August 12, 2011

Wei Huang
Fudan University
Division of Science and Technology
220 Handan Road
Shanghai 200433
P.R. China

Dear Wei Huang,

Effective Q3, 2011, Intel Corporation is pleased to grant $43,285 to the Software School at Fudan University to support the research of Prof. Yuan Tang in the area of "Enhancing the Pochoir Stencil Compiler" pursuant to your attached Research Proposal.

In consideration of this grant University hereby grants to Intel a non-exclusive, irrevocable, worldwide, royalty-free license, with no license fees, or other fees or costs payable by Intel, with the right to sublicense, to make, have made, use, sell, have sold and import any inventions made by Prof. Tang or members of his laboratory in the course of the research that is funded under this grant. The university agrees that its license to Intel Corporation will be non-exclusive and royalty-free, and its license to Intel Corporation will allow Intel Corporation to make any use of the invention, including any commercial use.

No overhead costs will be deducted from this grant.

Please have this letter signed by an appropriate representative of the University indicating the University's acceptance of these conditions.

The University agrees to comply with all relevant U.S. export laws and regulations that apply to the use of the funds provided under this grant.

Intel Corporation
Intel Labs
Mailstop: SC12-303
2200 Mission College Blvd
Santa Clara, CA 95052

When we receive this letter back, we will forward the funds to you. The return address is:

Carolyn Nelson
Intel Corporation
Mailstop: SC12-303
2200 Mission College Blvd.
Santa Clara, CA 95052
Email: carolyn.nelson@intel.com

If you have any further questions or concerns, please contact Erin Richards at (408) 765-1862.

Sincerely,

Robert Cohn
Acting Manager, Tools, Pathfinding, & Innovation
Developer Products Division
Software and Solutions Group

AGREED AND ACCEPTED:

BY: _____

Printed Name: _____

For: Fudan University

Title: _____

Dated: _____

Acknowledgement by Professor:

BY: _____

Printed Name: _____

cc: Prof. Tang

Intel Corporation
Intel Labs
Mailstop: SC12-303
2200 Mission College Blvd
Santa Clara, CA 95052

Enhancing the Pochoir Stencil Compiler

Charles E. Leiserson
Supertech Research Group
MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA 02139
Contact: cel@mit.edu

Yuan Tang
Fudan University, 220 Handan Road
Shanghai 200433, China
Contact: yuantang@fudan.edu.cn

July 22, 2011

Executive summary

The Pochoir 1.0 stencil compiler produces high-performance Intel Cilk Plus code for basic stencil computations. We propose to enhance the capabilities of Pochoir and explore how the Pochoir approach can be generalized. Our effort will be broken into six tasks:

1. We propose to widen Pochoir's ability to handle more complex and irregular stencil computations.
2. We propose generalize the Pochoir approach to handle more kinds of dependencies, such as those that arise in dynamic programming.
3. We propose to generalize and simplify the Pochoir specification language.
4. We propose to optimize Pochoir's core algorithms and strategies for data layout.
5. We propose to explore how Pochoir might be generalized to exploit other architectures and architectural features, such as distributed-memory clusters, graphical-processing units, and future exascale machines.
6. We propose to work with Intel to investigate how Pochoir technology might be incorporated in the Intel C/C++ compiler.

The research proposed below is much larger than the one year effort for which we request funding. We plan to apply for NSF funding to continue the project beyond this one-year time frame.

Background

Pochoir (pronounced "PO-shwar") is an open-source compiler and runtime system we have designed and implemented that optimizes "stencil" computations on multicore processors using Intel Cilk Plus. Pochoir was developed as a joint project between MIT and Intel Corporation. The MIT research was supported thanks to a generous grant from Intel Corporation.

A ***stencil*** defines the value of a grid point in a d-dimensional spatial grid at time t as a function of neighboring grid points at recent times before t. A ***stencil computation*** [1, 3–5, 7, 8, 10–15, 17, 19] computes the stencil for each grid point over many time steps. Stencil computations are conceptually simple to implement using nested loops, but looping implementations suffer from poor cache performance. Cache-oblivious [6, 16] divide-and-conquer stencil codes [7, 8] are much more efficient, but these recursive codes are difficult to write, and when parallelism is factored into the mix, most application programmers do not have the programming skills or patience to produce efficient multithreaded codes.

