写存储指令.

sigma指令的功能是归并四个簇上的值, 指令格式如下[1]:

{x’, y’, z’, t’}Rs = sigma{x, y, z, t}Rm

将{x, y, z, t}Rm中32位定点有符号数分别相加到{x’, y’, z’, t’}Rs寄存器上. 通过sigma指令可以对分散在不同寄存器中值进行收集, 减少了逐一合并计算的时间.

零开销循环指令.

零开销循环[2]是通过硬件实现一种加速循环执行的方法. BWDSP硬件提供零开销循环指令, 其指令集中LC0和LC1指令专门用于零开销循环, 指令形式为“if LCx B label”. 程序的循环往往需要大量的条件判断和跳转. 普通跳转指令需要判断比较, 同时要对于控制条件进行累加, 这样的跳转指令判断需要占用多个周期. 零开销循环指令相比于普通跳转指令而言只需占用1个时钟周期. 以零开销循环控制循环体的长度, 可使优化后的汇编函数性能可以得到显著提升.

3.2 指令重排技术

BWDSP可同时支持执行16条指令, 即一个执行拥有十六个执行槽. 通过指令重排技术调节指令次序来减少流水线的空转和等待时间. 通过手动优化, 避免BWDSP中16路发射和乱序执行的特性使得指令调度变得复杂. 若经过指令重排无法将每个执行16个指令槽填满, 根据BWDSP 16路特性, 可以通过简单的使用空操作填充指令槽, 这样使得流水线按照较为理想的状态发射指令[3].

3.3 软件流水技术

软件流水是一种强大的循环调度技术, 允许指令跨迭代的移动实现循环迭代的重叠执行, 通过发掘循

© 中国科学院软件研究所 http://www.c-s-a.org.cn

环的不同迭代的各部分之间的指令间并行性, 使这些指令并行地执行. BWDSP体系架构上的软件流水, 采用模调度算法[4,5]和模变量扩展算法[6]思想实现软件流水汇编代码.

3.3.1 模调度算法

模调度是软件流水调度的核心算法, 它获取核心循环体基本块信息, 结合目标体系硬件资源生成模资源约束表MRT并计算出资源限制(*ResII*), 且根据指令相关依赖图计算出依赖限制(*RecII*), 模调度的最小启动*II*从开始进行调度, 如果调度不成功则增加*II*再次进行循环调度, 直到找到一个满足资源和依赖约束的合适*II*[7,8]. 启动间距*II*是衡量软件流水效率的重要指标; 展开因子是影响循环展开(loop unrolling)的主要因素. 资源限制(*ResMII*)和依赖限制(*RecMII*)是决定启动间隔*II*和展开因子[9,10]的主要因素. 其中*ResMII*的计算是先把一个循环体对应需要的资源需求列表计算出来, 通过简单的累加得到; 然后把所需的每种资源数量除以体系结构提供的资源数量, 该值最大就作为*ResMII*. *RecMII*则是依赖图形成各个环中延迟除以环跨越迭代距离的最大值[10]. 模调度算法的优点是以固定的间隔启动每个迭代的调度, 有助于减少寄存器的压力.

3.3.2 模变量扩展

在软件流水执行过程中, 不同循环体重叠地执行. 每次迭代中对同一个变量用相同寄存器变量存放, 则迭代之间就会相互有影响, BWDSP并不支持IA64体系结构下的旋转寄存器, 模变量扩展通过识别那些在每次迭代开始处重新定义的变量, 利用寄存器重命名技术, 使得每次迭代对同一变量的引用指向一个不同的位置, 彼此之间不冲突.

3.4 性能分析

BWDSP汇编函数库主要采用循环展开和软件流水的优化方案. 只有当循环体可向量化时, 才能运行循环展开和软件流水操作.

循环展开能够增加参与运算的个数, 减少循环的重复次数. 展开后的循环体包含更多的指令, 可以使调度部件更加自由地挑选不相关的指令并行执行.

3.4.1 资源限制

资源限制(*ResMII*)考虑了体系结构中可用资源的数量, 是衡量被硬件所约束的模调度的重要指标. 受体系结构中可用硬件资源数量的约束, 在计算理论运行时间是需要考虑循环体内各种运算操作在各类运算部件上的分配情况.

具体计算方法:

统计循环体内所有运算(取数, 加法, 移位, 超算, 写等)的操作步数, 标记为*N*. 假设取数操作有*N*1步, 加法操作有*N*2步, 乘法操作有*N*3步, 移位操作有*N*4步, 超算操作有*N*5步, 写操作有*N*6步, 且划分共有:

$$N = N1 + N2 + N3 + N4 + N5 + N6$$

划分各类操作至相应的操作部件, 即加法操作分配至加法器, 乘法操作分配至乘法器等.

