Practical Diamond Tiling for Stencil Computations Using Chapel Iterators

Michelle Mills Strout – Colorado State University → University of Arizona
Ian Bertolacci – Colorado State University
Catherine Olschanowsky – Colorado State University
Ben Harshbarger – Cray Inc.
Bradford L. Chamerlain – Cray Inc.
David G. Wonnacott – Haverford College

June 12, 2015
CHIUW

This project is supported by
Department of Energy Early Career Grant DE-SC0003956 and
National Science Foundation Grant CCF-1422725

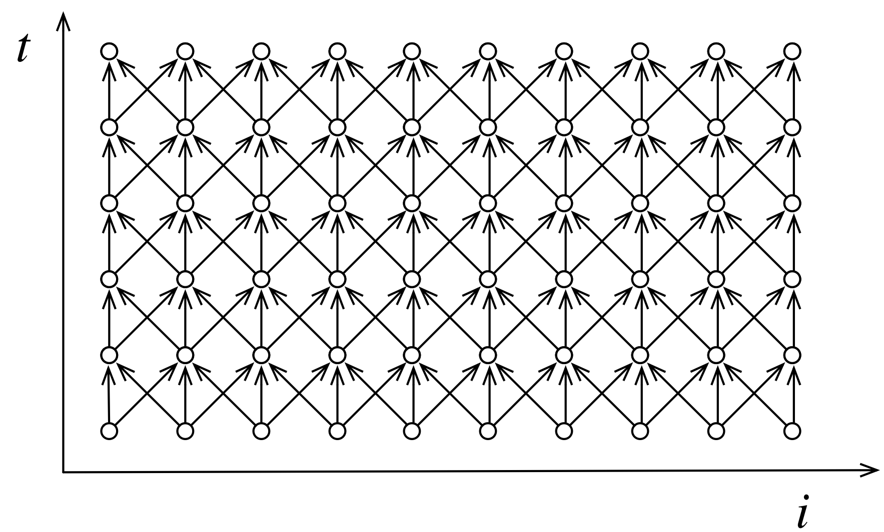
Practical Tiling

* Tiling strategies provide parallelism and data locality in stencil computations
* Problem: advanced tiling strategies are …
  * not generally implemented in compilers,
  * difficult to implement by hand, and
  * result in code that is difficult to maintain.
* Punchline: Abstract tile size and loop bounds
  * Parameterization of diamond tiling.
  * Use of Chapel iterators to abstract tile schedule.

Target: Stencil Computations

* Image processing
* Cellular automata
* Partial Differential Equation solvers
  * Air and fluid flow simulation
  * Seismic wave models and damage simulation
  * Blast-wave equations
  * Heat equations
  * Atmospheric modeling
  * Magnetic field simulations

Jacobi 1D Example



Stencil Computations

Jacobi 2D Example

C + OpenMP:

Chapel:

Stencil Computations

C + OpenMP:

Jacobi 2D Example

```
A[t][x][y]=( A[t-1][x-1][y] + A[t-1][x][y-1]
           + A[t-1][x+1][y] + A[t-1][x][y+1]
           + A[t-1][x  ][y] )/5;
```

Chapel:

```
A[t,x,y]=( A[t-1, x-1, y] + A[t-1, x, y-1]
         + A[t-1, x+1, y] + A[t-1, x, y+1]
         + A[t-1, x  , y] )/5;
```

Stencil Computations

C + OpenMP:

Jacobi 2D Example

```
for( int x = 1; x <= N; x += 1 )
  for( int y = 1; y <= M; y += 1 )
    A[t][x][y]=( A[t-1][x-1][y] + A[t-1][x][y-1]
               + A[t-1][x+1][y] + A[t-1][x][y+1]
               + A[t-1][x  ][y] )/5;
```

Chapel:

```
for (x,y) in {1..N, 1..M} do
  A[t,x,y]=( A[t-1, x-1, y] + A[t-1, x, y-1]
           + A[t-1, x+1, y] + A[t-1, x, y+1]
           + A[t-1, x  , y] )/5;
```

