积少成多




< 2021年11月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |



**公告**

昵称： south343
园龄： 9年11个月
粉丝： 6
关注： 7
+加关注



**搜索**

找找看
谷歌搜索

**常用链接**

我的随笔
我的评论
我的参与
最新评论
我的标签

**我的标签**

leetcode(62)
c++(47)
算法(15)
python(9)
实习(8)
linux(7)
shell(7)
linux basic(7)
多线程(6)
redhat(6)
更多

**随笔档案**

2020年4月(1)
2019年11月(2)
2018年7月(2)
2018年3月(2)
2017年10月(1)
2017年9月(10)
2017年8月(2)
2017年7月(11)
2017年6月(7)
2017年5月(5)
2017年4月(2)
2017年3月(1)
2016年12月(2)
2016年10月(1)
2016年9月(6)
更多

**阅读排行榜**

1. undefined reference to 'dlopen';undefined reference to 'dlclose';undefined reference to 'dlerror'等问题(16708)
2. c++ 中double与string之间的转换,char *(16285)
3. shell 中的expect 用法(15178)


## 编译器研究之路_未来50年

今天，复杂而泛在的软件架构支撑着全球经济，编译器和高级语言正是这些软件的基石。强大而优雅的编译技术在硬件综合等领域同样有着"连城"的价值。毫不夸张地说，与半导体技术一样，编译器和高级语言处于信息时代的核心地位。

本文将展示编译技术过去的辉煌与成就，当前的研究现状，未来的发展方向。我们还将提出四项建议，均来自美国科学基金资助的"编译器研究和教育的未来发展研讨会"上的主要内容。

IBM第一款优化编译器发布至今已50周年了。这正是对编译器发展现状进行评估，对未来50年发展方向进行探讨的一个契机。下一个10年，即2010年到2020年，在整个计算机领域面临的两个挑战中，编译器研究将会起到至关重要的作用。多核处理器的编程成本 在可预见的未来，计算能力将会继续飞速提升，并行度将取代时钟频率成为带动计算能力发展的原动力。正在实施的向并行结构的转变，给微处理器和软件产业带来了前所未有的挑战。2005年，英特尔公司首席技术官贾斯廷·拉特纳（Justin Rattner）说："我们正站在向多核和多线程架构演变的潮头，但我们还无法提供这个转变所必须的简单编程方式……"[3]

复杂软件系统的安全性和可靠性 软件系统正在日益复杂化，使解决程序缺陷和安全攻击问题变得更为迫切。2002年，美国商务部国家标准与技术研究院（National Institute of Standards and Technology，NIST）进行了一项关于由程序缺陷造成的重大经济影响的调查。调查指出，程序缺陷"普遍存在并且极为有害，已导致美国每年约595亿美元的经济损失，占全年国内生产总值的0.6%。"2005年，美国总统信息技术顾问委员会（President's Information Technology Advisory Committee，PITAC）《信息社会安全：转型期的研发重点》（Cyber Security：A Crisis of Prioritization）的报告，提出了十项重点研究项目，其中就包括了软件工程安全性和软件保障。该报告提到"目前普遍运用的软件工程措施无法防止危险的错误，例如对缓冲区溢出的不当处理，这些缺陷导致每年成百的攻击程序对数以百万计的计算机造成威胁。未来，由于敌对势力——无论是来自国外还是国内——会掌握更为复杂的向关键软件植入恶意代码的能力，国家（指美国）将面临更多的挑战……"

