The Pochoir Stencil Compiler Overview

Charles Leiserson, ***Yuan Tang***, Rezaul Chowdhury, Bradley Kuszmaul, CK Luk, Steven Johnson, Ekanathan Natarajan

Mar. 13, 2012

What's a stencil?

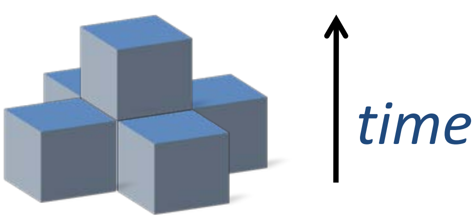
A stencil updates every point in a d-dimensional spatial grid at time t as a function of nearby grid points at times t–1, t–2, …, t–k, for T time steps.

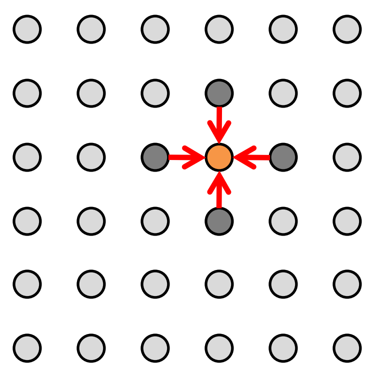
2D Heat equation

$$\frac{\partial a}{\partial t} = \alpha\left(\frac{\partial^2 a}{\partial x^2} + \frac{\partial^2 a}{\partial y^2}\right)$$

```
a(t,x,y) = a(t–1,x,y)
           + CX·(a(t–1,x+1,y) - 2·a(t–1,x,y) + a(t–1,x–1,y))
           + CY·(a(t–1,x,y+1) - 2·a(t–1,x,y) + a(t–1,x,y–1))
```

2D 5-point stencil





Story

- Stencils are prevailing
  - iterative PDE solvers such as Jacobi, multigrid, and AMR,
  - image processing,
  - geometric modeling.
- Highly cache-efficient stencil algorithm is known yet hard to write from case to case.
- Conventional numerical library focus on optimizing individual computation operator
- How to automate the optimization of a family of computation (such as stencil) in one framework is an open question.
  - Library?
  - DSL?
  - EDSL?
  - Compiler's pragma?
  - Autotuner?

Roadmap

- Release 0.5 (Feb. 2011)
- Release 1.0 (Mar. 2012)
- Release 2.0 (TBD)

Release 0.5

- Released in Feb. 2011
- Published in SPAA’11 & HotPar’11
- Simple, concise, declarative, and easily verifiable DSL embedded in C++, with Intel Cilk Plus extension.
- Arbitrary shaped, arbitrary depth stencil on arbitrary d-dimensional space-time grid, with complex boundary condition.

Current List of Known Users

- Oscar Barenys, Univ. Politechnica of Catalonia, Spain.
- Volker Strumpen, Johannes Kepler University, Austria.
- Nicolas Pinto, MIT/Harvard
- Nicolas Vasilache, Reservoir Lab.
- Patrick S. McCormick, Los Alamos National Lab.
- Mohammed Shaheen, Max Planck Institut Informatik, Germany
- Wim Vanroose, Universiteit Antwerpen, Belgium.
- Tom Henretty, Ohio State Univ.
- Protonu Basu, Univ. of Utah.
- Shoaib Kamil, Berkeley.
- Hal Finkel, Argonne National Lab.
- Matthias Christen, Klingelbergstrass, Basel, Switzerland.
- Vinayaka Bandishti, Indian Institute of Science, Bangalore, India.
- Hans Vandierendonck, Ghent University, Belguim.

Benchmark Suite

- Physics
  - Heat equation
  - Wave equation
  - Maxwell's equation
  - Lattice Boltzmann Method
- Computational Biology
  - RNA secondary structure prediction
  - Pairwise sequence alignment
- Computational Finance
  - American Put Stock Option Pricing
- Mechanical Engineering
  - Compressible Euler Flow
- Others
  - Conway's Game of Life
  - …

Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Algorithm

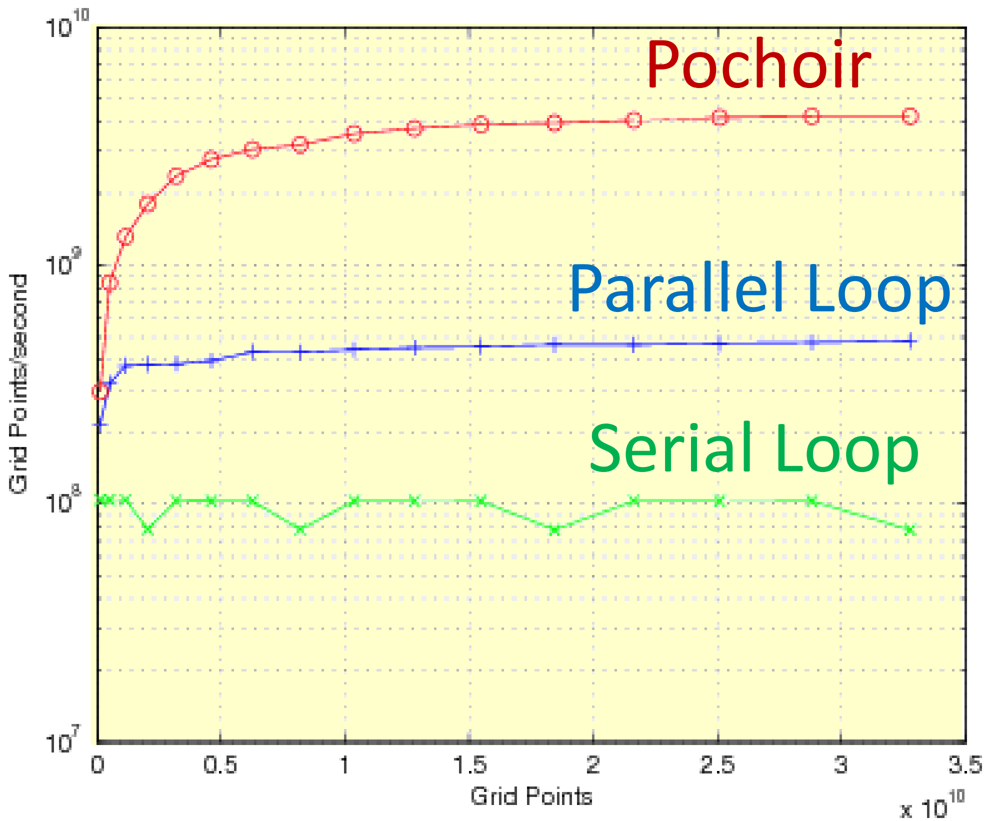
- Parallel cache-oblivious stencil algorithm
  - Based on Frigo and Strumpen's notion of trapezoidal decomposition
  - Improved parallelism by hyperspace cut strategy
  - Compared with looping implementation, reduce cache miss rate from $\Theta(NT/\mathcal{B})$ to $\Theta(NT/\mathcal{M}\mathcal{B})$
- C++ template meta-programming library to automatically expand for different stencils
  - Different kernels, boundaries, dimensionality, data types, etc.

Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

2D Heat Equation

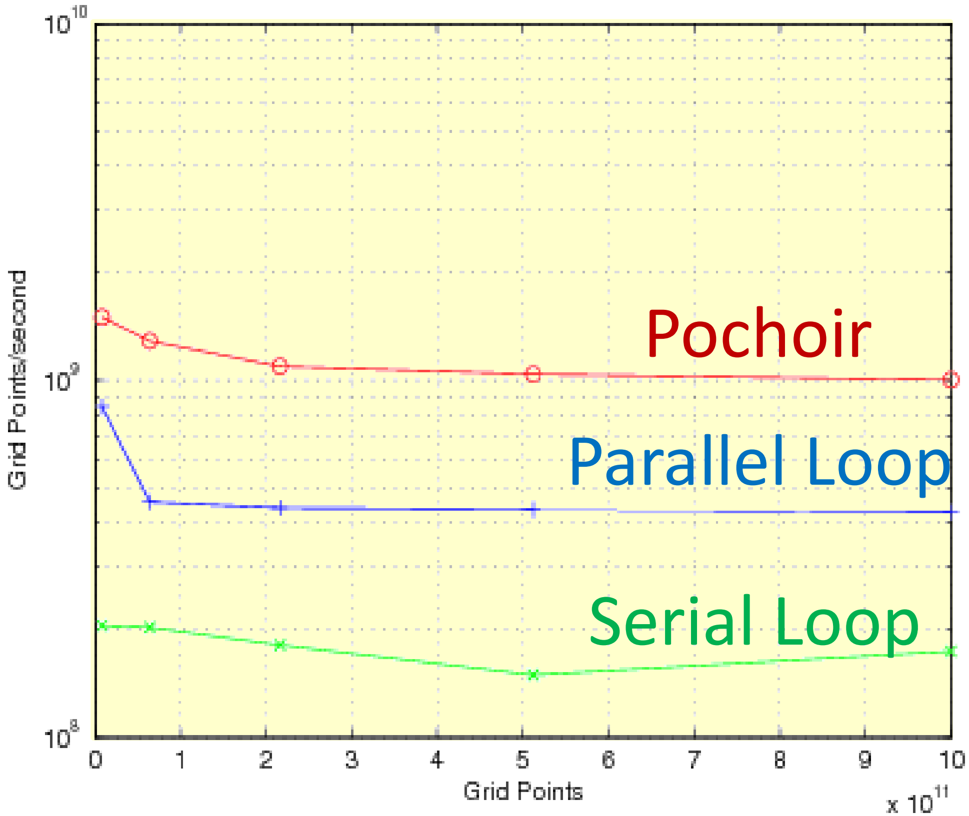
5-point stencil on a torus



Intel C++ compiler 12.0.0 with Cilk Plus on 12 core Intel core i7 (Nehalem)

3D Wave Equation

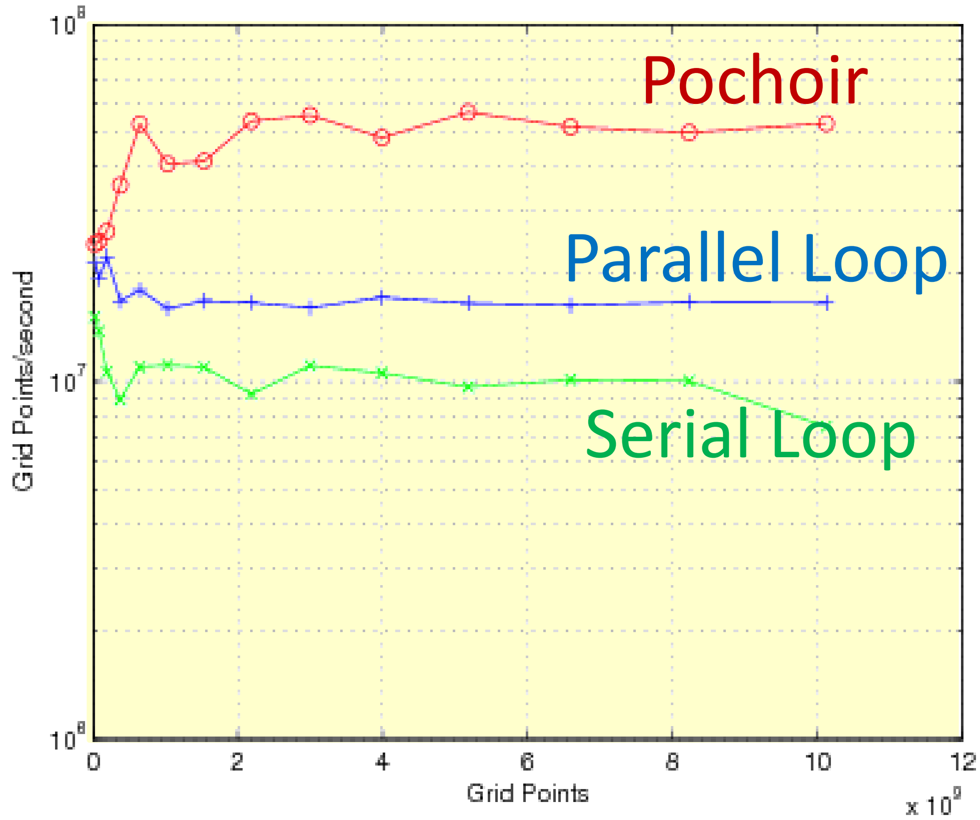
25-point stencil on a nonperiodic domain



Intel C++ compiler 12.0.0 with Cilk Plus on 12 core Intel core i7 (Nehalem)