The Pochoir stencil compiler allows a programmer to write a simple declarative specification of a stencil in a domain-specific stencil language embedded in C++. The Pochoir compiler then translates this specification into high-performing Intel Cilk Plus code that employs an efficient parallel cache-oblivious algorithm. Pochoir supports general d-dimensional stencils and handles both periodic and aperiodic boundary conditions in one unified algorithm. The Pochoir system provides a C++ template library that allows the user's stencil specification to be executed directly in C++ without the Pochoir compiler (albeit more slowly), which simplifies user debugging and greatly simplified the implementation of the Pochoir compiler itself.

Pochoir produces code that is comparable to the best hand-written stencil codes, including ones optimized by the Berkeley autotuner [2, 10, 19]. A host of stencil benchmarks run on a modern multicore machine demonstrates that Pochoir outperforms standard parallel-loop implementations, typically running 2–10 times faster. Pochoir typically outperforms standard serial-loop implementations on a 12-core Intel Core i7 (Nehalem) processor by a factor of 20–100 or more.

Pochoir includes many novel stencil-specific techniques for obtaining its high performance. The algorithm behind Pochoir improves on prior cache-efficient algorithms on multidimensional grids by making "hyperspace" cuts, which yield asymptotically more parallelism for the same cache efficiency. Pochoir's recursive cache-oblivious algorithm employs two code ***clones***: ***boundary clone***, which can handle complex periodic and aperiodic boundary conditions, including Dirichlet and Neumann conditions; and ***interior clone***, which is highly optimized for the more numerous interior grid points that require no boundary processing. Pochoir-generated code includes highly optimized iterators for traversing the grid points with minimal overhead. Pochoir optimizes the base case of the recursive algorithm using simple but effective heuristics.

A New Pochoir

We now briefly outline the issues that we propose to address in a new Pochoir.

Irregular stencil computations

Pochoir 1.0 can handle simple stencil computations, but existing and potential Pochoir users report that Pochoir is deficient in handling various sources of irregularity in otherwise regular stencil computations.

Irregular boundaries cannot currently be handled well by Pochoir. Although Pochoir handles irregular boundary conditions, it assumes that the basic grid is rectangular. For some applications, including those in the biological sciences, material sciences, and mechanical engineering, the boundary may have a nonrectangular shape. Although one can put a rectangular bounding box around the computational region, computing on the points outside the actual boundary can waste significant resources. Moreover, placing conditionals within the stencil kernel function can overwhelm the actual kernel computation, since unpredictable branches cost many processor cycles.

Another source of irregularity occurs within the interior of the computing domain itself. For example, in some wave-equation applications, different spatial regions may correspond to different materials or represent an absorbing boundary and thus require a different kernel function. Once again, putting conditionals in the kernel function can slow the computation dramatically.

Some stencils exhibit a staggered grid or leap-frog structure, where the stencil computation performs different operations at different time steps. The computation may nonetheless be regular, but repeat the same sequence of operations every other or every third time step. Although this kind of computation can be handled by the current Pochoir compiler by writing one large kernel function that does two or three steps at a time, this type of solution sacrifices the ability of neighboring grid points to share common subexpressions.

All these sources of irregularity have the property that the location in the space-time grid determines the correct code to execute. Although there are applications where the irregularity is dictated dynamically by the computation, many more applications have static irregularity predetermined by space-time coordinates. Moreover, there appear to be algorithmic strategies for coping with each kind of irregularity. For example, staggered grid and leap-frogging can be handled by loop unrolling, and irregular interiors and boundaries can be handled by code cloning. We propose to enhance Pochoir by implementing general and effective strategies for handling these static irregularities.

Generalized dependencies

Basic stencil computations update the value at a grid point at a given time t as a function of neighboring grid points at recent times before t. Pochoir 1.0 provides a simple specification language to describe such basic stencil computations, but some applications exhibit a dependency structure that is similar to basic stencils, but which Pochoir 1.0 does not support. We shall discuss three such generalizations which we propose to incorporate in the new Pochoir.

One phase of the Lattice Boltzmann Method (LBM) inverts the normal structure of a basic stencil. Instead of each grid point being updated at time t as a function of grid points at times $t-1$ and before — a ***pull*** stencil — this phase of LBM uses the value of the grid point at time t to update neighboring points at times $t+1$ and after — a ***push*** stencil. Although one can refactor the computation to convert push stencils into pull stencils, this process is onerous and error prone. We plan to extend Pochoir to deal with push stencils.