文化变迁

要解决这些难题，编译器领域需要改变现存的研究模式。目前，编译器相关课题主要由基于一次性组织结构的小规模或个别研究人员完成。一方面，对于学术研究环境而言，开发和维护完整的编译器架构过于复杂；另一方面，我们又确实需要从研究到实践的技术转移，需要将新的编译研究成果集成到已有的架构里。为了解决这个难题，学术界和工业界需要一种新的合作方式，以便开发出共享的开源架构、有代表性的测试基准以及可重复的实验。如果这种新模式能够成功， 它将同时影响商业应用和科研能力。2005年PITAC的另一篇报道《计算科学：确保美国的竞争力》（《Computational Science：Ensuring America's Competitiveness》）中强调应当研究如何利用包括编程模型及编译器在内的软件技术，以使美国在计算科学中保持竞争力（见文本框）。这份报道指出："由于国家（美国）的研究基础设施尚未跟上科技发展的步伐，当今的计算科学发展出现了不均衡的局面，表现在软件基础发展的步伐不能适应不断发展的硬件和应用的需求。研究极度匮乏造成的这种不均衡使（美国的）研究者不得不在一个支离破碎，而不是现代化高质量的软件基础上工作。后果是，无论研究人员还是计算系统的生产率严重下降。"

成就

2 0 世纪5 0 年代编译器领域刚刚起步，其研究焦点局限于从高级语言到机器码的转换以及优化程序对时间和空间的需求。此后，该领域产生了大量的有关程序分析与转换、代码自动生成以及运行时服务等方面的新知识。同时，编译算法也被用于便利软件和硬件开发、提高应用程序性能、检测或避免软件缺陷和恶意软件等方面。编译领域与其它方向越来越多地相互交叉渗透，这些方向包括计算机体系结构、程序设计语言、形式化方法、软件工程以及计算机安全等。诚如大家所见，术语"编译"与计算机科学太多的方面有关，而这个术语本身含义过窄，已无法表达这一领域目前的研究范围了。

到目前为止，编译器领域最为突出的成就是高级语言的广泛使用。从银行、企业的管理软件，到高性能计算和各种万维网（Web）应用，今天的绝大多数软件都是用高级语言编写并经过静态或动态编译的。在20世纪60年代末70年代初，面向对象和数据抽象语言刚面世时，虽然人们对其能否有效实现还心存疑虑，但已经意识到这些概念有可能极大地提高编程人员的生产率，编译器领域发明的静态和动态优化则完全打消了这些疑虑。*来，尤其是伴随着20世纪90年代中期Java的出现，易于管理的运行时系统，包括垃圾回收和即时（Just-in-Time）编译技术，通过根除内存泄露进一步提高了程序员的开发效率。

众多的编译算法，包括词法分析、类型检查和推导、数据流分析、基于数据依赖性分析的循环变换、基于图着色的寄存器分配以及软件流水等，都是计算机科学中的奇葩。通过集成到各种功能强大而应用广泛的工具中，这些方法极大地影响着计算领域的实践。与早期的编译器实现相比，今天的编译算法明显变得越来越复杂。有两个例子可以说明这方面的发展：早期的编译器采用简单直观的技术对程序进行词法分析，而今天的词法分析技术则基于形式语言和自动机理论，这使得编译器前端的开发更为系统化。同样，原来有关可重构编译器工作中采用简单直观的技术进行依赖分析和循环变换，而今天，编译器的这一部分工作采用了基于整数线性规划的强大算法。

4. ubuntu14.04 boost动态库找不到 libboost_system.so.1.58.0(10108)
5. 链表的问题，ListNode问题(9580)

**评论排行榜**
1. ubuntu14.04 boost动态库找不到 libboost_system.so.1.58.0(3)
2. C++函数的返回值——返回引用类型&非引用类型(1)

**推荐排行榜**
1. C++函数的返回值——返回引用类型&非引用类型(1)
2. shell中的>&1和 >&2是什么意思？(1)
3. c++封装编写线程池(1)
4. 模板与c++11--右值引用(1)
5. DFS和BFS遍历的问题(1)

**最新评论**
1. Re:C++函数的返回值——返回引用类型&非引用类型
int & rr = ref(); //err
当用引用变量接收返回引用类型时，栈内内存被释放
--BillWong
2. Re:ubuntu14.04 boost动态库找不到 libboost_system.so.1.58.0
@ 51博您好!我现在也遇到这个问题,请问您是怎么解决的呢...
--二哈这个名字被注册了
3. Re:ubuntu14.04 boost动态库找不到 libboost_system.so.1.58.0
@ 51博 linux系统下应该有这个文件吧。没有这个文件，那就自己touch一个新文件，添加上去（文件具体内容，自己搜索）； ubuntu系统（linux相关系统）编译、链接动态库和静态库是有规定。...
--south343
4. Re:ubuntu14.04 boost动态库找不到 libboost_system.so.1.58.0
你好，打扰您了，我遇到这样的问题了，请问如果我没有安装这个库，就是我的所有路径下根本就没有这个文件，那我要怎么安装啊?
--51博