Stencil Computations

C + OpenMP: **Jacobi 2D Example**

```
for( int t = 1; t <= T; t += 1 )
  for( int x = 1; x <= N; x += 1 )
    for( int y = 1; y <= M; y += 1 )
      A[t][x][y]=( A[t-1][x-1][y] + A[t-1][x][y-1]
                 + A[t-1][x+1][y] + A[t-1][x][y+1]
                 + A[t-1][x  ][y] )/5;
```

Chapel:

```
for t in 1..T do
  for (x,y) in {1..N, 1..M} do
    A[t,x,y]=( A[t-1, x-1, y] + A[t-1, x, y-1]
             + A[t-1, x+1, y] + A[t-1, x, y+1]
             + A[t-1, x  , y] )/5;
```

Stencil Computations

C + OpenMP:

Jacobi 2D Example

```
for( int t = 1; t <= T; t += 1 )
  #pragma omp parallel for collapse(2)
  for( int x = 1; x <= N; x += 1 )
    for( int y = 1; y <= M; y += 1 )
      A[t][x][y]=( A[t-1][x-1][y] + A[t-1][x][y-1]
                 + A[t-1][x+1][y] + A[t-1][x][y+1]
                 + A[t-1][x][y] )/5;
```

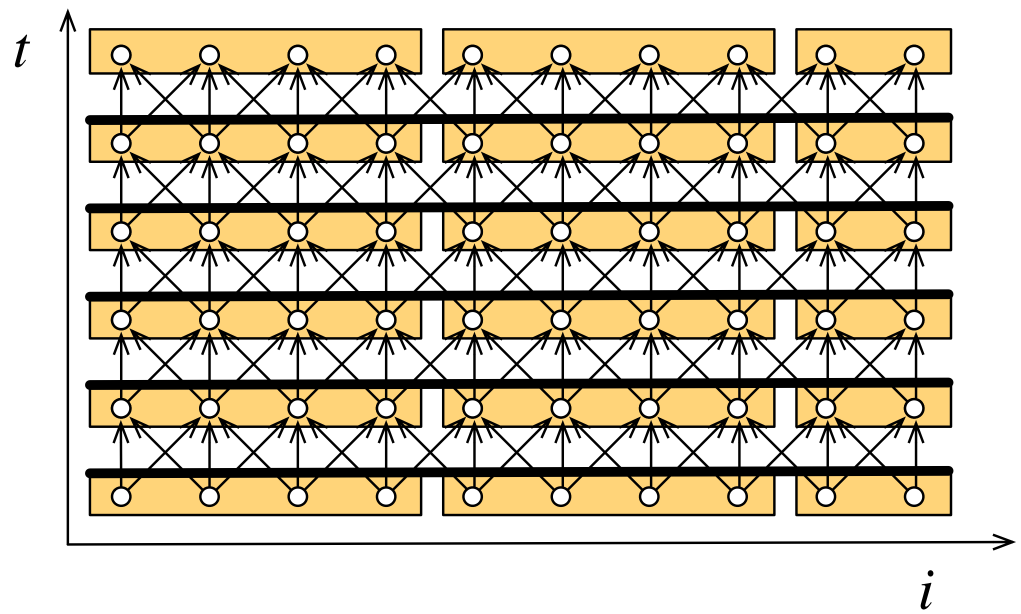
Chapel:

```
for t in 1..T do
  forall (x,y) in {1..N, 1..M} do
    A[t,x,y]=( A[t-1, x-1, y] + A[t-1, x, y-1]
             + A[t-1, x+1, y] + A[t-1, x, y+1]
             + A[t-1, x  , y] )/5;
```

