一种面向向量化的动态指针别名分析框架

刘 鹏 赵荣彩 李朋远

(信息工程大学 郑州 450001) (数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 指针别名分析是数据流分析中的关键性技术,其分析结果是编译优化和程序变换的基础。在向量化方法和动态指针别名分析相关研究的基础上,设计了一种面向向量化的动态指针别名分析框架。该框架通过动态插桩和试运行提取指针别名信息,并反馈到向量化阶段指导向量化代码生成。从提取候选别名分析集、插桩及试运行和反馈优化3个方面对整体框架进行分析和研究。该框架基于Open64实现,并以通用测试集GCC-VECT和典型应用进行了实验评估,结果表明,该框架相比静态指针别名分析具有更精确的别名分析结果,该结果能够有效改进向量化程序的加速比。

关键词 指针别名分析,向量化,动态分析,依赖分析

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.3.005

Dynamic Pointer Alias Analysis Framework for Vectorization

LIU Peng ZHAO Rong-cai LI Peng-yuan

(The PLA Information Engineering University, Zhengzhou 450001, China)

(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract Pointer alias analysis is a key technology in data-flow analysis, whose results are the basis of compiler optimization and program transformation. Based on the related research of vectorization method and dynamic pointer alias analysis, a dynamic pointer alias analysis framework oriented to vectorization was designed. The dynamic pointer alias information is extracted by dynamic instrument and test run, and the vectorization code generation is guided by the feedback information. The whole framework was studied from three aspects, which are candidate alias analysis set extraction, instrument and test run. The framework was implemented on Open64 and evaluated in benchmark GCC-VECT and typical applications. The experimental results show that the framework has the more precise alias analysis results compared with static pointer alias analysis, and can significantly improve the speedup of vectorization program.

Keywords Pointer alias analysis, Vectorization, Dynamic analysis, Dependence analysis

1 引言

指针是C、C++等多种编程语言的重要语法特征,函数参数传递、引用堆内存空间、指针算术等指针操作体现了编程的灵活性,也减小了运行时的时空开销。然而,指针操作的灵活性也带来了很多负面效果,很难分析出指针在运行时的指向位置。若两个或多个左值表达式在相同的程序点引用相同的存储位置,则存在指针别名。指针别名会使得编译器难以理解程序的行为,并阻碍编译器进行一些潜在的优化。指针别名分析是一项基本的数据流分析技术,能够计算指针变量可能指向的内存对象的集合,分析两个指针变量是否为指针别名,其分析结果往往是其他程序分析和变换的基础,包括自动向量化[1]、安全分析[2]、bug检测[3]、硬件合成[4]和多线程程序分析[5]等等,这些分析的效果和效率严重依赖于指针别名分析的精度。

许多研究工作围绕指针分析算法的精度改进展开,主要包括考虑控制流的流敏感分析[6-8]、考虑函数调用语义的上下文敏感分析[9-11]和考虑聚类数据类型成员指向关系的域敏感分析[12,13]等,这些工作有效地推动了指针别名分析技术的发展。2008年AT&T的Lerner等人[14]指出,指针的指向关系不明是并行化面临的一个严重挑战。以如下代码的SIMD并行化为例:

```
void sum_i16(int *a,int *b,int n)
{
  for(int i=0;i<n;i++)
    a[i]+=b[i];
}
```

在不考虑指针别名引起的依赖时,仅存在循环无关依赖,对向量化不构成影响,可将8次迭代中的操作打包为向量,实现SIMD并行,生成向量化代码。然而在函数内部,指针a和b是作为形式参数传入函数体,在无法取得a和b明确指向信息时,指针a可能存在循环携带的反向真依赖,若依赖距离小于向量长度,将导致该循环无法进行向量化。

前文所述的流敏感分析等相关研究工作均为静态分析,

到稿日期:2014-05-06 返修日期:2014-07-22 本文受"核高基"国家科技重大专项(2009ZX01036)资助。

刘 鹏(1981-),男,博士生,主要研究方向为高性能计算和先进编译技术,E-mail:magiceelp@gmail.com;赵荣彩(1957-),男,博士,教授,博士生导师,主要研究方向为先进编译技术、软件逆向工程等;李朋远(1989-),男,硕士生,主要研究方向为高性能计算和先进编译技术。

