黄维

复旦大学科技处

中国，上海200433，邯郸路220号

尊敬的黄维先生，

从2011年第三季度起，英特尔公司很高兴的资助复旦大学软件学院的唐渊教授USD$43,285，以支持唐渊教授在“增强Pochoir Stencil编译器”领域的工作，该工作详见附件的研究提议。

审议这项拨款的大学将特此授予英特尔非独家的，不可撤销的，全球范围内的免版税的许可证，没有牌照费或其他需要英特尔支付的费用。英特尔并将同时获得再许可的权利，可以使用，销售，和进口任何由唐教授或他的实验室的成员在本项资助下所做出的研究及发明。取得这项资助的大学需同意授予英特尔公司的许可证是非排他性和免版税的，授予英特尔公司的许可证将允许英特尔公司对由该项目资助的研究成果进行任何形式的使用，包括任何商业用途。对该项目的拨款将不会扣除任何开销的费用。

请让具有适当授权的学校代表签署本信，以表明大学接受这些条件。

大学同意对于这笔款项下提供的资金的使用将遵守所有相关的美国出口法律和法规。

英特尔公司  
英特尔实验室  
Mailstop：SC12- 303  
2200 Mission College Blvd  
Santa Clara，CA95052

当我们收到回信后，我们将向你们拨出这笔款项。回邮地址是：

Carolyn Nelson

Intel Corporation

Malstop: SC12-303

2200 Mission College Blvd.

Santa Clara, CA95052

Email: [carolyn.nelson@intel.com](mailto:carolyn.nelson@intel.com)

如果您有任何意见或建议，请联系Erin Richard (408-765-1862)

真诚的，

Robert Cohn

执行经理，Tools, Pathfinding & Innovation

Developer Products Division

Software and Solutions Group

同意并接受：

复旦大学：

签名：

姓名：

职称：

日期：

教授姓名：

签名：

英特尔公司  
英特尔实验室

Mailstop, SC12-303

2200 Mission College Blvd,

Santa Clara, CA95052

# 增强Pochoir Stencil编译器

### Charles E. Leiserson

### SuperTech Research Group

### MIT Computer Science and Artificial Intelligence Laboratory

### Cambridge, MA 02139

### 联系方式：[cel@mit.edu](mailto:cel@mit.edu)

### 唐渊

### 复旦大学，上海市邯郸路220号，200433，中国

### 联系方式：[yuantang@fudan.edu.cn](mailto:yuantang@fudan.edu.cn)

### 执行摘要：

Pochoir1.0 Stencil编译器为基本的stencil计算产生高性能的Intel Cilk Plus代码。我们建议，加强Pochoir编译器的能力，并探讨如何进一步推广现有的Pochoir方法。我们的工作将主要包括以下六个任务：

1）我们建议增强Pochoir编译器的能力以处理更复杂的非规则的stencil计算。

2）我们建议增强Pochoir编译器的能力以处理在stencil计算中更一般的数据依赖关系，比如说在动态编程（dynamic programming）中的数据依赖关系。

3）我们建议进一步拓展并简化Pochoir的stencil描述语言。

4）我们建议进一步优化Pochoir的核心算法以及数据结构。

5）我们建议在其他的计算平台上也开发相应的Pochoir stencil编译器，比如分布式存储体系结构，GPGPU，未来的exascale计算平台。

6）我们建议和英特尔公司联合研究如何将Pochoir stencil编译器中的技术嵌入到英特尔的C/C++编译器中。

以下研究计划中所列举的目标将远远大于我们在一年的时间中所可能做的工作。但我们只要求从英特尔公司得到第一年的资金支持，并计划申请自然科学基金以支持本项目后续的研究工作。