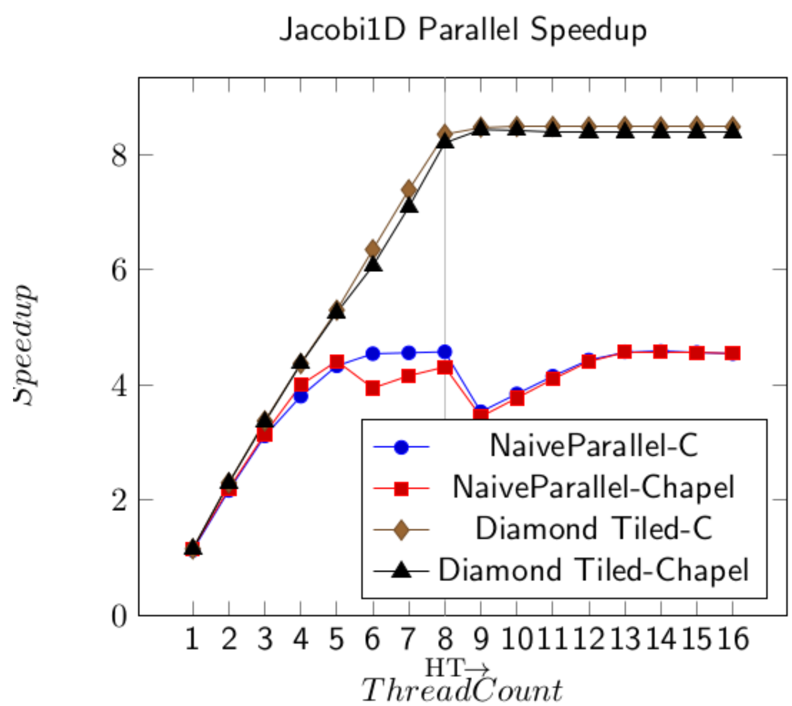
Stencil Computations

* Naïve parallelization performance does not scale with additional cores!
  * Bandwidth-bound computation
  * Naïve parallelism does not have enough data locality

Jacobi 1D Example



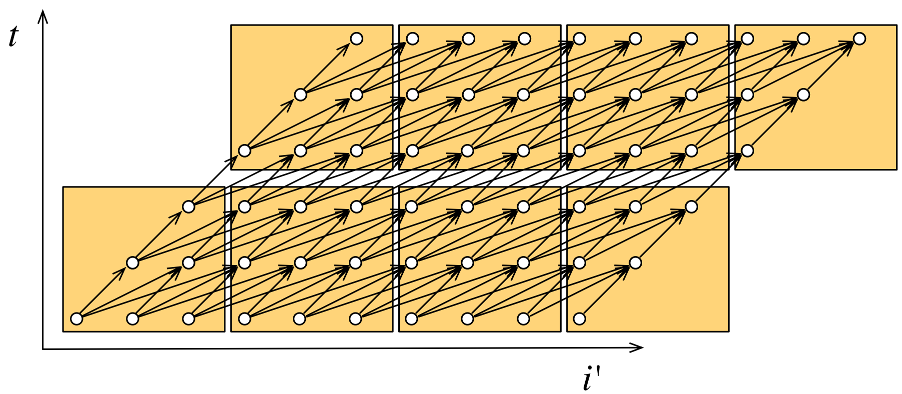
Tiling Mitigates the Scaling Problem by Reducing Memory Bandwidth Pressure

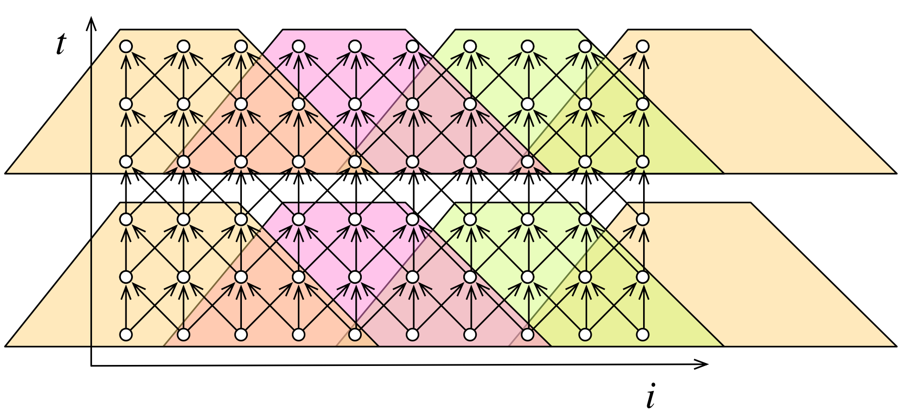


* Jacobi: 1D data, 1D time
* Naïve parallelization does not scale past 5 threads
* Diamond tiled scales to 8 threads