为了追求一系列优化目标，比如避免冗余计算、合理分配寄存器、增强局部性以及更好地利用指令级并行等等，大多数商品编译器都提供了代码优化的能力。编译器优化通常会带来令人满意的性能提升，在某些情况下，编译器生成代码的性能甚至接*目标*台的峰值性能。而利用传统的手工调优方法来达到相*的结果，尤其是对于大型应用程序，则会极其困难，极其费时费力，而且很容易出错。为线性代数和信号处理开发的自动性能调优程序生成器在这方面做得相当有成效。用于检测程序缺陷和安全隐患的工具日渐普及，大量软件开发人员都在使用它们。这些工具可以十分有效地检测出一些常见的错误或缺陷（例如不当的内存分配和释放、数据竞争以及缓冲区溢出等）。用于软件可靠性和安全性的程序分析算法越来越重要，其标志就是运用这些算法的软件工具产业正在稳步成长。

研究人员正在编译器领域中进行着激烈的智力角逐，在主流会议中，包括“ACM1Symposium on Programming Language Design and Implementation”（编程语言设计与实现，PLDI）、“ACM Symposium on Principles of Programming Languages”（编程语言原理，POPL）、“ACM Symposium on Principles and Practice of Parallel Programming”（并行编程原理与实践，PPoPP）以及“ACM Conference on Object-Oriented Programming Systems,Languages and Applications”（面向对象的编程系统、语言和应用，OOPSLA）等都有体现。这些会议在计算机科学界有很大影响力，备受推崇，并且其评审过程极为严格2。这一领域影响力的另一个证明是“编程语言”和“编译器”这两个词，至少在7届图灵奖得主的引用中出现，包括2005年的彼得·诺尔（Peter Naur）和2006年的弗兰·艾伦（Fran Allen）。（译者注：最新公布的2008年图灵奖得主芭芭拉·利斯科夫（Barbara Liskov）女士也是在程序语言方面做出奠基性贡献。）

编译器面临的挑战

机器与软件的日益复杂化、多核处理器的介入以及对安全的关注都是当今急需解决的重要紧迫问题。这里试述编译技术在这些问题的解决中所扮演的角色：

程序优化

我们正生活在多核时代，今后即使时钟频率还能提高，也将是非常缓慢的。而处理器芯片上的内核数将很可能会每两年翻一番。按此计算，到2020年，一个微处理器芯片可能搭载着数百甚至上千个内核，并且这些核可能是异构的，其中有些还可专门提供特定功能。要改进应用程序的性能和能力（提高执行速度，降低能耗），最关键的就是更好地利用大规模并行的硬件。这方面对编译研究的挑战是，如何在不对编程人员造成过多负担的条件下尽可能地利用目标机器的潜力，包括其并行性。

英特尔高级研究员大卫·库克（D a v i d Kuck）在交流中，强调了编译器在解决多核带来的挑战中的重要性。他认为对于优化编译而言，其难点在于其组合复杂度。当计算机扩展到新的应用领域，或出现新的架构和特性时，编程语言也会同时进行扩展。为了提高性能，计算机体系结构（单核或多核）正在变得越来越复杂，而编译器的优化功能必须能填补这一日益加宽的鸿沟。如今人们对编译器的基础问题已经有了很好的认识，但在过去几十年中，何时何处采用何种优化手段一直是困扰人们的难题。目前的编译器被设定为使用固定的策略（如在某种特定数据环境下对单个函数做优化），如果直接转到全局环境中（如整个应用程序中），对遇到的不同代码做优化是不行的。