### 背景：

Pochoir（发音为“PO- shwar”）是一个开源的编译器和运行时系统，是在多核处理器上针对stencil计算进行优化的一个编译系统。该编译系统的目标语言是Intel Cilk Plus。Pochoir最初是作为麻省理工学院和英特尔公司之间的一个联合项目进行研究与开发的。麻省理工学院从英特尔公司得到了慷慨的捐赠。

stencil计算的定义是指在一个d维空间网格对每一个网格点的在时间步t的值的更新是一个以前任意时间步的网格点的值函数[1，3-5，7，8，10-15，17，19]。stencil计算最简单的实现是使用嵌套循环，但循环实现对高速缓存的利用效率很低，性能很差。缓存无视（cache-oblivious）[6,16]分而治之的计算方法具有更高的缓存效率[7,8]，但这些递归代码非常难写，而当代码中需要进一步融合并行分解的技术时，大多数应用程序的程序员不具备编程技巧或耐心生产高效的多线程代码。

Pochoir stencil编译器允许程序员编写一个简单的声明规范来描述其特定的stencil计算，该声明规范是一个嵌入式的C++语言。Pochoir然后将其转换成高性能Intel Cilk Plus代码，并采用了有效的并行缓存无视（cache-oblivious）算法。 Pochoir支持一般d维空间，并使用同一个算法处理周期和非周期的边界条件。 Pochoir系统提供一个C + +类库，它允许用户的声明规范可以在不经过Pochoir编译器的预处理而直接被任何标准的C++编译器和运行时系统执行（虽然速度会更慢），从而便于用户调试，并同时大大简化了Pochoir编译器本身的实现。

Pochoir产生的代码可以媲美的最好的手工编写的代码，包括伯克利的自动代码优化器所产生的代码[2，10，19]的。一系列的标准stencil计算测试表明Pochoir编译器产生的代码优于标准的并行循环实现，通常运行速度会快2-10倍，比标准的串行循环实现快20-100倍。

Pochoir包括许多新的技术以获得高性能的stencil计算的代码。Pochoir背后的算法比原先的缓存无视（cache-oblivious）stencil算法对于多维网格来说有渐进的提高（asymptotically faster），其主要思想是采用了在多维网格的“超空间”的分割，以在缓存利用率相同情况下得到更高的并行度。 Pochoir的递归缓存无视算法采用了两个代码克隆：边界克隆，它可以处理复杂的周期和非周期的边界条件，包括的Dirichlet和Neumann条件;和内部的克隆，这是一个经过高度优化的代码克隆以处理众多的内部网格点，其最大优点在于不需要任何特殊的边界处理。 Pochoir生成的代码包括高度优化的迭代器遍历网格点。 Pochoir在递归算法中使用了简单而有效的启发式条件以帮助算法的加速。

### 一个新的Pochoir Stencil编译器

我们现在简单介绍一下我们打算研究并开发的一个新的Pochoir stencil编译器的框架设计：

### 非规则的stencil计算：

Pochoir 1.0编译器可以处理简单的stencil计算，但现有和潜在Pochoir用户报告Pochoir编译器在处理各种非规则的stencil计算时是不足的。

Pochoir目前不能处理非规则边界条件。对于非规则边界条件的处理，目前Pochoir都假定其基本网格形状是长方形。然而，对于某些应用，包括那些在生物科学，材料科学和机械工程领域的应用，其边界形状都可能是一个非矩形的形状。虽然可以在非规则区域外面套一个矩形区域，但是对于在所关心的非规则区域外的点的计算将会浪费大量的计算资源。此外，若是将边界条件的判断放置在内核计算函数内部，由于其属于不可预测的条件转移指令，当预测出错时，其开销将会远远超出内核计算函数中实际有用计算所占用的开销。

另一个非规则stencil计算来源于计算域内部。例如，一些波动方程的应用，不同的空间区域可能对应不同的材料或不同的吸收边界，因此对应一个不同的内核函数。同样的，如果通过在函数内部放置条件语句来实现非规则计算的话，将会显著降低计算的效率。