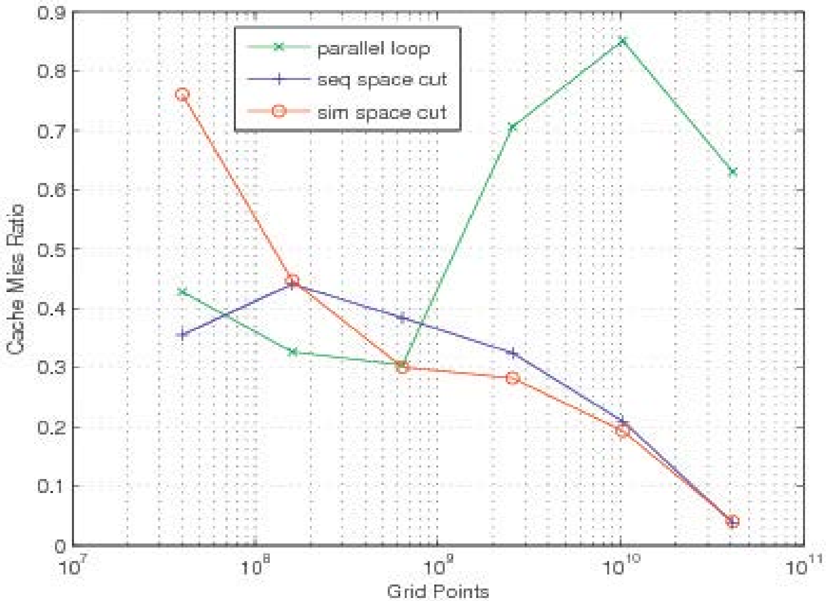
3D Lattice Boltzmann Method

19-point stencil on a nonperiodic domain

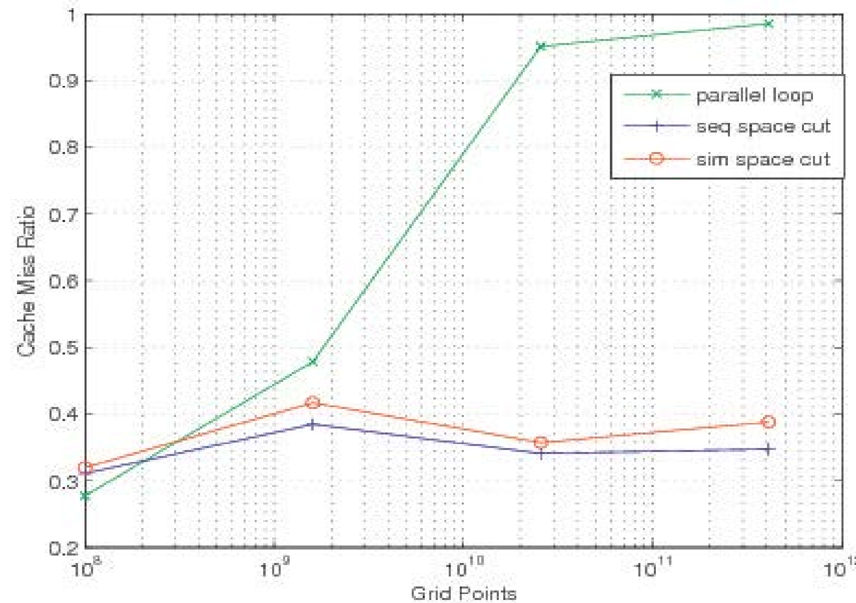


Intel C++ compiler 12.0.0 with Cilk Plus on 12 core Intel core i7 (Nehalem)

Cache miss ratio of Pochoir vs Parallel loops



Cache Miss Ratio of heat_2D_NP



Cache Miss Ratio of 3dfd

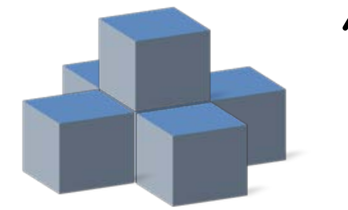
Pochoir vs Autotuner

	Berkeley Autotuner	***Pochoir***
CPU	Xeon X5550	Xeon X5650
Clock	2.66GHz	2.66 GHz
cores/socket, total	4, 8	6, 12
Hyperthreading	Enabled	Disabled
L1 data cache/core	32KB	32KB
L2 cache/core	256KB	256KB
L3 cache/socket	8MB	12 MB
Peak computation	85 GFLOPS	120 GFLOPS
Compiler	icc 10.0.0	icc 12.0.0
Linux kernel		2.6.32
Threading model	Pthreads	Intel Cilk Plus
Problem Size	$\mathbf{258^3 * 1}$	$\mathbf{258^3 * 200}$
3D 7-point 8 cores	2.0 GStencil/s 15.8 GFLOPS	***2.49 GStencil/s*** ***19.92 GFLOPS***
3D 27-point 8 cores	***0.95 GStencil/s*** ***28.5 GFLOPS***	0.88 GStencil/s 26.4 GFLOPS