大卫·库克还说：“未来最好的解决方案是采用自适应的编译策略，使之能基于‘脚本代码codelet）‘语法和优化的潜能来发掘出各种等价类。这是一项艰深的研究课题，其相关细节目前只是处于萌芽状态。然而它的成功将会带来巨大的性能提升以及更简化的编译器结构，同时满足新语言和体系结构的需要。”

并行编程的主流化

虽然人们从30年前就开始了有关并行程序设计的研究，但是到目前为止只有小部分应用程序领域（例如计算科学和数据库）被视为规范。这些服务器端的应用，大多数只处理结构化数据（例如数组和关系表），其中的计算通常可以在同步很少的情况下并行执行。与此相反，许多客户端应用程序需要处理非结构化的数据（例如集合和图），并需要在更细粒度上做多处理器间的同步。程序员缺少能够帮助他们在高层次上抽象地为这类应用进行编程的工具，因而只能显式地管理并行、局部性、通讯、负载均衡、能耗需求以及其它需要考虑的优化工作。由于并行已无处不在，因此对于开发产品的软件来说，跨并行*台和跨处理器的程序的性能可移植性会变得至关重要。编译器技术的突破是使并行编程成为主流的关键。这方面的突破需要我们与其它领域紧密合作，包括将编译器与编程语言、库以及运行时环境更紧密地集成，以便获得为程序的并行执行做优化所需的程序语义信息。那种对“整个程序”做分析的传统方式已经无法满足要求，需要开发一种分层的软件开发和优化方案，其中在优化一个软件层的代码时无需分析更低层次的代码。这种抽象方法要求在每层都有定义良好并带有语义的应用程序接口(Application Programming Interface，API），其语义能描述该层信息模型或是本体特征（例如谷歌的Map-Reduce编程模型）。编译器可以利用这些语义在更高层次上进行软件优化。反过来，来自高层的上下文信息也可以用于低层代码的专用化或优化。由于来自程序员的指导仍然必不可少，因此还需要开发交互式或是半自动的基于编译器的工具。一个值得注意的例子，是研究人员利用检测数据竞态的软件交互式完成对Intel IA32编译器的并行化的项目[1]。交互性可能需要将编译器结合到集成开发环境里，重新设计编译器的优化方法，使之减少对各种优化应用的顺序的依赖，还要重新设计编译器算法和框架，使之更适合交互式的环境。这种类似Java语言编译器的即时编译方法模糊了编译时和运行时的界限，给程序优化提供了新的机遇，使之能在动态程序运算赋值的情况下进行。由于并行客户端应用程序的出现，运行时的依赖关系检查和优化对于处理动态数据结构（例如图结构）的程序可能会更为重要。

开发针对特定体系结构的优化的严格方法

20世纪60至70年代，编译器前端大大受益于基于自动机理论的词法和语法分析技术的系统化理论的发展。然而，正像前文引述的大卫·库克所言，目前还没有针对特定体系结构进行优化的系统化方法，而这大大阻碍了并行和优化编译器的开发。为了开发有效的全局优化策略，尤其是当程序性能依赖于输入数据时，要求程序员去处理大量的交互式的变换、大量非线性目标函数以及大量的性能预测，显然，程序优化带来的挑战异常困难，但障碍也并非不可逾越。

围绕这一问题，最*很多研究机构都做出了相当大的贡献，例如卡内基梅隆大学、加州大学伯克利分校、伊利诺伊大学、麻省理工学院以及田纳西大学，他们的研究展示了基于离线的经验搜索进行针对特定体系结构的性能调优

方法的潜力。这种技术称为"自动性能调优"，在线性代数和信号处理等经过深入研究的领域中，这种方法的效果可以与手工调优的效果媲美。基本方法是让程序员标注出或者让编译器生成出一个程序的各种不同实现的良好定义的搜索空间，然后由编译器和运行时工具对其进行系统化探察，这样的优化过程能得到与手工调优媲美的结果。这里的难点是如何将其扩展到并行系统和多核处理器中，以及如何将这种技术融入到能够用于大多数复杂应用的、具有内在一致性的简单易用的系统中。构建于上述具有自动调优特性的线性代数和信号处理库例子之上的，支持自动调优的语言和编译器技术，会对开发更多领域中的数值和符号算法库有极大帮助。这些库里封装了只有少数编译器专家才能实现的高效算法，并将其提供给广大普通程序员使用。此外，通过展现新鲜而有趣的表达计算的模式，这种库还对未来高级语言和高层次抽象的开发有借鉴作用。更进一步，通过用机器可识别的格式描述这些库的特征，使编译器能掌握这些库函数的语义性质和性能特征，将有利于实现能对代码中库函数调用做出合理化建议的交互式工具。