有些stencil计算具有一个跨越式 (leap-frog) 的结构，其中的stencil计算在不同的时间步执行不同的操作。计算仍然可以是规则的，但每隔一个或若干个时间步将重复相同的计算顺序。虽然目前的Pochoir编译器可以通过在一个大的内核函数中加入条件语句的形式来实现这种跨越式的stencil计算，但是，同样的，这种类型的解决方案将牺牲周边网格点共享公共子表达式的能力，牺牲了计算的效率。

所有这些非规则的stencil计算有一个共同的性质：每个点在网格中的位置决定了对它进行更新时所要执行的代码。虽然有些应用是在计算的动态过程中决定所要执行的非规则代码，但更多的应用是在编译时刻，根据每个网格点的空间坐标静态的决定其更新代码。此外，现在还有一些算法上的策略来处理各种不同类型的非规则stencil计算。比如，跨越式stencil可以通过某种形式的循环展开来处理，而不规则的内部和边界条件则可以通过代码克隆来处理。我们建议在Pochoir编译器中加入一些通用的，有效的策略在编译时刻统一处理这些非规则计算。

### 广义的数据依赖

基本的stencil计算对每一个网格点的在时间步t的值的更新是作为其邻居点在时间步t之前值的一个函数。Pochoir1.0提供了一个简单的stencil描述语言来指定这些基本的stencil 计算。有一些应用中所展现的数据依赖和基本stencil计算中的相似，但不完全相同，现有的Pochoir并不能很好的支持。 在这里，我们将讨论三个这样的新的数据依赖，并打算将它们整合进新的Pochoir描述语言。

比如，Lattice Boltzmann Method (LBM)在第一阶段的计算反转了的一个基本的stencil数据依赖的方向。该计算是使用每个网格点在时间步t的值去更新所有在时间步t+1及以后的网格点的值。如果说基本的stencil计算的数据依赖关系是“拉 (pull)”关系，那这种反转了的数据依赖关系则可以被称为“推 (push)”关系。尽管，可以通过一定的改写，将所有的“推”关系转换为基本的“拉”关系，但是，这样的工作是繁重而容易出错的，也是和原有的计算语义不完全相符的。所以，我们打算将Pochoir的stencil描述语言进行拓展，以便能够较好的支持这种少见的“推”关系。

许多动态编程的问题有这样一个结构，每个网格点（X，Y）的值取决于其在两个不同方向的邻居点的值，比如（X - 1，Y）和（X，Y- 1）。无论是X，还是Y，都无法提供一个合适的时间轴，以执行一个基本的stencil计算，因为无论采用哪个方向，都将产生网格点(X, Y)在时间步t的值将取决于相同时间步t的其他网格点的值。这种数据依赖关系，在目前的Pochoir中无法支持，因为现有Pochoir编译器背后所使用的缓存无视算法最主要的想法是打乱同一个时间步中各个网格点的计算顺序，以达到一个较好的缓存利用率的目的。就目前Pochoir的解决方案来说，可以采用将t= X + Y作为一个时间轴，将原先的坐标轴旋转45度的方法来处理一个菱形的边界条件来解决这个问题。原则上，这种计算可以在不依赖于梯形分解的情况下，仍然有效的通过递归方式执行。但显然，其对于普通程序员来说是很不方便的。我们计划拓展Pochoir的stencil描述语言规范以使得程序员们不需要考虑需要将坐标轴旋转的情况。

一些stencil计算的数据依赖既可以依赖于当前时间步的数据点的值，也同时依赖于之前时间步的数据点的值。这样的情况类似于动态编程的情况。在新的Pochoir中，我们也将提供一个通用的解决方案。

### Pochoir的stencil描述语言

除了在算法层面来处理非规则stencil计算，Pochoir规范语言必须重新设计，使用户可以很容易地以这种方式来指定在时空网格中较罕见的各种非规则stencil 计算，并且使得Pochoir编译器可以很容易地利用更大范围内内规则的stencil计算以提高运算效率。现有的Pochoir stencil描述语言对于内部和边界之间的区别将为我们预期新的Pochoir语言提供设计灵感。