Many Advanced Tiling Strategies

Jacobi 1D Example



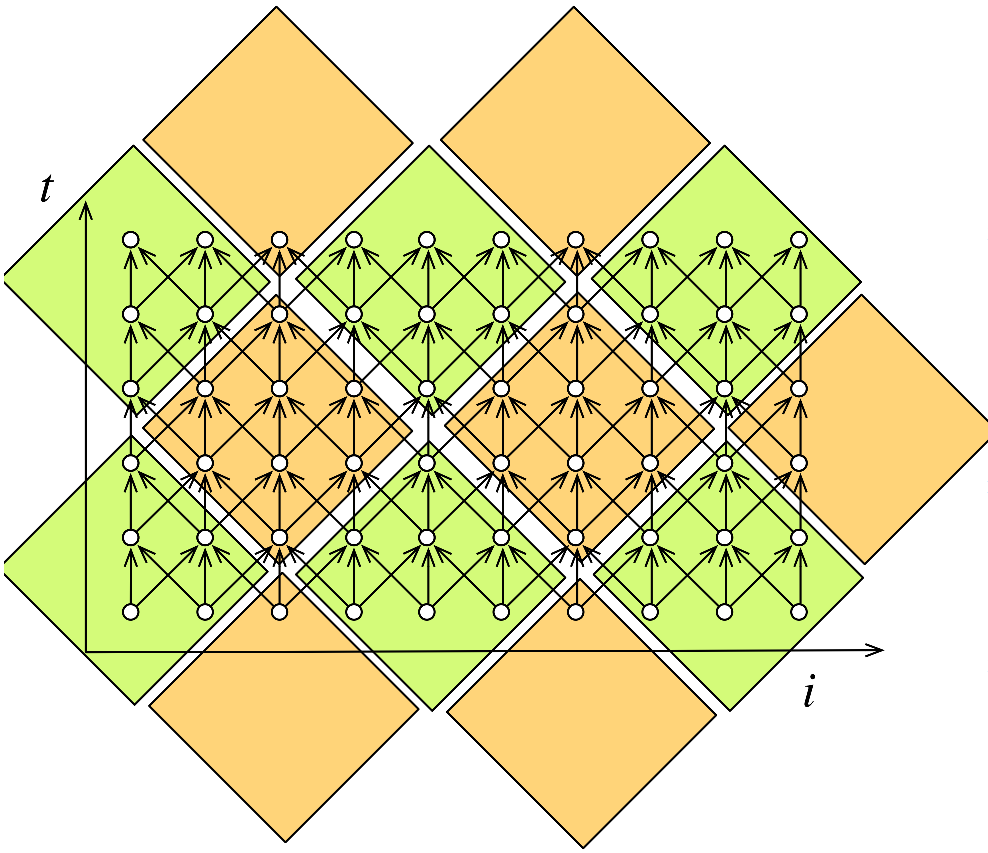


- Pipelined tiling (shown)
- Overlapped tiling (shown)
- Diamond tiling
- Split tiling
- Hexagonal tiling
- Hybrid hexagonal and diamond
- …

Modern Solution: Diamond Tiling

[Bandishti et al. 2012]

Jacobi 1D Example



Mixes space and time tiling

* A single tile will perform multiple time steps
  * Reuse just-computed values
* Concurrent Startup
  * Many tiles can start in parallel
* Load balanced wavefronts