Release 0.5

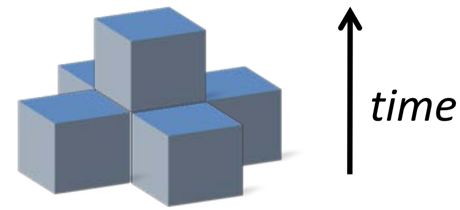
- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Specification of 2D Heat Equation




```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
             + 0.125*(a(t-1,x+1,y) - 2.0*a(t-1,x,y) + a(t-1,x-1,y))
             + 0.125*(a(t-1,x,y+1) - 2.0*a(t-1,x,y) + a(t-1,x,y-1));
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```

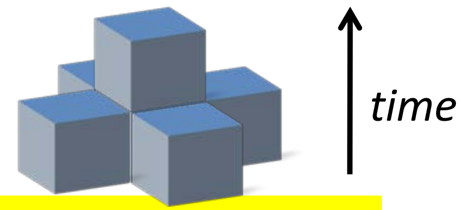
Specification of 2D Heat Equation




```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
             + 0.125*(a(t-1,x+1,y) - 2.0*a(t-1,x,y) + a(t-1,x-1,y))
             + 0.125*(a(t-1,x,y+1) - 2.0*a(t-1,x,y) + a(t-1,x,y-1));
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```

Declare a ***kernel function*** kern with time parameter t and spatial parameters x and y.

Specification of 2D Heat Equation

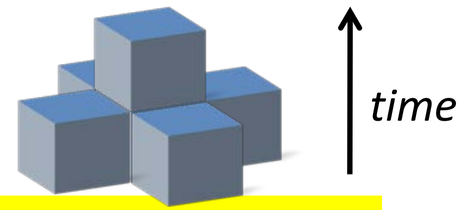



```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End

int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);

  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);

  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) -
              + 0.125*(a(t-1,x,y+1) -
  Pochoir_Kernel_End

  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();

  heat.Run(T, kern);

  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);

  return 0;
}
```

Declare the 2-dimensional ***Pochoir shape*** `2D_five_pt` as a list of 6 cells. Each cell specifies the relative offset of indices used in the kernel function, *e.g.*, for `a(t,x,y)`, we specify the corresponding cell `{0,0,0}`, for `a(t-1,x+1,y)`, we specify `{-1,1,0}`, and so on.

Specification of 2D Heat Equation

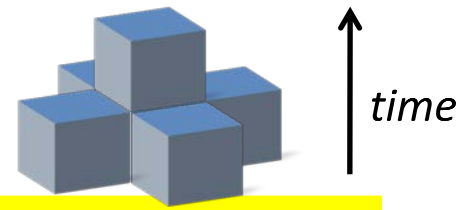
```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) -
              + 0.125*(a(t-1,x,y+1) -
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```



Declare the 2-dimensional ***Pochoir shape*** `2D_five_pt` as a list of 6 cells. Each cell specifies the relative offset of indices used in the kernel function, *e.g.*, for `a(t,x,y)`, we specify the corresponding cell `{0,0,0}`, for `a(t-1,x+1,y)`, we specify `{-1,1,0}`, and so on.

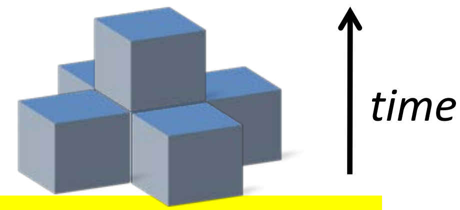
Specification of 2D Heat Equation

```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) = a(t-1,x,y)
             + 0.125*(a(t-1,x+1,y) -
             + 0.125*(a(t-1,x,y+1) -
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```



Declare the 2-dimensional ***Pochoir shape*** `2D_five_pt` as a list of 6 cells. Each cell specifies the relative offset of indices used in the kernel function, *e.g.*, for `a(t,x,y)`, we specify the corresponding cell `{0,0,0}`, for `a(t-1,x+1,y)`, we specify `{-1,1,0}`, and so on.