目前，用户在指定核心函数的同时，还需要同时指定一个边界函数，以处理边界条件。根据用户的指定，Pochoir编译器会相应生成一个边界代码克隆和内部代码克隆。我们可以进一步拓展这个思想，允许用户对每一个非规则区域设定一个特定的内核函数来专门处理所有在该非规则区域中的网格点。

在我们未来的设计中，用户可以指定一个的保护条件，描述了对某个时空网格区域应该采用的特定计算方法。新的Pochoir编译器将收集所有的这些保护条件并使用符号计算预先确定在缓存无视算法中所用到的递归树，这样，在随后的正式计算中就可以除掉所有的对时空区域的条件判断语句，以达到一个较优的执行效果。

### 核心算法和数据结构

我们计划继续为Pochoir改进现有的基础算法。在算法层面上，我们将探讨递归展开，重叠时间步倾斜的粗化策略，改进的分而治之的切割策略，和其他几个算法优化策略。在实施层面，我们计划将深入探讨使用缓存无视的数据结构来存储stencil计算中所需要的各种数据类型，自动生成的存储阵列光晕，启发式自动调优，采用锯齿状的遍历策略，并使用数据位的技巧，以避免不可预知的分支转移语句，等等。

粗化是对递归程序的一个基本优化。粗化的想法是在递归树的更高一层的结点上更早的终止递归，以避免了大量的递归函数调用的开销。由粗化产生的一个问题是基本情况 (base case) 中将无法使用递归顺序来遍历所有网格点，而递归次序的遍历是对于缓存利用和寄存器分配最优的遍历次序。相反，在基本情况中将只能使用循环的线性次序，线性遍历次序对于处理器流水线而言是最优的。我们计划调查如何展开递归以使得遍历同时对缓存和处理器流水线达到最优。

我们相信，我们可以改进的特定分而治之的递归方法以提供更高的并行度。目前，stencil计算的迭代空间被划分为一个个的超梯形 (hyperzoids) ——梯形在多维空间的类比。我们目前的分而治之的策略，是基于Frigo和Strumpen的思想基础[7，8]，提供了一个渐近最优的缓存效率。我们最近的一篇文章[18]表明，渐近更高的并行度是可能的。该方法业已在Pochoir1.0中予以实现，并取得较好的效果。我们相信，在这个实现中，我们可以进一步的提高一些常数因子，以达到更好的执行效率。关键的思想是每次对超梯形的分割都以较短的底边为基准，而不是原来算法中的以较长的底边为基准。这一策略可以减少超梯形在时间轴上的切割，从而提高并行度。

对于那些具有恒定边界条件的stencil计算，一个普遍采用的优化策略是在  
（在pochoir中没有使用）所有stencil定义的区域周围存储一圈光环 (halo points)。该光环可以设置为一个固定的边界值，邻近的网格点可以直接存取这些光环值而无须通过执行不可预知的分支转移预测语句。Pochoir不使用这一优化策略，因为它已经使用了代码克隆的技术来优化对边界内外不同区域内网格点的更新。对于大规模的stencil计算，光环策略不会比Pochoir所采用的代码克隆技术取得更优的性能，却要求对边界区域的网格点采取一个特殊处理，因为边界区域是渐近比内部区域体积要小。而对于中小规模的stencil计算，光环优化策略可以产生显着的性能收益。我们相信，我们可以实现让Pochoir对恒定的边界条件自动产生光环代码，而无须用户的干预。