为保证程序变换的正确性，对指针往往进行保守处理，很可能将一些运行中不互为别名的指针对判定为指针别名，这种不精确分析将导致部分循环无法向量化。针对此问题，本文给出一种面向向量化的动态别名分析框架，对一些静态分析无法判明的别名关系进行处理，并基于改进后的别名关系进行向量化变换。

本文第 2 节对向量化和动态指针分析的相关研究进行阐述；第 3 节给出了动态别名分析框架的总体设计；第 4 节详细阐述了整个框架在各个阶段采用的关键技术；第 5 节在相关测试集进行实验评估；最后是结论。

2 相关研究

本文的研究工作是为提高向量化编译器的性能提出的一种动态指针别名分析框架。当前的向量化方法主要针对 SIMD 扩展部件，主要方法包括由向量机上发展而来的传统向量化[15]、Larsen 提出的超字级并行[16]和模式匹配的方法[17]、在函数级向量化[18]、基于动态规划的向量指令选择[19]、流水向量化[20]等方面，也存在一些典型研究工作。由于这些方法的实施所基于的依赖理论大致相当，因此本文所提框架可以无差别地应用于上述针对 SIMD 扩展部件的向量化方法。

别名分析经过 20 多年的研究，可伸缩性和计算精度有了很大的提升，研究内容涉及到流敏感、路径敏感、上下文敏感、域敏感等多个维度的静态分析。动态指针别名分析的研究工作相对较少，Mock 等人[21]在 Calpa 系统[22]中实现了动态指针分析，并将分析结果与 Das 的一级流分析[23]的静态指针信息进行比较，动态指向集的平均大小接近 1，而静态指向集的平均大小远远大于动态指向集。Gross 等人[24]对动态指向分析进行评估，并指出动态指向信息能够显著增强程序理解和编译优化的能力。Wu 等人[25]在 LLVM 上实现了一种别名纠错系统 NEONGOBY，用来更正静态分析中的漏报和误报。Zhang[26]通过动态指针追踪计算精确的指向集，其主要应用于内存保护和垃圾收集。这些动态分析方法虽然改进了静态分析的精度，但无法直接应用于向量化编译器。

3 总体设计

面向向量化的指针别名信息通过动态插桩和试运行获得，并反馈到向量化阶段指导向量化代码生成，总体框架如图 1 所示。

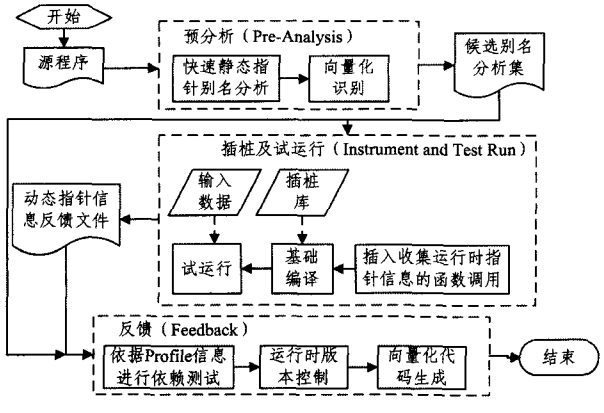

图 1 面向向量化的动态指针别名分析框架

该框架主要包括预分析、插桩及试运行和反馈 3 个部分，具体分析如下。

1）预分析。通过快速静态指针分析和向量化识别提取出候选别名分析集。为保证整个框架的可伸缩性，这里的静态指针别名分析采用一种流不敏感和上下文不敏感的 Steensgard 类型的指针分析算法，该算法在分析中采用 Union-Find 进行指针合并，分析中的不精确性将保守地产生更多的别名信息，因此可将该别名信息作为分析的基准别名集。向量化识别分析循环是否能够向量化，并对由指针别名引起的无法判定的循环进行分析，将指针读写信息与快速静态指针分析的结果进行对比，并提取候选别名分析集。

2）插桩及试运行。对预分析确定的候选别名分析集中的指针信息进行插桩，分析运行时的指针地址、上下文、循环的迭代次数和引用点的跨步长度等信息，链接插桩库进行基础编译，并进行试运行，运行结束后生成动态指针信息反馈文件，反馈信息中包括运行时的指针地址、读写信息、访存跨度、指针在源程序中的位置和频度等信息。

3）反馈。根据 Profile 信息进行向量化依赖测试，根据测试结果进行运行时版本控制，仅激进地生成向量化版本或同时生成串行及向量化多个运行时版本，并进而进行向量化代码生成。