Specification of 2D Heat Equation

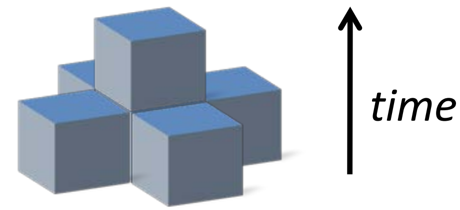



```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) -
              + 0.125*(a(t-1,x,y+1) -
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```

Declare the 2-dimensional ***Pochoir shape*** `2D_five_pt` as a list of 6 cells. Each cell specifies the relative offset of indices used in the kernel function, *e.g.*, for `a(t,x,y)`, we specify the corresponding cell `{0,0,0}`, for `a(t−1,x+1,y)`, we specify `{−1,1,0}`, and so on.

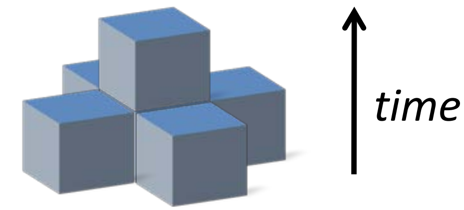
Specification of 2D Heat Equation

```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,0,0}, {-1,1,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) - 2
              + 0.125*(a(t-1,x,y+1) - 2.0*a(t-1,x,y) + a(t-1,x,y-1));
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```



Declare a ***boundary function*** `zero_bdry` on the 2-dimensional Pochoir array `arr` indexed by time coordinate `t` and spatial coordinates `x` and `y`, which always returns `0`.

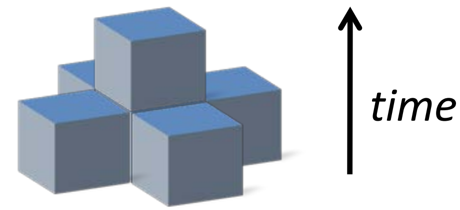
Specification of 2D Heat Equation




```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,1,0}, {-1,0,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) - 2.0*a(t-1,x,y) + a(t-1,x-1,y))
              + 0.125*(a(t-1,x,y+1) - 2.0*a(t-1,x,y) + a(t-1,x,y-1));
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```

Initialize all points of the grid at time `0` to a random value.

Specification of 2D Heat Equation




```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
int main(void) {
  Pochoir_Shape_2D 2D_five_pt[6]
    = {{0,0,0}, {-1,1,0}, {-1,0,0}, {-1,-1,0}, {-1,0,-1}, {-1,0,1}};
  Pochoir_2D heat(2D_five_pt);
  Pochoir_Array_2D(double) a(X,Y);
  a.Register_Boundary(zero_bdry);
  heat.Register_Array(a);
  Pochoir_Kernel_2D(kern, t, x, y)
    a(t,x,y) =  a(t-1,x,y)
              + 0.125*(a(t-1,x+1,y) - 2.0*a(t-1,x,y) + a(t-1,x-1,y))
              + 0.125*(a(t-1,x,y+1) - 2.0*a(t-1,x,y) + a(t-1,x,y-1));
  Pochoir_Kernel_End
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      a(0,x,y) = rand();
  heat.Run(T, kern);
  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << a(T,x,y);
  return 0;
}
```

Run a stencil computation on the Pochoir object heat for T time steps using kernel function kern. The Run method can be called multiple times.

Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Various Boundary Conditions

Nonperiodic zero boundary

```
Pochoir_Boundary_2D(zero_bdry, arr, t, x, y)
  return 0;
Pochoir_Boundary_End
```

Periodic (toroidal) boundary

```
#define mod(r,m) (((r) % (m)) + ((r)<0)?(m):0)
Pochoir_Boundary_2D(periodic, arr, t, x, y)
  return arr.get( t,
                  mod(x, arr.size(1)),
                  mod(y, arr.size(0)) );
Pochoir_Boundary_End
```

Cylindrical boundary

```
#define mod(r,m) (((r) % (m)) + ((r)<0)?(m):0)
Pochoir_Boundary_2D(cylinder, arr, t, x, y)
  if (x < 0) || (x >= arr.size(1))
    return 0;
  return arr.get( t, x, mod(y, arr.size(0)) );
Pochoir_Boundary_End
```

Various Boundary Conditions

Dirichlet boundary