例如:

在BWDSP104x中共存在16个slot(一般情况下N步操作分配于12个slot中), 8个ALU, 8个MUL, 4个SHIFT, 1个SPU, 3次读写.

则资源限制的最小理论周期:

$$max1 = Max(N/12, N1/2, N2/8, N3/8, N4/4, N5/1, N6/1)$$

3.4.2 依赖限制

依赖限制(*RecMII*)是衡量体系结构被数据依赖的重复回路所约束的模调度的重要指令. *RecMII*限制了软件流水循环体的最小规模, 如果小于这个值, 则会存在数据覆盖从而导致流水无法完成, *RecMII*被数据相关图中的重复回路所约束.

具体计算方式:

根据步长约束计算循环展开的次数*p*1, 计算循环体内*RecMII*的最小长度*s*. 根据寄存器约束计算循环展开的次数*p*2

计算理论上的循环展开次数*p*为:

$$p = Min(p1, p2)$$

计算依赖限制理论运行时间最小周期:

$$max2 = s * 3/P$$

3.4.3 计算理论运行时间

资源限制和依赖限制共同确定了指令的最小规模. 在依赖限制的流水计算中, 还需要考虑寄存器的上限, 即是否会因为循环体过大而产生寄存器不够用的问题.

假设输入向量长度的长度为*num*的函数, 要得到可向量化时汇编语言版库函数的理论运行时间, 还需综合考虑资源限制和依赖限制, 计算在完全流水条件下完成一组循环体内所有操作的最小理论运行时间*R_cycle*为:

$$R_cycle = Max(max1, max2)$$

以及统计循环体外执行串行操作的时钟周期数*C*.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

计算汇编语言版函数理论运行时间L_cycle为:

$$L_cycle = C + R_cycle * num$$

对于每个可向量化的函数, 当数据规模达到一定程度时, 理应满足运行时间小于汇编版函数理论运行时间的1.5倍.

4 字符串与内存处理函数

在库函数中字符串与内存处理函数存在大量的循环体与向量化, 是主要的优化对象, 其主要可以分为以下几种: 内存拷贝和赋值函数, 字符串拷贝与连接函数, 字符串与内存比较函数, 字符串与内存查找函数. 字符串与内存的操作通常一一对应, 区别在于字符串用空字符NULL来表示结尾, 而内存块则明确给出长度[11].

4.1 内存拷贝与赋值函数

memcpy, memmove及memset函数的基本实现流程为, 首先判断拷贝的数据大小是否符合优化的范围, 逐一拷贝源地址的字符到目的地址.

BWDSP为保证在源地址与目的地址的字符交叠时能够正确的复制, memmove在拷贝过程中需添加一个临时拷贝区域. Memcpy直接从源地址内存进行拷贝. memset为目的地址内存赋予特定值.

以memcpy函数为例, 函数功能是复制目的地址的字符到源地址, 通过平台支持的双字寻址指令中的读取指令读取目的地址的字符, 再利用存储指令存储读取的字符到源地址, 并且通过零开销循环指控制程序循环长度.

memcpy向量化的汇编代码如下所示:

```
1 _kernel:
2    r7:6=[v1+=8, 1]
3    || xr1=r1+1
4    nop
5    nop
6    if xr1!=r13 b _kernel
7    || [u0+=8, 1]=r7:6
```

由计算结果可得, 该循环体在一个迭代的过程中, 所需资源为2个slot(2/8), 1个branch(1/1), memread为(1/2), memwrite为(1/2). 访存资源利用率在一个迭代4个周期中的利用率为50%, 资源ALU利用率为25%.

汇编代码的性能仍然有提高效率和优化的空间, 可通过软件流水填补循环体中的指令之间的空缺周期方式进行优化, 使指令得到重排, 不同流水线可以同时执行, 避免了空闲周期的存在, 有效地提升了循环体性能. 软件流水汇编核心代码如下所示:

```
1 _kernel:
2    if nlc0 b _exit
3    || [u0+=8, 1]=r7:6
4    || r7:6=[v1+=8, 1]
5    if nlc0 b _exit
6    || [u0+=8, 1]=r17:16
7    || r17:16=[v1+=8, 1]
8    if lc0 b _kernel
9    || [u0+=8, 1]=r27:26
10   || r27:26=[v1+=8, 1]
```

对于memcpy函数中的主要片段, 该循环体在一个迭代周期中所需资源为1个slot(1/8), 1个branch(1/1), memread为(1/2), memwrite为(1/2), ResMII为1, RecMII为1, II为1. 优化后的汇编代码在1个周期的情况下即可完成向量化一次迭代所完成的任务, 同时平台硬件资源利用率明显得到提升.

表1所示内存拷贝与赋值函数优化结果, 以内存10000个字作为测试数据, 得到各个函数所运行的时钟周期数. 优化率是优化后的周期数与优化前函数的周期数之比.