4 面向向量化的动态别名分析框架

4.1 提取候选别名分析集

该框架中指针分析的目的是提升分析的精度，并向量化尽可能多的循环，若对所有指针变量进行插桩，将产生过多冗余分析，从而带来不必要的时间和空间开销耗费。因此，通过提取候选别名分析集，针对向量化的特点仅对部分指针变量进行分析，这种可伸缩性体现在如下两个方面：

（1）由候选别名分析集确定插桩的指针变量，减小程序试运行的开销和试运行后生成的动态指针信息反馈文件；

（2）以候选别名分析集中的可能别名作为基准，在动态反馈阶段结合动态指针信息反馈文件进行运行时可能别名的验证，以提升动态反馈阶段的分析效率。

以如下代码段进行分析：

```
int a[1024],b[1024];
void main()
{
    //initialize
    int * x=a;
    foo(x+4,x)
}
foo(int * p0,int * q0)
{
  int * p1=b;
  int * q1=b+4;
  for(int i=0;i<512;i++) //L1
  {
    *(q0+i)=*(q0+2*i); //S1
    *(p0+i)=*(q0+i); //S2
  }
}
```

该例子在过程间产生约束 $p0=x+4$ 和 $q0=x$，在过程内产生约束 $p1=b$ 和 $q1=b+4$，Steensgard 类型的快速静态指针分析将判定 $p0$ 和 $q0$，$p1$ 和 $q1$ 为两对可能别名。由于指针 $p0$ 和指针 $q0$ 互为别名，在向量化识别中，循环 L1 为指针别名引起的无法判定能否向量化的循环，通过分析，将别名对提

取出来插入候选别名分析集。制约向量化的依赖关系是循环携带真依赖和循环携带反依赖，依据指针的读写关系提取别名对，并记录其在程序中的位置。候选别名分析集中每一项为一个偶对，表示程序中的可能别名关系。偶对中的第一元素为读指针信息，第二元素为写指针信息，每个元素依次记录了指针变量、指针在程序中的位置、指针访存位置和指针数据类型信息。

则示例中提取的候选别名分析集中的元素为：

$\langle(q0,L1\_S1,q0+2*i,I4),(q0,L1\_S1,q0+i,I4)\rangle$

$\langle(q0,L1\_S1,q0+2*i,I4),(p0,L1\_S2,p0+i,I4)\rangle$

$\langle(q0,L1\_S2,q0+i,I4),(p0,L1\_S2,p0+i,I4)\rangle$

由于循环无关依赖并非影响向量化的依赖关系，因此候选集中不包含元素$\langle(q0,L1_S2,q0+i,I4),(q0,L1_S1,q0+i,I4)\rangle$。集合中的指针变量 $p0$ 和 $q0$ 将作为插桩对象进一步分析，而 $p1$ 和 $q1$ 虽然为可能别名，但未包含在任何循环中，则不对该指针变量进行插桩。除指针变量外，程序中会包含全局数组和栈空间数组，在插桩和反馈阶段，这些数组按指针变量相同方式进行处理。

4.2 插桩及试运行

为提高框架的可伸缩性，我们尽量减少动态插桩的位置。由于指针别名分析的目的是充分发掘 SIMD 并行性，一些复杂的语句可能导致无法向量化或向量化无明显收益。因此，我们仅处理引用点跨步长度为常量的循环，对形如$a[i+x]=b[i+y]$的语句，x 或 y 通过前向替代后无法用归纳变量 i 线性表出时，不对包含该语句的循环进行插桩。进行折中后，插桩位置为包含当前语句循环的第一次迭代及循环的迭代次数，则时间开销由多项式时间减少为 O(1)，在保证向量化分析能力的同时，有效减少了试运行的时间开销。

动态插桩对候选循环进行遍历，在循环前、循环体和循环后插入运行时插桩库的函数调用。在热点循环前进行插桩函数初始化；在循环内对候选别名分析集中的指针地址进行分析，记录循环内第一次迭代的首地址和循环内的迭代次数信息；在热点循环的出口对收集的信息进行处理。

链接插桩库通过基础编译，接收输入数据试运行可执行文件。试运行中，插桩代码对程序进行分析，提取动态信息，并将收集的信息存入动态指针信息反馈文件，生成的动态指针信息包括循环标识信息和指针访存信息。