Polyhedral Code Generators

Cloog+ISL, Omega+

Jacobi 1D Example

ISCC: ISL calculator

```
# Jacobi1D stencil computation
#   for (t=1; t<=T; t++) {
#     for (i=1; i<=N; i++) {
#  S:   A[t][i] = (A[t-1][i-1]+A[t-1][i]
#                 + A[t-1][i]);
#     }
#   }
I := [T,N] -> { S[t,i] : 1<=t<=T and 1<=i<=N };

# Diamond Tiling
T_diamond:=[T,N]->{ S[t,i] -> [kt, k1, t, i] :
    exists k0,r0,r1: 0<=r0<12 && 0<=r1<12
    && 12*k0+r0 = t+i && 12*k1+r1 = t-i
    && kt = k0+k1
};

codegen (T_diamond*I) assuming [T,N]->{:T>1&&N>1};
```

Output Code

```
for (int c0 = -1; c0 <= T / 6; c0 += 1)
  for (int c1 = max(max(floord(-N + 6 * c0 + 6, 12),
c0 - (T + N + 12) / 12 + 1), -((N + 10) / 12)); c1 <=
min(floord(c0, 2), (T - 1) / 12); c1 += 1)
    for (int c2 = max(max(max(12 * c1 + 1, -N + 12 *
c0 - 12 * c1), 6 * c0), 1); c2 <= min(min(min(N + 12
* c1 + 11, 6 * c0 + 11), 12 * c0 - 12 * c1 + 10), T);
c2 += 1)
      for (int c3 = max(max(-12 * c1 + c2 - 11, 12 *
c0 - 12 * c1 - c2), 1); c3 <= min(min(N, -12 * c1 +
c2), 12 * c0 - 12 * c1 - c2 + 11); c3 += 1)
        S(c2, c3);
```

Opportunities for Improvement

* Steep learning curve to use code generation tools.
* Tile size needs to be constant.
* Compilers have a difficult time determining when tiling is applicable.
* Often must modify generated code to insert parallel for pragmas.
* Resulting code is difficult to maintain.

Code Generated is Difficult to Read and Maintain

Jacobi 2D Example

```
for (int c0 = -2; c0 <= floord(T - 1, 4); c0 += 1)
  for (int c1 = max(max(floord(Lj + 4 * c0 + 4, 12), c0 + floord(-2 * T + Lj + 1, 12) + 1),
floord(Lj + 1, 12)); c1 <= min(min(floord(Uj + 4 * c0 + 6, 12), floord(T + Uj - 2, 12)), c0 +
floord(Uj - 5, 12) + 2); c1 += 1)
    for (int c2 = max(max(max(max(max(max(c0 - 2 * c1 + floord(-Ui + Lj + 1, 12), -c1 +
floord(-2 * Ui - Uj + 12 * c0 + 12 * c1 - 9, 24) + 1), c1 + floord(-Ui - 2 * Uj + 3, 12)),
floord(-Ui - Uj + 3, 12)), c0 - c1 + floord(-Ui - 4 * c0 + 5, 12)), c0 - c1 + floord(-T - Ui +
1, 12) + 1), -c1 + floord(-Ui + 4, 12) - 1); c2 <= min(min(min(min(min(min(c0 - 2 * c1 +
floord(-Li + Uj - 2, 12) + 1, c0 - c1 + floord(-Li - 2, 12) + 1), c0 - c1 + floord(-Li - 4 *
c0 - 5, 12) + 1), -c1 + floord(2 * T - Li - 2, 12)), floord(T - Li - Lj - 1, 12)), c1 +
floord(-Li - 2 * Lj - 1, 12) + 1), -c1 + floord(-2 * Li - Lj + 12 * c0 + 12 * c1 + 11, 24));
c2 += 1)
      for (int c3 = max(max(max(max(max(max(4 * c0, 6 * c0 - 6 * c1 + floord(Lj + 1, 2)), 6 *
c1 + 6 * c2 + floord(Li + 1, 2)), Li + Lj + 12 * c2), -Ui + 12 * c0 - 12 * c1 - 12 * c2 + 1),
-Uj + 12 * c1 + 1), 1); c3 <= min(min(min(min(min(min(-Lj + 12 * c1 + 11, -Li + 12 * c0 - 12 *
c1 - 12 * c2 + 11), Ui + Uj + 12 * c2 + 9), T - 1), 6 * c1 + 6 * c2 + floord(Ui + 1, 2) + 10),
6 * c0 - 6 * c1 + floord(Uj + 1, 2) + 10), 4 * c0 + 11); c3 += 1)
        for (int c4 = max(max(max(-12 * c1 - 12 * c2 + 2 * c3 - 22, -Uj - 12 * c2 + c3 - 10),
12 * c0 - 12 * c1 - 12 * c2 - c3), Li); c4 <= min(min(min(12 * c0 - 12 * c1 - 12 * c2 - c3 +
11, -12 * c1 - 12 * c2 + 2 * c3), -Lj - 12 * c2 + c3), Ui - 1); c4 += 1)
          for (int c5 = max(max(12 * c1 - c3, Lj), -12 * c2 + c3 - c4 - 11); c5 <= min(min(Uj
- 1, -12 * c2 + c3 - c4), 12 * c1 - c3 + 11); c5 += 1)
            computation(c3, c4, c5);
```