表1 内存拷贝与赋值函数优化效果

函数名	理论优化	向量化	软件流水	优化率(%)
memcpy	3024	21258	3175	14.94
memset	1023	9043	1291	14.28
memmove	5983	44456	6354	14.29

向量化与软件流水优化下的memcpy函数在字符长度从1000字~10000字的优化效果如图2所示.

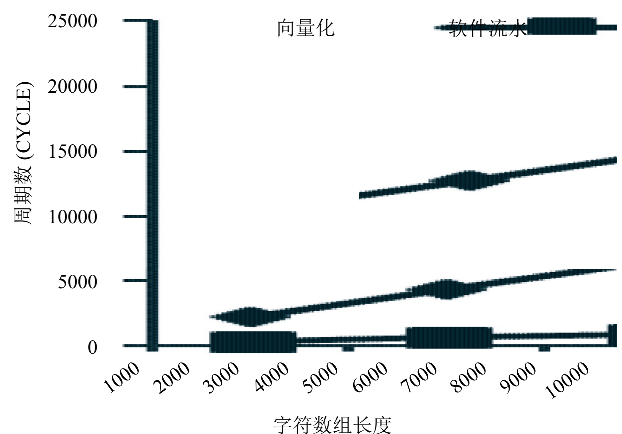

图2 向量化和软件流水优化效果

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4.2 字符串拷贝与连接函数

strcat和strncat逐字符进行比较直到目标字符串的末尾, 再将源字符串的字符拷贝到目标字符串上. 对于后面的拷贝操作仿照内存赋值函数进行拷贝.

strcpy和strncpy函数相比于字符串连接函数, 在拷贝过程中需比较目的字符串的结尾字符.

函数都需要判断字符结尾的字符, 执行过程中需要的比较语句是必不可少的, 比较语句所占的周期很大, 利用指令重排技术可以进一步提高效率.

表2所示字符串拷贝与连接函数函数优化结果, 以内存10000个字作为输入.

表2 字符串拷贝与连接函数优化效果

函数名	理论优化	向量化	软件流水	优化率(%)
strcat	12675	34848	13590	38.99
strncat	10244	32402	11144	34.39
strcpy	10305	31831	10573	33.21
strncpy	10289	32679	11421	34.94

字符串拷贝与连接函数在字符长度为10000个字下向量化相比于软件流水取得加速比的收益如图3所示.

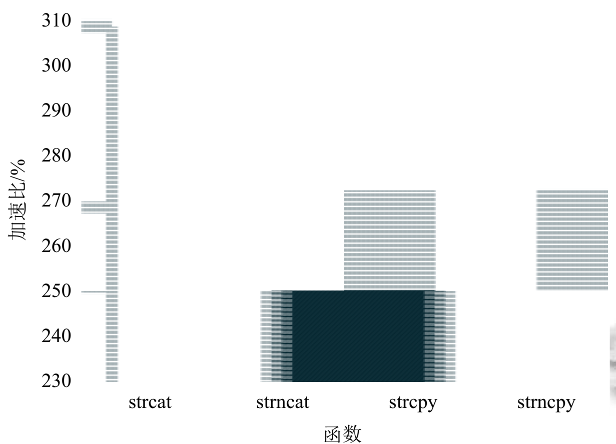

图3 字符串拷贝与连接函数优化效果

4.3 字符串和内存比较函数

Strcmp和memcmp需要同时访问两个字符串或两内存并对读取的字符进行比较. 函数算法流程是先逐字读取两个字符串到寄存器中, 接着对寄存器中字符进行比较, 如果相等, 通过移位调整字符串位置再进行比较, 否则退出.

字符串比较需要考虑字符长度问题以及所比较的范围, 在拷贝过程中还要找出不同的字符. 采取的方案是每次批量读取字符进行比较, 如果不相等, 则返回比较结果. 如果该部分子串相等, 则继续比较下一批量字符[12]. 这种逐批量读取的方式避免了逐字遍历一个很长的字符串浪费的时间. 在BWDSP中, 寄存器的读取内存的长度最大可为8个字的长度, 为了达到读取与比较的字符串长度一致, 采用一次批量读取的长度是8个字. 比较过程建立了两种比较方式.

第一种在判断长度的同时比较两个字符串的不同字符, 并将两个不同的字符返回. 此种方式, 主要思想是对所读取的源字符串或内存的8个字和目的字符串或内存8个字一一作比较. 同时使用指令重排技术把指令放在同一个执行, 让指令能够达到并行的最大程度..

第二种比较方式把比较字符作互相做&操作, &操作的结果与0比较, 判断是否相等. 这样我们可以把第一种方式的多条比较语句优化成简单比较语句, 这种算法相对于第一种减少了每次比较的过程. 同时在比较过程中, 使用BWDSP本身自带的指令sigma, 将不同字符比较的结果收集到一个临时寄存器中, 将多条比较语句优化成单条比较语句进行判断, 大大减少了比较过程中损耗的时间. 此种方式支持BWDSP的指令, 使比较结果更加简便.