循环标识信息按上下文区分整个循环的每次执行，该结构为 $Lid=(LoopID,CStr)$，其中：$LoopID$ 用于标记当前的循环；$CStr$ 为当前循环的上下文。

循环标识信息与指针访存信息为一对多的映射关系，指针访存信息记录了当前 Lid 中每个指针的读引用或写引用信息，该结构是一个七元组：$PtsMem=(V,SID,Addr,N,St,Ty,Flag)$，其中：$V$ 为指针变量；SID 用于标记当前的语句；$Addr$ 是指针访问的首地址；N 是迭代次数；St 是访存跨步长度；Ty 是指针访问的数据类型；$Flag$ 用于标记指针读或写。

Lid 中的 $CStr$ 为运行时的插桩得到的当前循环的上下文调用信息，因此该分析是上下文敏感的。这些信息将作为动态反馈阶段的输入数据，修正候选别名分析集中的别名信息，并进行运行时版本控制和向量化代码生成。

4.3 反馈优化

程序执行前的二次编译需要将动态指针的 profile 信息反馈到编译器的向量化阶段，根据 profile 信息进行优化和向量化代码生成。由于编译优化对程序的中间表示进行变换，为保证反馈的正确性，一般原则为：将 profile 信息反馈到插桩的编译阶段，这样能防止出现插桩位置和反馈位置不一致而导致错误的程序变换的问题。

动态反馈的目的是依据试运行中的信息判断指针别名之间所产生的依赖，并指导程序变换和优化。进行向量化变换的主要思想是根据动态指针别名信息生成串行版本或向量化版本的代码段。若程序中存在 M 组制约向量化的别名关系，每组至少包含一个别名对，M 组别名不存在相互包含的关系，则穷尽式的做法是生成 M 个版本的循环体，这些循环体可能为串行版本、部分向量化版本或完全向量化版本。然而，过多的判断条件可能导致分支计算操作过多，在极少出现非别名对的情况生成向量化代码反而会增加时间开销。例如：在 1000 次迭代中，仅 1 次迭代不存在依赖，其他 999 次迭代中均有依赖产生，对于这种情况，显然不应生成向量化版本的代码。因此，在分析别名间依赖的同时，还应分析依赖的频度信息。

该阶段的输入数据为候选别名分析集和动态指针信息反馈文件，根据这些信息建立两个映射关系，则动态别名信息的映射关系如图 2 所示。第一个由循环标识号 $LoopID$ 映射到该循环内的候选别名集，第二个由候选别名映射到不同 $CStr$ 所包含两指针变量的访存信息。进一步在该映射结构的基础上进行依赖分析，分析试运行中产生的真依赖、反依赖以及依赖的频度信息。

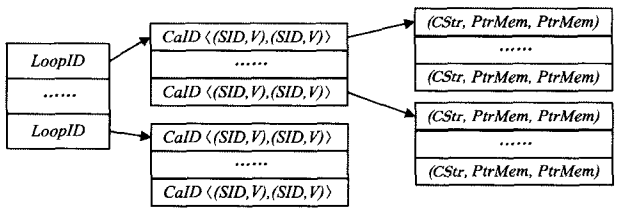

图 2 动态别名信息的映射关系

分析流程如图 3 所示。依次遍历每一个循环，在循环内遍历候选别名集中的每一组别名对，由于候选别名集中的别名对可能出现在多组上下文调用中，需要对每一处上下文访存信息依次进行判别。

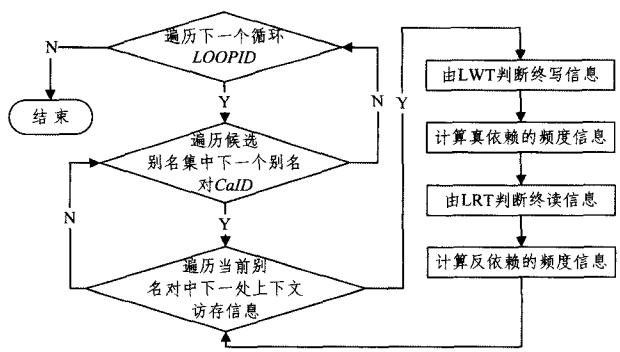

图 3 动态别名信息依赖分析流程

依赖关系的判断采用终写树(LWT)[27]结构，该结构存储了一个读引用到写引用的映射，其设计目的主要是用于数组私有化，而终写关系恰恰能够计算依赖产生的频度。本文对该结构进行了两点扩展：