```
Pochoir_Boundary_2D(dirichlet, arr, t, x, y)
  return 100+0.2*t;
Pochoir_Boundary_End
```

Neumann boundary

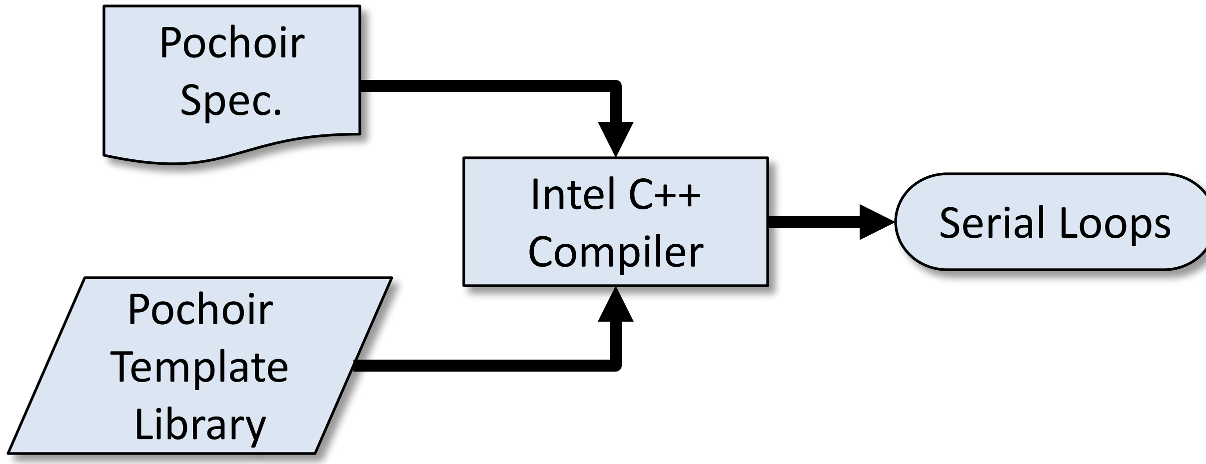
```
Pochoir_Boundary_2D(neumann, arr, t, x, y)
int xx(x), yy(y);
  if (x<0) xx = 0;
  if (x>=arr.size(1)) xx = arr.size(1);
  if (y<0) yy = 0;
  if (y>=arr.size(0)) yy = arr.size(0);
  return arr.get(t, xx, yy);
Pochoir_Boundary_End
```

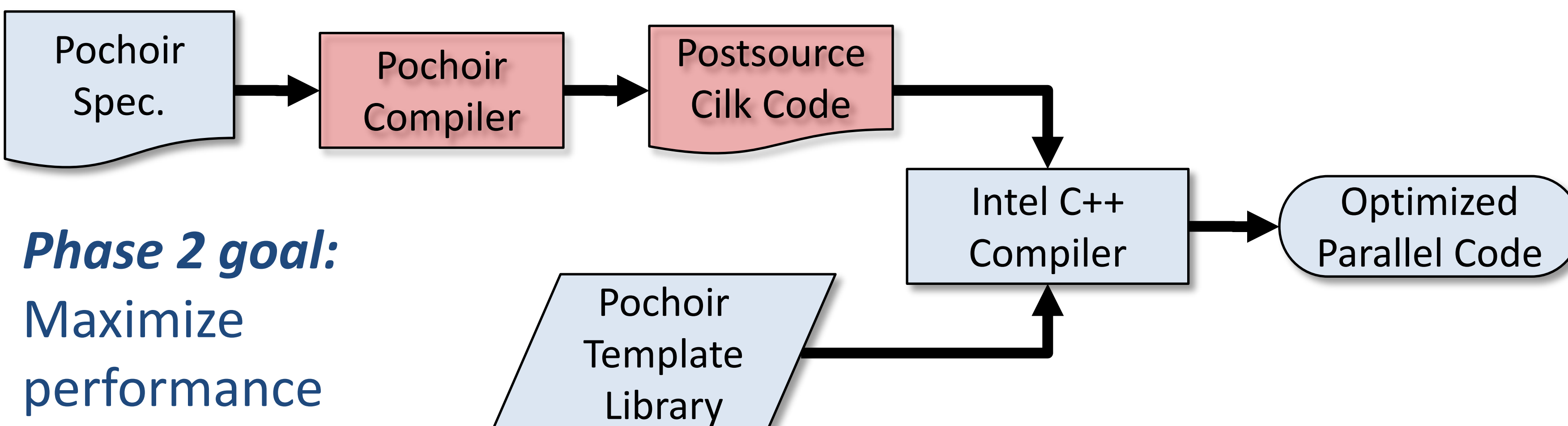
Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Two-Phase Compilation Strategy

Phase 1 goal:
Check functional correctness

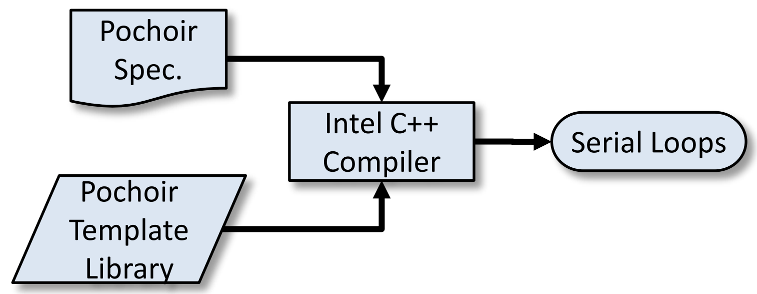




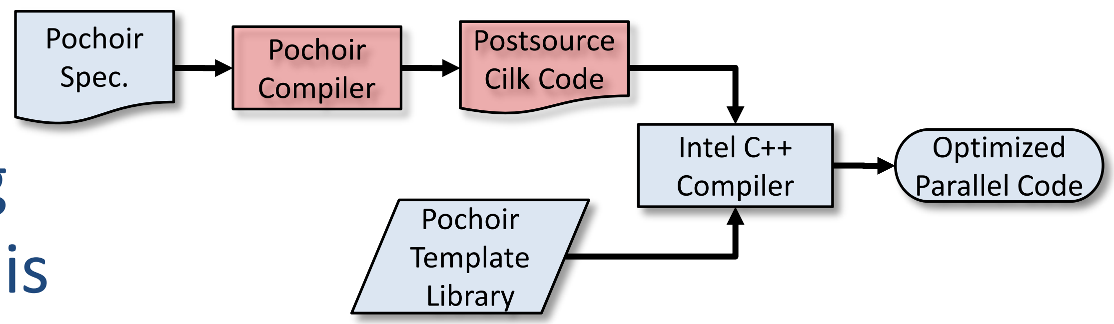
Phase 2 goal:
Maximize performance

Pochoir Guarantee

If a stencil program compiles and runs with the Pochoir template library during Phase 1,



then no errors will occur during Phase 2 when it is compiled with the Pochoir compiler or during the subsequent running of the optimized binary.



Release 0.5

- Algorithm:
- Performance Results:
- Specification:
- Boundary Conditions:
- Compilation strategy:
- Optimization strategies:

Optimization Strategies

- Two code clones
- Unifying the handling of periodic and nonperiodic boundary conditions