由pochoir1.0自动生成的代码包含调优参数，这些调优参数由经验而得。我们也已经将Pochoir和英特尔的软件自动调优工具 (ISAT)相连接，ISAT工具可以自动在所有参数的空间进行穷举搜索。最近，启发式的自动调优器[9]可以提供性能更好的调整启发式搜索，而不是穷举搜索。我们也正在计划研究如何将启发式调优搜索纳入Pochoir。

### 拓展Pochoir的应用范围

Pochoir 1.0是设计用于多核计算机体系结构的。我们提议将Pochoir拓展到其他体系结构的计算平台，比如，分布式内存的集群系统，图形处理单元 (GPU)，以及未来的exascale计算机。

特别的，我们提议将Pochoir中使用的缓存无视算法从单个节点的多核机器拓展到多核集群系统。主要的想法是利用MPI来进行节点间通讯，而使用Intel Cilk Plus进行同一个节点内部的通讯。我们相信，我们可以利用一个二级结构的设计来实现这个目标。其底层结构将直接调用现有的多核Pochoir，而高层结构将是一个全新的MPI实现，其目标语言将由底层的Pochoir编译器解释执行。比如说，高层的Pochoir所产生的代码将可以在内核函数及边界函数中调用MPI来实现必要的通讯功能。

GPU及向量计算硬件为Pochoir提供了另一种机会。对于GPU上的应用，需要Pochoir针对其特定的体系结构产生相应的递归结构。这里一个主要问题是向量化的循环结构将比分而治之的递归结构占用更多的内存带宽。这样，尽管在单个处理器核心上，使用向量化代码可能取得更好的速度，但在多个处理器核心上，向量化代码的速度反而可能更慢。一个有趣的可能是根据目标体系结构上处理器核心的多寡，自动选择代码向量化的程度。我们计划试验该自适应方法。

在exascale的计算机上，一个很重要的问题是容错。我们提议将研究如何在Pochoir中开发应用程序级的自动化检查点和容错技术。这里的一个主要问题是Pochoir中所使用的缓存无视算法对于缓存的使用是不均匀的，这点是和标准的循环代码不同的。这样，从一个检查点恢复一个stencil计算所需要的信息量就可能要超出相同的循环代码实现。我们计划研究如何将检查点和容错技术集成进Pochoir。

### 将pochoir嵌入Intel编译器

如果能够得到一个工业化标准的编译器的支持，我们相信目前Pochoir所使用的stencil描述语言可以得到进一步的简化，并更具有直观性。基本想法是定义一个C++的Pochoir类。当需要实现一个特定的stencil计算时，程序员将定义一个该Pochoir的子类，直接继承父类中的基本功能，并只对该stencil计算特定的一些参数，比如不同的内核计算函数，边界处理函数定义相应的代码。

对于这个方向，绝大多数的工作需要由Intel的编译器团队来实现。我们计划在语言设计，算法，运行时系统方面提供必要的支持。

### 工作声明

本研究项目将由MIT的Charles E. Leiserson教授作为研究小组组长领衔运行，复旦大学的唐渊教授作为主要参与者进行。唐渊教授在MIT做访问学者期间，曾参与了Pochoir 1.0项目的主要工作。在本项目的绝大多数时间里，唐渊都将在中国参与该项目的研究与开发工作。Charles E. Leiserson教授和唐渊教授将定期使用远程电话或者视频会议进行交流。在该项目期间，唐渊教授将访问MIT一次。该项目的主要参与者和贡献者还包括，Intel的Chi-Keung Luk，MIT的Bradley C. Kuszmaul博士，MIT的Steven G. Johnson教授，纽约州立大学石溪分校的Rezaul Chowdhury教授。另外，在MIT还将分别有一名工程硕士和一名本科生参与本项目的研究与开发工作。

目前，我们仅要求第一年的资金支持。我们打算在所有以上六个研究目标展开工作，并在本年度结束的时候发布一个主要的软件版本。另外，我们还将寻求自然科学基金对本项目的后续资金支持。