(1)扩展到指针变量的读写引用，分析中对数组和指针均采用首地址及归纳变量表达式偏移的形式；

(2)扩展为终读树(LRT)结构，该结构类似地存储了一个

写引用到读引用的映射，用于分析程序中所出现的反依赖。

在每一处上下文访存信息的依赖分析中，首先由 LWT 判断终写关系，并计算循环携带真依赖的频度信息，该信息包括依赖出现的次数和迭代次数；而后由 LRT 判断终读关系，并计算循环携带反依赖的频度信息。分析得到的频度信息包括每次循环上下文 *CStr* 中产生循环携带真依赖的迭代次数 N_{TD}、产生循环携带反依赖的迭代次数 N_{AD} 和总迭代次数 N_{Total}。当依赖距离大于向量长度时，不会对向量化产生影响，此情况下的依赖不计入 N_{TD} 和 N_{AD}。

运行时版本控制基于依赖关系和频度信息生成串行版本和向量化版本的代码，使得多版本向量化代码生成更为精简、高效，在减小运行时的时间开销的同时，使得代码规模的增长最小化。在由判明的依赖关系构建的依赖图中，若动态别名信息不影响向量化，则仅生成激进的完全向量化代码；若制约向量化的别名对大于等于 1，则需根据依赖的频度信息进一步分析。

由依赖迭代次数 N_{TD}、N_{AD} 和 N_{Total} 计算候选别名集中的每个别名对在迭代中产生依赖的比率：

$$rate=\frac{\sum_{CStr}(N_{TD}+N_{AD})}{\sum_{CStr}N_{Total}}$$

若在依赖图上存在 M 组制约向量化的别名关系，不考虑依赖的频度信息时，将生成 M 个版本的向量化代码。即便不同组的别名关系可能生成相同版本的向量化代码，而且可将部分相同的分支版本进行合并，但仍会存在较大的向量化代码变换开销和运行时分支计算开销。而实际测试集中，在依赖图上大多仅存在一到两组制约向量化的别名关系。本文的方法是生成一个较优的向量化版本，首先计算 M 组别名关系中依赖比例的最小值 T_x：

$$T_x=\min(\sum_{i_1}rate_{i_1},\sum_{i_2}rate_{i_2},\cdots,\sum_{i_M}rate_{i_M})$$

其中，$1\leqslant x\leqslant M$ 且 $x\in Z$。

通过设置依赖比率阈值 T，将 T_x 与 T 进行比较，若存在 $T_x\leqslant T$，则在生成串行版本代码的同时，生成向量化版本代码。该向量化版本代码对应第 x 组别名关系，进行分支条件判断时插入形如 *NoAlias*(a,b) 的语句，使得程序在运行时进入向量化分支。若上式不成立，则仅生成串行版本的代码。

5 实验与分析

实验采用面向国产 CPU 申威 1600 的向量化编译器 SW-*VEC*，*该编译器基于高性能开源编译器 Open64 5.0 研制开发*，在已开发的反馈式编译工具[28]上实现了动态指针别名分析。编译环境为 IBM 3850 服务器，处理器频率 2.0GHz，内存 4GB，L1 数据缓存为 32kB，L2 缓存为 256kB，基本页面为 8kB。操作系统内核为 Linux 2.6.18，版本为 Redhat Enterprise AS 5.0。

为了验证动态指针分析框架的优势，测试集采用 gcc-vect 中包含指针语法特征的 8 个 C 程序和浅水波应用中的两个核心循环。10 个测试用例采用静态程序分析将保守地产生可能别名对，导致核心循环不能向量化。评估内容为串行程序的执行时间、向量化后程序的执行时间和加速比。运行环境为国产高性能神威蓝光服务器，CPU 为申威 1600 处理器，SIMD 扩展部件的向量长度为 256 位。表 1 列出了动态指针分析框架在 SW1600 处理器上的测试结果。

表 1　动态指针分析框架在 SW1600 的测试结果

测试用例	串行程序(秒)	向量化程序(秒)	加速比
no-vfa-vect-51	17.4333	3.799	4.59
no-vfa-vect-53	30.269	3.803	7.977
no-vfa-vect-57	7.824	2.591	2.929
no-vfa-vect-61	7.592	4.921	1.544
no-vfa-vect-79	2.309	0.586	3.909
vect-60	3.516	1.297	2.712
no-vfa-pr29145	3.534	3.534	1
pr21591	8.033	2.05	3.925
swe_core1	214.715	131.517	1.633
swe_core2	809.278	351.586	2.302