Existing Solutions for Advanced Tiling

* Pluto compiler [Bondhugula et al. 2008]
  * Loops written in straight-forward C can be tiled and parallelized automatically
  * Pipelined tiling and diamond tiling incorporated
  * Parameterized tiling for rectangular tiles
* Domain Specific Languages
  * Halide [Ragan-Kelley et al. 2013]
  * Stencil DSL [Holewinski et al. 2012]
  * Pochoir [Tang et al. 2011]
  * PATUS [Christen et al. 2011]
  * Alphaz [Yuki et al. 2012]
  * …

Practical Solution for Incorporating Advanced Tiling (ICS 2015)

* Parameterized Diamond Tiling
* Demonstration of Chapel iterators as effective tiling schedule deployment mechanism

(ICS 2015) I. J. Bertolacci, C. Olschanowsky, B. Harshbarger, B. L. Chamberlain, D. G. Wonnacott, and M. M. Strout. Parameterized diamond tiling for stencil computations with chapel parallel iterators. In Proceed- ings of the 29th International Conference on Supercomputing (ICS), 2015.

Main Result:
Reduction of Code Complexity

Without Iterator

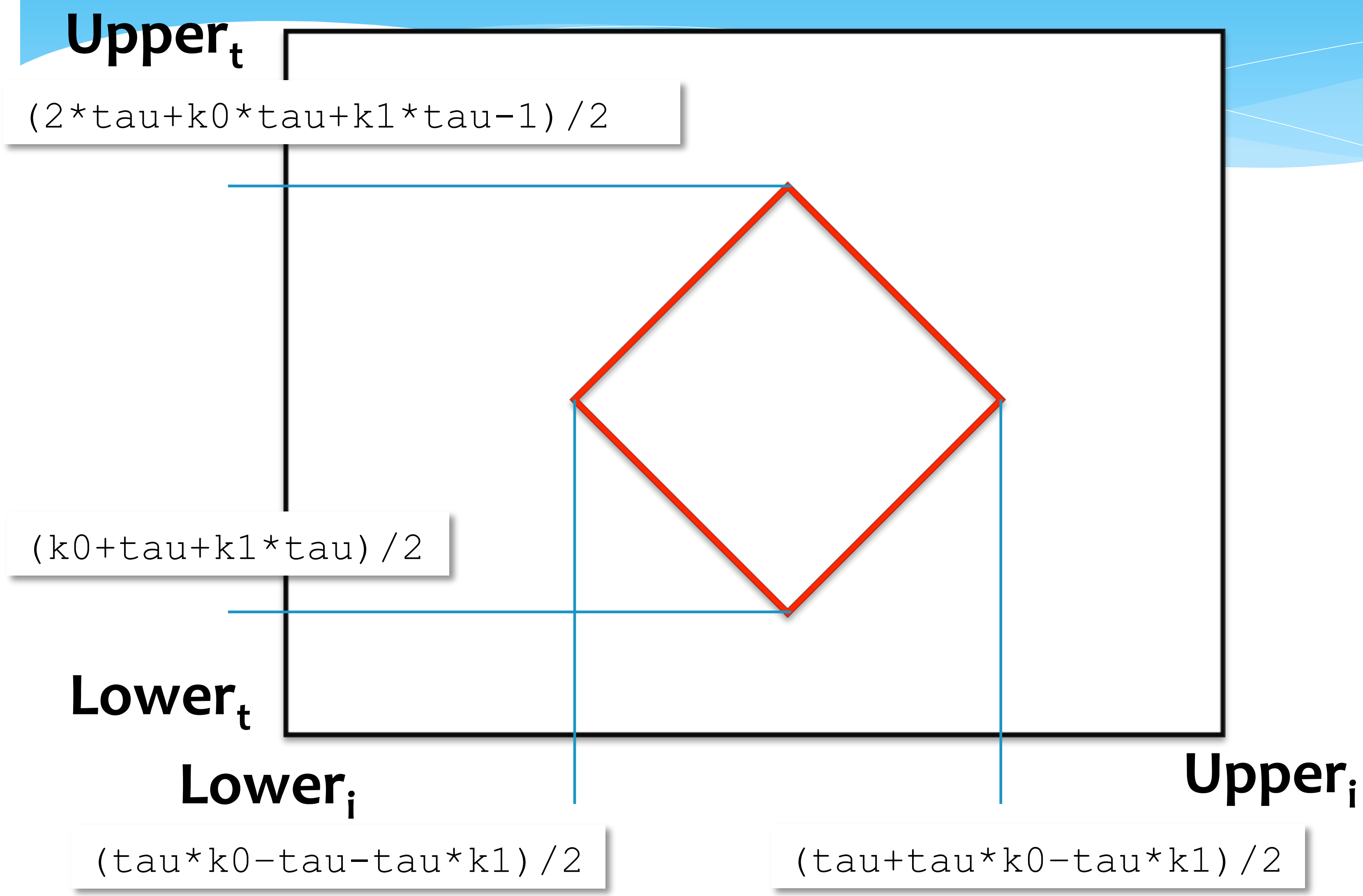
```
for kt in -2 .. floord(3*T,tau) {
 var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
 var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
 var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
 var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);
 forall k1 in k1_lb .. k1_ub {
  for x in k2_lb .. k2_ub {
   var k2 = x-k1;
   for t in max(1,floord(kt*tau,3))
         .. min(T,floord((3+kt)*tau-3,3)){
    write = t & 1;  read = 1 - write;
    for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
          .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-
                 (k1+k2)*tau)) {
     for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
           .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){
      A[write,x,y]=(A[read,x-1,y]+A[read, x, y-1]+ A[read,x+1,y]
                    + A[read,x,y+1]+ A[read,x,y] )/5;
} } } } } }
for kt in -2 .. floord(3*T,tau) {
 var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
 var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
 var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
 var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);
 forall k1 in k1_lb .. k1_ub {
  for x in k2_lb .. k2_ub {
   var k2 = x-k1;
   for t in max(1,floord(kt*tau,3))
         .. min(T,floord((3+kt)*tau-3,3)){
    write = t & 1;  read = 1 - write;
    for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
          .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-
                 (k1+k2)*tau)) {
     for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
           .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){
      B[write,x,y]= B[read,x-1,y]+ B[read, x, y-1]+B[read,x+1,y]
                    + B[read, x, y+1]+ 4*B[read,x,y];
} } } } } }
```