对编译器研究人员来说，一项更重要的挑战是如何将这一方法转化为在线调优。通过在实际机器上运行程序的不同版本以评测其性能并不具备可扩展性，无法将其扩展到更多的内核或更庞大的程序，哪怕用离线模式也不行。对于编译器研究者，一个策略是开发易于操控的硬件和语言抽象，使对程序不同实现的性能评价更为高效，毕竟我们需要的只是这些不同实现的相对性能，而非绝对运行时间。虽然在很多文献中提及了并行计算模型（例如并行随机访问机器RAM（Parallel Random Access Machine,）以分析并行算法以及LogP模型以分析通信），但对于作为编译器性能优化的有用目标而言，它们都显得过于抽象。通过与理论和体系结构领域专家的交流，我们认为应该开发出能够更准确描述今天的多核*台的新模型。该模型还可以成为程序并行化和针对性能、功耗、吞吐率以及其它指标进行优化的系统化理论的基础。

正确性和安全性

检查程序中的编码缺陷一直都是编译器的一项重要任务。在帮助程序编写方面，高级语言既能简化计算的表达方式，也能帮助识别各种常见的编程错误，包括使用未声明的变量，以及通过日益复杂的类型检查机制能检测的各种语义错误。但要希望编译器帮助避免错误的运行结果以及各种安全隐患，我们还能而且必须在程序分析方面做更多工作。

关于软件安全性，微软公司可信计算组的安全工程策略高级主任史蒂夫·利普纳（Steve Lipner）在一次私下交流中谈到："自2002年这一可信计算项目启动以来，微软开始特别强调其软件的安全性。相关的软件分析工具对这方面工作的成功至关重要，这些工具可以在软件发布前帮助工程师检测并剔除出各种安全隐患。目前微软通过安全开发生命周期（SecurityDevelopment Lifecycle，SDL）来对其工程实践中的各种安全措施形式化，强制性地要求使用程序分析工具和（编译器的）各种安全增强选项。这些工具和编译选项是多年来程序分析和编译器领域研究工作的结晶，对解决工业界所面临的安全挑战是无价之宝。微软的安全团队正翘首以盼，希望利用编译器技术的进一步研究成果和高效工具的相关进步，来保证其软件产品更为安全。"

程序正确性和安全性检测工具必须在保证软件可靠性和安全性的前提下尽量减少误报，以避免浪费程序员的时间，而且应能在显示检测结果时排出优先顺序，并加以分类。此外，这些工具还不能对程序的性能和易用性产生消极影响。同应对其它挑战一样，只有通过编译器、语言、库以及运行时系统的紧密结合，才能给出效果最好的解决方案，尤为重要的是，我们认为针对特定问题领域定制的程序分析系统将扮演重要角色。这个领域中最重要挑战就是除错，它一直被称为"Debugging"。其任务是开发相关的工程技术来帮助检测和避免程序缺陷；第二个挑战是安全风险，其目标是开发相关的策略来检测程序对外部攻击的抵御能力；最后一个挑战是开发自动的程序验证技术。

让软件开发像飞机一样可靠

根据上文提到的美国国家标准与技术研究院报告，通过减少程序缺陷以提高软件质量，不仅给计算机科学领域提出了许多的研究课题，更对美国的整体经济有着深远影响。事实证明，编译器技术中静态和动态的程序分析对于识别复杂错误非常有用，但我们要做的研究还很多。为了能够拓展这方面的工作，我们需要新的程序分析策略，它们应能改进软件的构建、维护和演化技术，并让程序开发过程能够达到最高的工程标准（就像汽车生产、航空航天以及电气工程）。