在 SW1600 上所评估的 10 个静态分析无法向量化的测试用例中，有 9 个能够通过动态指针分析框架达到较好的加速比，no-vfa-pr29145 用例由于存在自身真依赖导致无法向量化。由实验结果可以看出，对于通用测试集和典型应用，当静态指针分析不能判定别名关系而导致无法向量化时，动态指针分析框架能够判定更多的非别名对，消除节点间的依赖并进行向量化变换，能够显著提升程序性能。

结束语　C、C++等弱类型语言的静态指针别名分析过于保守，其分析精确不高，保守的别名信息导致向量化编译器无法对部分热点循环进行向量化。本文针对此问题提出了一种面向向量化的动态别名分析框架。通过快速静态别名分析提取候选别名分析集，以提升框架的可伸缩性；针对向量化的特点，对程序进行插桩和试运行；采用终写树进行动态别名信息的依赖分析，根据分析结果进行版本控制并进行向量化代码生成。实验结果表明：该框架所计算的别名信息明显优于静态分析的结果，显著提升了向量化程序的加速比。

参 考 文 献

[1] Pryanishnikov I, Krall A, Horspool N. Pointer Alignment Analysis for Processors with SIMD Instructions[C]//Proceedings of the 5th Workshop on Media and Streaming Processors. San Diego, CA, 2003. 50-57

[2] Fink S J, Yahay E, Dor N, et al. Effective typestate verification in the presence of aliasing[J]. ACM Transaction on Software Engineering and Methodology(TOSEM), 2008, 17(2): 133-144

[3] Li Lian, Cifuentes C, Keynes N. Practical and effective symbolic analysis for buffer overflow detection[C]//Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering(FSE'10). ACM, 2010: 317-326

[4] Zhu Jian-wen. Towards scalable flow and context sensitive pointer analysis[C]//Proceedings of the 42nd annual Design Automation Conference(DAC '05). ACM, 2005: 831-836

[5] Salcianu A, Rinard M. Pointer and escape analysis for multithreaded programs[C]//Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming(PPoPP '01). ACM, 2001: 12-23

[6] Rei T. Expression Data Flow Graph: Precise Flow-Sensitive Pointer Analysis for C Programs[D]. Diss. University of Alberta, 2011

[7] Yu Hong-tao, Xue Jing-ling, Huo Wei, et al. Levelby Level: Making Flow- and Context-Sensitive Pointer Analysis Scalable for Millions of Lines of Code[C]//Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization(CGO'10). ACM, 2010: 218-229

[8] Hardekopf B, Lin C. Semi-Sparse Flow-Sensitive Pointer Analysis[C]// Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'09). ACM, 2009: 226-238

[9] Yan Da-cong, Xu Guo-qing, Rountev A. Demand-Driven Context-Sensitive Alias Analysis for Java[C]// Proceedings of the 2011 International Symposium on Software Testing and Analysis(ISSTA'11). ACM, 2011: 155-165

[10] Li Xin, Ogawa M. An ahead-of-time yet context-sensitive points-to analysis for Java[J]. Electronic Notes in Theoretical Computer Science, 2009, 253(5): 31-46

[11] Chris L, Lenharth A, Adve V. Making context-sensitive points-to analysis with heap cloning practical for the real world[J]. ACM SIGPLAN Notices, ACM, 2007, 42(6): 278-289

[12] Pearce D J, Kelly P H J, Hankin C. Efficient Field-Sensitive Pointer Analysis of C[J]. ACM Transactions on Programming Languages and Systems(TOPLAS), 2007, 30(1): 4

[13] Yu Hong-tao, Zhang Zhao-qing. An Aggressively Field-Sensitive Unification-Based Pointer Analysis[J]. Chinese Journal of Computers, 2009, 32(9): 1722

[14] Experiences coding Non-Uniform Parallem using the CUDA GPGPU Architecture[C]// NJPLS2. August 2008

[15] Randy A, Kennedy K. Optimizing Compilers For Modern Architectures: A Dependence-based Approach [M]. Morgan Kaufmann, 2001: 790

[16] Samuel L, Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets[J]. Sigplan Notices-SIGPLAN, 2000, 35(5): 145-156