Many dynamic-programming problems have a structure in which each grid (x,y) point depends on its nearest neighbors in two dimensions, e.g., $(x-1,y)$ and $(x,y-1)$. Neither x nor y provides a suitable time dimension to perform a basic stencil computation, because grid points at time t would depend on other points also at time t. Nevertheless, using $t = x+y$ as a time dimension allows the computation to be viewed as a stencil, albeit by rotating axes by 45 degrees and handling a diamond-shaped boundary. In principle, however, this kind of computation can be performed efficiently in recursive fashion even without a trapezoidal decomposition. We plan to extend Pochoir to handle with this kind of dependency without requiring programmers to shoehorn their specification into the existing form.

Some stencil computations are naturally expressed as depending on grid points at the current time t as well as previous times. This problem is similar to the dynamic-programming issue, and we intend to provide a general solution to both problems.

The Pochoir specification language

In addition to handling irregularities algorithmically, the Pochoir specification language must be redesigned so that the user can easily specify infrequent irregularities in space-time in such a way that the Pochoir compiler can readily take advantage of the presumably larger regular regions. The method by which the existing Pochoir specification language distinguishes between interior and boundary grid points serves as inspiration for the design we envisage for the new Pochoir.

Currently, the user specifies a general kernel function, as well as a boundary function to handle the boundary conditions. The Pochoir compiler then generates a boundary clone and an interior clone of the kernel function. We can generalize this strategy by allowing the user to specify various regions in space-time for which a particular kernel function should be employed.

In our prospective design, the user denotes a ***guard*** condition that describes the region of space-time for which a corresponding function should be employed. The new Pochoir compiler would then assemble all the guards and use symbolic computation to predetermine for the recursion of the cache-oblivious algorithm which subtrees can be executed with no conditionals. It would then generate optimized clones for those regions. Staggered grid and leap-frogging would be handled by unrolling. As long as the vast majority of the space-time grid is uniform and regular, they will dominate the few places where conditionals need to be employed and lead to a highly efficient implementation.

Core algorithms and data layout

We plan to continue to improve existing algorithmic infrastructure for Pochoir. At an algorithmic level, we shall explore coarsening of base cases by unrolling recursion, overlapped time skewing, improving the divide-and-conquer cutting strategy, and several other strategies. At an implementation level, we plan to investigate storing arrays using cache-oblivious layouts, automatic generation of halos, heuristic autotuning, employing a zigzag style of traversal for base cases, and the use of bit tricks to avoid unpredictable branches. We mention only a few of these ideas here.

Coarsening is a fundamental optimization for recursive programs. The idea of coarsening is to terminate the recursion at some point above the leaves of the recursion tree and engage and revert to a highly optimized base-case that avoids function-call overhead. One problem with coarsening is that the base case may no longer traverse the grid points in a recursive order, which is generally the best for caches and register assignment by the compiler. Instead, the focus must be on traversing the grid points in linear order, which is generally better for processor performance. We plan to investigate how unrolling the recursion can generate base cases that simultaneously work well with caches and the processor pipeline.

We believe we can improve the particular method we are using for divide-and-conquer recursion to provide more parallelism. Currently, the iteration space of the stencil is divided into ***hyperzoids*** — a multidimensional analog of a trapezoid. Our current divide-and-conquer strategy, which is based on the ideas of Frigo and Strumpen [7, 8], provide asymptotically cache-efficient performance. Our recent paper [18] showed that asymptotically more parallelism is available, and our new method is incorporated in Pochoir 1.0. We believe that the constant factors in this implementation can be reduced, which should lead to a more efficient implementation. The key ideas is to divide a hyperzoid by cutting into the smaller of the two faces normal to time, rather than the face normal to time having the smaller time index. This strategy should reduce the total number of so-called "time-cuts," which directly limits the parallelism.

For stencil computations with constant boundary conditions, a common implementation strategy (not used by Pochoir) is to store a ***halo*** of ***ghost cells*** around the perimeter of the stencil domain. The ghost cells can be set to a fixed boundary value and are never updated, but neighboring cells can obtain the boundary value without using an unpredictable branch to special-case the boundary. Pochoir does not use this strategy, because it special-cases the boundary outside the main computation using a two-clone strategy. For large stencil computations, halos would not improve Pochoir's strategy, because the boundary is asymptotically smaller than the volume. For small to medium-sized stencil computations, however, the halo strategy can yield significant performance gains. We believe that we can implement a strategy that will allow Pochoir to employ halos automatically for constant boundary conditions.