一个行之有效的策略可能会涉及各种分析技术，能够提高生产率和可靠性的新语言特性，新的软件开发范型。不过，处于这些工具和策略核心的依然是先进的编译器以及程序分析技术，它们能在保证所生成代码质量的同时给出符合真实情况的结果，其中衡量质量指标的可能包括执行时间、功耗以及代码规模。除了能生成更加可靠的程序外，这些研究产生的工具还可以使编程专业工作更富成效，因为它们将使开发人员和测试人员可以更集中于那些工作中最有创意的部分。让系统软件的每一个层次都变得安全*年来，为了解决软件的可靠性问题，人们用复杂的程序分析和变换技术来检测程序，以避免由代码缺陷引起的软件脆弱性（如缓冲区溢出以及悬空指针）。由于任何脆弱性都可以认为是软件缺陷，所以检测和避免软件脆弱性方面面临的挑战性问题与前面讨论的软件可靠性有重叠的地方。然而，它们之间的区别在于安全性策略需要在分析和变换的设计中考虑可能的外来攻击。故而某些技术（例如系统调用认证和防SQL（Structured Query Language，结构化查询语言）注入）只用于计算机安全。编译器在加强计算机安全方面扮演着至关重要的角色，因为它可以减少由于编码错误带来的脆弱性，为程序员提供一些工具，使错误的检测和避免能够自动化。最有效的降低风险的策略可能是用具有强类型检查特性的语言编程。与命令式语言的程序相比，函数语言的程序具有更透明的语义，所以程序语言研究者早就说函数语言是降低软件脆弱性的最佳解决方案。当然，可能需要为其扩展事务语义，以处理可变的状态。

自动验证完整的软件栈对程序是否符合规格进行形式化证明（程序验证），是彻底避免软件缺陷的强有力的策略。一个佐证就是长久以来程序验证都被认为是计算机科学中最大的挑战之一。40多年前，约翰·麦卡锡（John McCarthy）（时为斯坦福大学计算机科学教授）写道："我们应该应当用一个计算机程序自动证明一个程序确实满足其规格，而不是去排除其中的错误。"[2]虽然程序验证传统上不被认为是编译器领域的挑战，但我们在这里提出它，是因为它很有可能成为上述两个挑战的形式化解决方案：程序分析、自动验证以及日益加深的对编译器验证的兴趣，这三者正在相互影响。验证编译器的代码和算法可能成为应对挑战的一个良好开始，这里有2个原因。一是编译器包含了对其自身的正确性的规格要求，这使验证过程有清晰的需求。是，对编译器生成代码的验证是通常软件

验证中的一个必要方面。*来编译器验证方面的进展让人看到，未来我们有可能依靠源代码层上的形式化机器证明。这里的最终目标是证明编译器是正确的（输入输出的等价性保证），并且符合对时间、空间和能耗的要求。

强大而高效的验证工具将是对计算实践的重大贡献，但程序验证的重要性远不只限于提高软件的质量。作为形式化推理系统的一个实例，程序验证本身具有重要的智力价值。机器学习方面的研究人员认为，对于开发高级的推理系统，程序验证是一个理想的课题。毕竟，编程是一个研究推理系统的计算机科学家需要真正理解的问题。建议为了应对上述各种挑战，编译器领域需要一个由全球工业界、国家实验室和大学联合参与的活跃的研究计划。编译器技术的进步需要所有研究人员个人的创造力、热情和能量。但考虑到编译技术和软件系统的复杂性，为了获得成功，长期的研究项目应由工业界和研究机构的编译器专家来领导。