本项目的主要重点在于任务1--4。正如我们在以前其他一些研究项目中所做的，比如Cilk，FFTW，和Pochoir 1.0，我们将继续致力于使Pochoir的代码成为开源软件。我们计划与计算科学家联手，使得Pochoir能够满足计算科学家们的实际需求。我们还将发布一个Pochoir的实验版本以生成任务5的一些实验数据。特别的，我们认为任务5可以单独列出，成为一个学生的研究子课题。我们将继续和我们在Intel的合作者一起，探讨在任务6以及自动调优方面的合作。

### 参考文献

[1] R. Bleck, C. Rooth, D. Hu, and L. T. Smith. Salinity-driven thermocline transients in a wind- and thermohaline-forced isopycnic coordinate model of the North Atlantic. Journal of Physical Oceanography, 22(12):1486–1505, 1992.

[2] K. Datta. Auto-tuning Stencil Codes for Cache-BasedMulticore Platforms. PhD thesis, EECS Department, University of California, Berkeley, Dec 2009.

[3] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In SC, pages 4:1–4:12, Austin, TX, Nov. 15–18 2008.

[4] H. Dursun, K.-i. Nomura, L. Peng, R. Seymour, W. Wang, R. K. Kalia, A. Nakano, and P. Vashishta. A multilevel parallelization framework for high-order stencil computations. In Euro-Par, pages 642–653, Delft, The Netherlands, Aug. 25–28 2009.

[5] H. Dursun, K.-i. Nomura, W. Wang, M. Kunaseth, L. Peng, R. Seymour, R. K. Kalia, A. Nakano, and P. Vashishta. In-core optimization of high-order stencil computations. In PDPTA, pages 533–538, Las Vegas, NV, July13–16 2009.

[6] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In FOCS, pages 285–297, New York, NY, Oct. 17–19 1999.

[7] M. Frigo and V. Strumpen. Cache oblivious stencil computations. In ICS, pages 361–366, Cambridge, MA, June 20–22 2005.

[8] M. Frigo and V. Strumpen. The cache complexity of multithreaded cache oblivious algorithms. Theory of Computing Systems, 45(2):203–233, 2009.

[9] F. Hutter, H. H. Hoos, and T. St¨utzle. Automatic algorithm configuration based on local search. In Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI ’07), pages 1152–1157, 2007.

[10] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Implicit and explicit optimizations for stencil computations. In MSPC, pages 51–60, San Jose, CA, 2006.

[11] S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick. Impact of modern memory subsystems on cache optimizations for stencil computations. In MSP, pages 36–43, Chicago, IL, June 12 2005.

[12] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan. Effective automatic parallelization of stencil computations. In PLDI, San Diego, CA, June 10–13 2007.

[13] A. Nakano, R. Kalia, and P. Vashishta. Multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers. Computer Physics Communications, 83(2-3):197–214, 1994.

[14] A. Nitsure. Implementation and optimization of a cache oblivious lattice Boltzmann algorithm. Master’s thesis, Institut f¨ur Informatic, Friedrich-Alexander-Universit¨at Erlangen-N¨urnberg, July 2006.

[15] L. Peng, R. Seymour, K.-i. Nomura, R. K. Kalia, A. Nakano, P. Vashishta, A. Loddoch, M. Netzband, W. R. Volz, and C. C. Wong. High-order stencil computations on multicore clusters. In IPDPS, pages 1–11, Rome, Italy, May 23–29 2009.

[16] H. Prokop. Cache-oblivious algorithms. Master’s thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1999.

[17] A. Taflove and S. Hagness. Computational Electrodynamics: The Finite-Difference Time-Domain Method. Artech House, Norwood, MA, 2000.

[18] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The Pochoir stencil compiler. In SPAA, San Jose, CA, USA, June4 – 6 2011.

[19] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice Boltzmann simulation optimization on leading multicore platforms. In IPDPS, pages 1–14, Miami, FL, Apr. 2008.