- Automatic selection of traversal strategy
  - -split-macro-shadow
  - -split-opt-pointer
- Coarsening of base cases
- Adaptive trapezoidal decomposition

Release 1.0

- Mar. 2012
- Bug Fix
- User's feedback
- Variadic Template Support
  - Even Simpler user interface

Release 2.0

- TBD
- Inhomogeneity
  - Macroscopic Inhomogeneity
  - Microscopic Inhomogeneity
- JIT compilation
  - Generate the computational kernels on-the-fly
- Generalized Dependency
  - From orthogonal grid to general graph

Contributions

- Simple, concise, declarative, and easily verifiable DSL embedded in C++, with Intel Cilk Plus extension.
- Arbitrary shaped, arbitrary depth stencil on arbitrary d-dimensional space-time grid, with complex boundary condition.
- Inhomogeneous regions
  - Macroscopic inhomogeneity
  - Microscopic inhomogeneity
- JIT compiler for stencil
- Generalized dependency
  - From orthogonal grid to general graph

Funded in medium scale by NSF

Pochoir Team

- Charles E. Leiserson, Project Leader
- Yuan Tang
- Rezaul Alam Chowdhury
- Bradley C. Kuszmaul
- Chi-Keung Luk
- Steven G. Johnson
- Ekanathan Palamadai Natarajan