为此我们提出四项主要建议：

促进编译器研究中心的设立 寻遍整个世界，目前也找不出几个大型的编译器研究团队。大学里的编译器研究组一般由一个资深研究人员和几个学生组成。他们的项目一般是短期的，通常也就是一个博士课题的时间。另一方面，工业界中为数不多的几个高级编译技术研究团队也倾向于关注短期的解决方案，虽然编译器、其所分析和翻译的程序以及目标运行环境的复杂性都在不断增长。结果是，今天大多数编译器的研究都在关注很窄的问题，具有革命性的策略，则因需要长期的开发和评价而被忽略了。目前在一些编译器研究方向上（例如程序分析和优化），这种增量式的研究所带来的回报已经越来越少。研究人员必须去大胆尝试全新的解决方案，而这些方案可能是长期的和复杂的。编译器研究领域（大学和工业界）需要建立研究中心，或开展若干大规模的项目，来支持稳定队伍。工业界和资助机构需要共同采取措施，促进研究中心的建设，并为其创造机会。工业界和政府的资金也需要支持那些正在进行的变革及相应的软件维护工作。

大学/工业界/政府部门需要在开发基础设施及收集基准测试程序方面建立长期合作关系，同时资助大学和其它研究中心的独立项目 对编译器领域而言，实验和实现是非常重要的。无论是新的技术和工具，还是已有方法的新实现，只有当它们被结合到具有工业质量的编译器基础架构中，并采用最新的优化方法以及代码生成策略后，才能得到有意义的评价。缺乏广泛接受的研究*台，长期制约着编译器技术、新编程语言设计以及新体系结构相关优化的研究工作。开发完整的编译器需要大量工作，通常无法由单独的研究组完成。GNU（一个自由软件工程项目）编译器以及其它一些工业界开发的编译器（例如IBM的Java Jikes编译器、英特尔的Open Research Compiler（现为Open64，译者注）、以及微软的Phoenix框架）开放了源码。它们有可能进一步发展为所需的基础架构，但目前还没有一个能满足本领域的所有要求。

实验性研究需要用能代表各种重要应用的基准测试程序进行评估，这意味着收集具有代表性的程序将是一个持续不断的过程。在收集测试程序上我们已经投入了大量精力，但所得到的集合仍然很有限，并且人们也一直在置疑它们的代表性。这当中一个难题是很多广泛使用的程序都有版权保护。在一些领域中开源软件可能可以代表有版权的软件，对于评价编译器而言，在这些地方使用性能超过相应版权保护版本的开源版本就足够了。这些测试基准程序还需要附带有代表性的输入数据集以及相关算法和数据结构的描述，这些信息可以帮助开发更适用于表达这些计算的新型编程语言及其扩展特性。

联邦机构和工业界需要共同建立所需要的资金和激励机制，以鼓励大学和工业界的编译器专家去解决基础设施和测试基准的问题。与此同时也要给较小的学术研究机构和开发者以资助机会。计算机科学界已经在开发全新的编译器基础设施方面取得了不小的成功，有关成果包括：编译器前端开发，为编译器研究搭建的程序分析框架，验证工具、安全应用程序、以及各种软件工程工具，还有为主流机器和专用语言开发的虚拟机框架。虽然GNU和工业界开发的编译器对研究来说都是有用的基础设施，但是，为了未来发展的需要，开发新型、健壮以及对有关研究人员而言更加易用的基础设施是非常必要的。其重要性不仅在于支持相关的研究课题，也应当认识到它们的学术价值。开发使人们可以更方便地比较不同技术途径并重现实验结果的方法学和资料库对于策略、机器、语言以及问题领域的比较来说，实验的可重复性非常关键。当今能够得到的信息很多都没有可信的依据。在很多其他学科，评审人员和相关科学家们都要求论文包括足够的信息，以便其它研究组可以重现论文中的结果。但编译器方面的论文中往往无法充分地描述这些实验，因为许多实现方面的细节决定了结果。现在有了开源软件的帮助，我们可以通过互联网发布软件和报告中用到的有关数据。主流的会议和组织（例如ACM）应当提供机制以发布软件以及实验数据，作为报道这些实验的文章的元数据。这种资料库可用于评价技术的进步，而且对于那些对技术历史感兴趣的人来说也是很有用的资源。