表3所示字符串比较和内存比较函数经过第二次优化算法的优化结果, 以内存10000个字作为输入.

表3 字符串和内存比较函数优化效果

函数名	理论优化	向量化	软件流水	优化率(%)
strcmp	17632	101213	25300	24.99
memcmp	13200	62148	15300	24.62

第一次优化算法和第二次优化算法以函数为memcmp为例, 内存中都以最后一个字符不同为标准.

Memcmp函数在内存长度为1000字~12000字的长度下基本算法和优化算法的比较下得到的优化效果如图4所示.

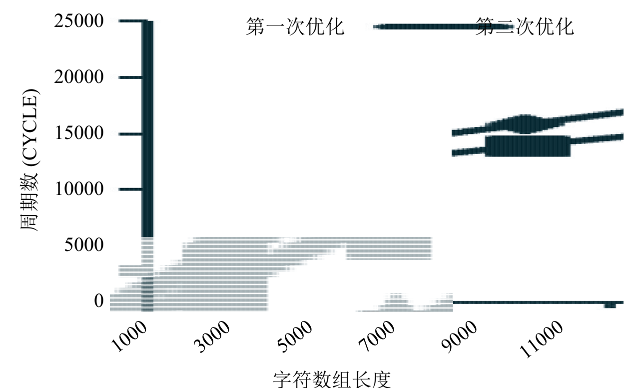

图4 memcmp在基本算法和优化算法下的效果

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4.4 字符串和内存查找函数

Memchr及strchr都是查找字符串中的特定字符, 并将该字符的地址返回. 函数的流程是逐个字符与目标字符进行比较, 直至找到该字符.

查找函数同样由于拥有大量的比较操作, 其优化过程与字符串和内存比较函数类似.

表4所示字符串和内存查找函数优化结果, 以内存10000个字作为输入.

表4 字符串和内存查找函数优化效果

函数名	理论优化	向量化	软件流水	优化率(%)
strchr	17589	113422	26049	22.97
memchr	10578	63219	12740	20.15

5 结束语

BWDSP是一款针对高性能计算领域设计的32位静态数字处理器, 本文针对BWDSP体系结构, 利用其自带指令及硬件等相关结构, 在其平台上对字符串与内存处理函数的进行优化, 实现了相关函数汇编代码, 经过最后的实验结果分析可以知道, 汇编函数的优化效果可以提升函数的性能.

参考文献

1 CETC38. BWDSP100软件用户手册. 合肥: 中国电子科技集团第三十八研究所, 2011: 1–2.

2 甄扬. 基于多核VLIW DSP的数字信号变换函数并行优化[硕士学位论文]. 合肥: 中国科学技术大学, 2015.

3 甄扬, 顾乃杰, 叶鸿. 数字信号变换函数在多簇VLIW DSP上的优化. 计算机工程, 2016, 42(3): 47–52.

4 Rau BR. Iterative module scheduling: An algorithm for software pipelining loops. Proc. of the 27th Annual International Symposium on Microarchitecture. San Jose, CA, USA. 1994. 63–74.

5 Kim W, Yoo D, Park H, *et al*. SCC based modulo scheduling for coarse-grained reconfigurable processors. Proc. of International Conference on Field-Programmable Technology. Seoul, Korea. 2012. 321–328.

6 Lam MS. Software pipelining: An effective scheduling technique for VLIW machines. Proc. of the SIGPLAN'88 Conference on Language Design and lmplementatlon. Atlanta, Georgia, USA. 1988. 318–328.

7 林海波. 基于EPIC体系结构的软件流水技术研究[博士学位论文]. 北京: 清华大学, 2003.

8 王向前, 郑启龙, 洪一. 分簇结构模调度框架研究. 中国科学技术大学学报, 2016, 46(2): 104–112.

9 Hiroyuki S, Teruhiko Y. Characteristics of loop unrolling effect: Software pipelining and memory latency hiding. Innovative Architecture for Future Generation High-Performance Processors and Systems. Maui, HI, USA. 2001. 63–72.

10 Yoshida T, Sato H. Characteristics extraction of loop unrolling and its modeling. Solid-State Electronics, 1989, 32(1): 65–68. [doi: 10.1016/0038-1101(89)90049-X]

11 李恺 翁玉萍. 基于龙芯2F的Glibc库优化. 电子技术, 2010, 37(10): 27–29. [doi: 10.3969/j.issn.1004-373X.2010.10.009]

12 李恺. Glibc库在龙芯2F上的优化[硕士学位论文]. 合肥: 中国科学技术大学, 2010.

© 中国科学院软件研究所 http://www.c-s-a.org.cn