The code that Pochoir 1.0 generates contains tuning parameters that Pochoir sets empirically to achieve good performance. We have interfaced Pochoir to the Intel Software Autotuning Tool (ISAT), which can adjust these parameters using an exhaustive search of the tuning space. Recently, heuristic autotuners [9] have been developed which provide good tuning of parameters using heuristic search instead of exhaustive search. We plan to investigate how heuristic autotuners can be incorporated into Pochoir.

Widening the applicability of Pochoir

Pochoir 1.0 was designed to target multicore computers. We propose to explore what other architecture platforms and architectural features might be also be targeted, including distributed-memory clusters, graphical-processing units (GPU's), and future exascale computers.

In particular, we propose to extend Pochoir's cache-oblivious algorithm from a single multicore node to a cluster of multicore processors. The idea is to employ MPI for internode communication, and Intel Cilk Plus for intranode parallelism. We believe that we can accomplish this goal using a two-level design, where the low level is implemented by the existing multicore Pochoir and the high level is implemented by a new cluster Pochoir that employs MPI and targets the low level. For example, the high-level cluster Pochoir could generate Pochoir 1.0 code that employs MPI in its kernel and boundary functions.

GPU's and vector hardware present a different opportunity. Here, the idea would be for Pochoir to produce base cases of the recursion that exploits these architectural features. A challenging aspect of this idea is that vectorized loops consume more memory bandwidth than divide-and-conquer recursion. Thus, although the code may go faster using vector technologies on one processing core, it may go slower on multiple cores. An intriguing possibility is to adaptively select whether vectorization is used depending on how many cores the code executes on. We plan to experiment with adaptive ideas.

At an exascale level, the issues of checkpointing and fault tolerance arise. We propose to investigate how application-level automatic checkpointing and fault-tolerance mechanisms can be incorporated into Pochoir. The key issue that Pochoir will face is that its cache-oblivious algorithm moves though time nonuniformly, unlike the standard looping algorithm. Thus, resuming a computation after a checkpoint may require additional information that would not be required for the less computationally efficient looping implementation. We plan to explore how checkpointing and fault tolerance can be incorporated into Pochoir.

Embedding Pochoir in the Intel compiler

With direct compiler support, we believe that the specification language for Pochoir can be made even simpler and more intuitive. The idea would be to define a special "Pochoir" class in C++. To implement a stencil computation, the programmer would define a class that inherits from this Pochoir class and provide member functions to define kernel functions, boundary functions, etc.

Most of this work would need to be done by the Intel compiler team. We would plan to assist with the language design, algorithms, and runtime system needed to implement the system.

1 Statement of Work

The research project will be run by Professor Charles E. Leiserson at MIT and involve Professor Yuan Tang of Fudan University in Shanghai, China, who has been the principal developer of Pochoir while a Visiting Scientist at MIT. Professor Tang will reside in China during most of the

research project. Professors Leiserson and Tang will meet regularly using phone and video conferencing, and Professor Tang will visit MIT once during the project. The project will involve Dr. Chi-Keung Luk of Intel Corporation, Dr. Bradley C. Kuszmaul of MIT, Professor Steven G. Johnson of MIT, and Professor Rezaul Chowdhury of Stony Brook University. In addition, an M.Eng. student and undergraduate student will work on the project.

We are asking at this time for only one year of funding. We intend to work on all six tasks and produce a major release of prototype software by the end of the year. In addition, we shall seek NSF funding to continue the project after the year's end.

The focus of the software release will be Tasks 1–4. As we have done in the past with our research projects, including Cilk, FFTW, and Pochoir 1.0, we shall make our software freely available. We plan to work with computational scientists to make the Pochoir software responsive to their needs. We will also produce more experimental releases that explore the ideas outlined in Task 5. In particular, we envisage Task 5 as appropriate for student research. We shall continue to work with our collaborators at Intel, especially on Task 6 and on autotuning.

References

[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-Based Multicore 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. Stützle. 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 für Informatic, Friedrich-Alexander-Universität Erlangen-Nürnberg, 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.