开发与编译技术相关的课程

编译技术非常复杂，这一方向的发展需要睿智的研究人员和实践工作者。与此同时，计算机科学已经成长为融汇了无数令人神往的研究内容的领域。编译界需要将编译的重要性和它的美妙性传承给一代又一代的学生们。编译器课程必须向学生们清楚地展示其特别的重要性、广阔的应用前景以及其作为计算机科学技术必不可少基石的自身的优雅特征。

以前编译技术是多数本科教学计划中的一门核心课程，而现在很多学院都将其作为计算机科学与工程系高年级学生的选修课程，其中常常包含一些有趣的课程设计，使学生获得重要的实际经验。在一门优秀的高层次编译课程里，学生通过构造比较大的能完成有趣而实际功能的软件，学到数据结构、算法以及相关工具的综合性知识。然而，无论对于教师还是学生，这样的课程都会被认为过于复杂，而学生通常还有其它兴趣选择。这样一来，往往只有很少的学生能真正接触到编译器的基础理论，或者在研究生期间选择编译器作为研究方向。任何对编译器实现和机器实现感兴趣的人都会认为编译算法具有极高的教学价值。有关编译算法的能力和局限性的知识，对编译器、排错工具和任何使用编译算法构造的工具的用户都极有价值。这些算法里封装了许多（就算不是大部分）用于保证程序正确性和性能的重要程序分析及变换策略。所以说，学习编译算法是一种学习程序优化和常见编程错误的严谨方法。基于这些原因，拥有深厚编译器背景的程序员的专业表现将更为出众。提升有关编译算法知识重要性的一种方式，是在各种计算机科学课程中，例如在自动机理论、程序设计语言、计算机体系结构、算法、并行编程以及软件工程等课程中，讨论一些特定的编译算法。这方面的挑战性问题是如何定义出所有计算机专业学生都应该知道的有关编译器的关键概念，并提出在计算机科学的核心课程中应该包括的内容的建议。另一补充措施是开设一些高级课程，其中重点关注软件工程、科计算以及安全性问题中基于编译器的分析。这些课程可以假设学生们已经在核心课程中学习过编译器的基本概念，因此可以很快切换到各种新课题（例如基于编译技术的病毒检测）中。这方面的挑

战性问题是如何设计这些课程，使学生能在高级编译器技术领域得到训练，并将其应用到各种有趣的相关领域（例如软件可靠性和软件工程）里。这些课程可以不用“编译器课程”来冠名，而是根据其目标领域予以命名（例如计算机安全、验证工具、程序理解工具等），也可用如程序分析和维护等更为通用的名称。

结语

虽然编译领域的工作已经使计算科学发生了沧海桑田般的变化，但我们依然面临为数众多的难题，现在又出现了许多新的挑战（如多核编程）。这些未解决的编译器挑战（例如怎样提高并行编程的抽象层次，开发安全而健壮的软件，以及验证整个软件栈）在实际应用中占据着非常重要的地位，同时也是当今计算机科学中最具挑战性的难题。为了解决这些问题，编译领域必须开发新的技术，使得编译器领域在经历了50年的阔步前进之后“百尺竿头，更进一步”。计算机科学教育工作者需要向学生展示编译器深厚的智慧基础，强调其在计算机科学中的广泛应用前景，以期吸引一些最聪颖的学生到编译器领域来工作。该领域所面对的另外一些挑战（例如缺少可扩展且强大的编译器基础设施）则需要整个编译界共同努力才能克服。同时，我们还必须让资助机构和工业界知道这些问题的重要性和杂性，并为解决这些问题投入长期的人力和物力资源。

作者：Mary Hall：犹他大学计算机学院副教授。mhall@cs.utah.edu
David Padua：伊利诺依大学乌尔巴尼-香槟分校计算机科学Donald Biggar Willett教授。
padua@illinois.eduKe sha v Pi nga l i：德克萨斯大学奥斯汀分校计算机科学系网格和分布式计算教授的W.A. “Tex” Moncrief 主席以及奥斯汀分校计算工程与科学研究所教授。pingali@cs.utexas.edu



标签: 编译器





« 上一篇： undefined reference to 'dlopen';undefined reference to 'dlclose';undefined reference to 'dlerror'等问题
» 下一篇： 体系结构-指令集结构