[17] Transforming and parallelizing ANSI C programs using pattern recognition[C]// 7th Interational Conference, HPCN Earope 1999 Amsterdam. 1999: 673-682

[18] Ralf K, Hack S. Whole-function vectorization. Code Generation and Optimization[C]// 2011 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization(CGO). IEEE, 2011: 141-150

[19] Rajkishore B, Zhao Ji-sheng, et al. Efficient selection of vector instructions using dynamic programming[C]// 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2010

[20] Markus W, Luk W. Pipeline vectorization. Computer-Aided Design of Integrated Circuits and Systems[J]. IEEE Transactions on Computer-aided Design of Integrated Circuits and System, 2001, 20(2): 234-248

[21] Markus M, et al. Dynamic points-to sets: A comparison with static analyses and potential applications in program understanding and optimization[C]// Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering. ACM, 2001

[22] Mock M, Berryman M, Chambers C, et al. Calpa: A tool for automating dynamic compilation[C]// 2nd Workshop on Feedback-Directed Optimization. 1999: 291-302

[23] Manuvir D. Unification-based pointer analysis with directional assignments[J]. Acm Sigplan Notices, 2000, 35(5): 35-46

[24] Axel G. Evaluation of dynamic Points-To Analysis[M]. 2004

[25] Wu Jing-yue, et al. Effective dynamic detection of alias analysis errors[C]// Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013

[26] Zhang Kun. Dynamic pointer tracking and its applications[M]. 2010

[27] Maydan, Dror E, Amarasinghe S P, et al. Array-data flow analysis and its use in array privatization[C]// Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1993

[28] Hao Yun-long. Research on Profile-Guided SIMD Vectorization Identification and Optimization [D]. Information Engineering University, 2011

(上接第 25 页)

[8] Modha D S, Singh R. Network architecture of the long-distance pathways in the macaque brain[J]. Proceedings of the National Academy of Sciences, 2010, 107(30): 13485-13490

[9] Hagmann P, Cammoun L, et al. Mapping the Structural Core of Human Cerebral Cortex[J]. PLOS Biology, 2008, 6(7): 1479-1493

[10] Preissl R, Wong T M, et al. Compass: A scalable simulator for an architecture for Cognitive Computing[C]// High Performance Computing, Networking, Storage and Analysis (SC) 2012. Almaden, San Jose, CA, USA, 2012: 1-11

[11] Eliasmith C, Stewart T C, et al. A large-scale model of the functioning brain[J]. Science, 2012, 338(6111): 1202-1205

[12] George D, Hawkins J. Towards a Mathematical Theory of Cortical Micro-circuits[J]. PLoS computational biology, 2009, 5(10): 1-26

[13] Jaros R G, George D. Hierarchical temporal memory system with enhanced inference capability[J]. Google Patents, 2012(7)

[14] 姚宏亮,王秀芳,王浩. 一种基于结构分解和因子分析的贝叶斯网络隐变量发现算法[J]. 计算机科学, 2012, 39(2): 244-249

[15] Liu Lu, Zhu Fei-da, Zhang Lei, et al. A probabilistic graphical model for topic and preference discovery on social media[J]. Neurocomputing, 2012, 95: 78-88

[16] Frey B J, Jojic, Nebojsa. A comparison of algorithms for inference and learning in probabilistic graphical models[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, 27(9): 1392-1416

[17] Ngiam Ji-quan, Khosla A, et al. Multimodal deep learning[C]// Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011. Bellevue: ACM, 2011: 689-696

[18] Srivastava N, Salakhutdinov R. Multimodal learning with deep Boltzmann machines[C]// Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems(NIPS), 2012. Lake Tahoe: Curran Associates, 2012: 2231-2239

[19] Blei D M. Probabilistic topic models[J]. Communications of the ACM, 2012, 55(4): 77-84

[20] 李志欣,施智平,李志清,等. 融合语义主题的图像自动标注[J]. 软件学报, 2011, 22(4): 801-812

[21] Modha D S, Singh R. Network architecture of the long-distance pathways in the macaque brain[J]. Proceedings of the National Academy of Sciences of the United States of America, 2010, USA, 2010, 107(30): 13485-13490

[22] Joseph R. Neuropsychiatry, Neuropsychology, Clinical Neuroscience[M/OL]. http://brainmind.com/BrainOverview.html, 2000