With Iterator

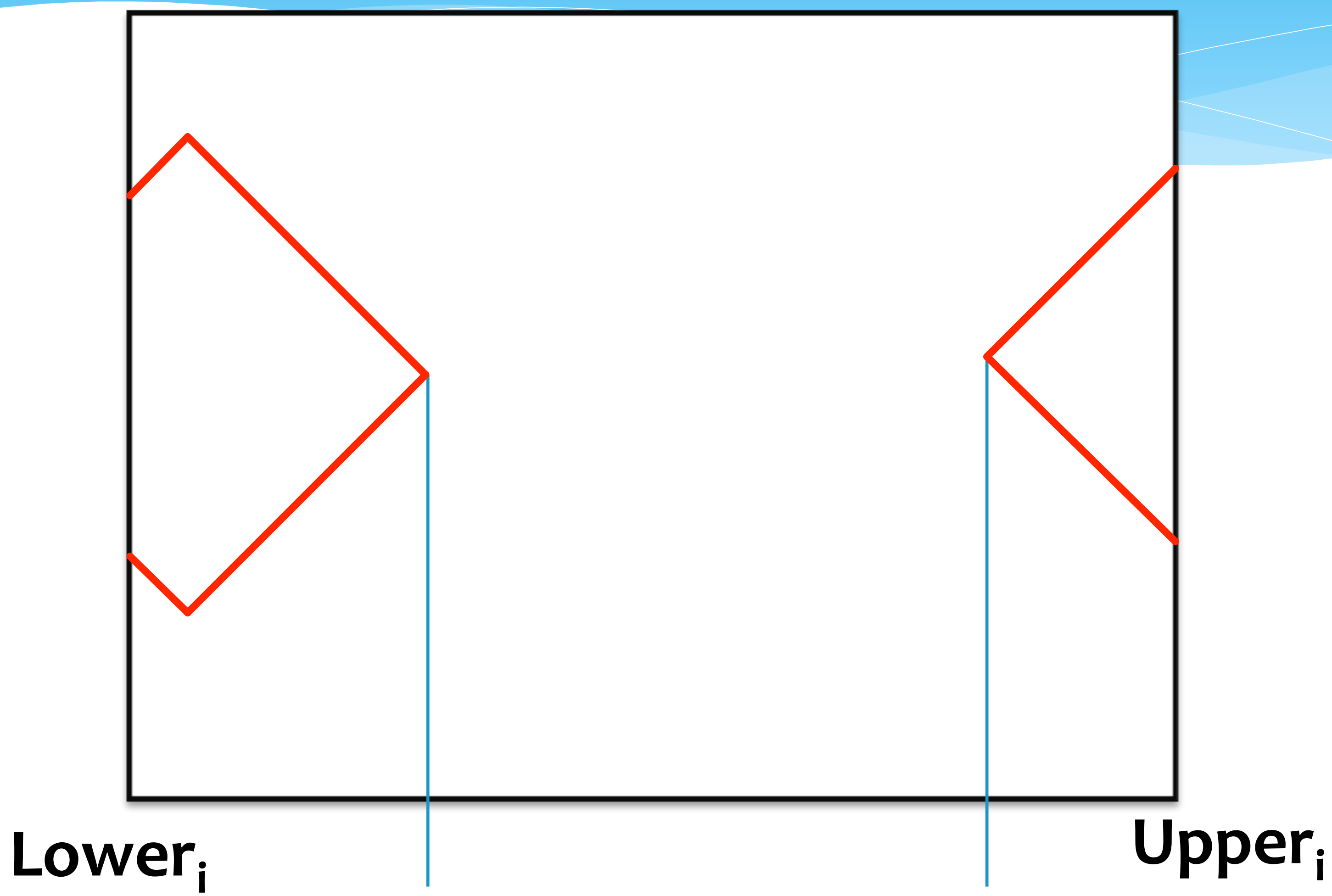
```
forall (read, write, x ,y) in
 DiamondTileIterator(L, U, T, tau) {
  A[write, x, y] = ( A[read, x-1, y] +
                     A[read, x, y-1] +
                     A[read, x, y  ] +
                     A[read, x, y+1] +
                     A[read, x+1, y] )/5;
}

forall (read, write, x ,y) in
 DiamondTileIterator(L, U, T, tau) {
  B[write, x, y] =   B[read, x-1, y] +
                     B[read, x, y-1] +
                     4* B[read, x, y  ] +
                     B[read, x, y+1] +
                     B[read, x+1, y];
}
```

Parameterized Diamond Tiling



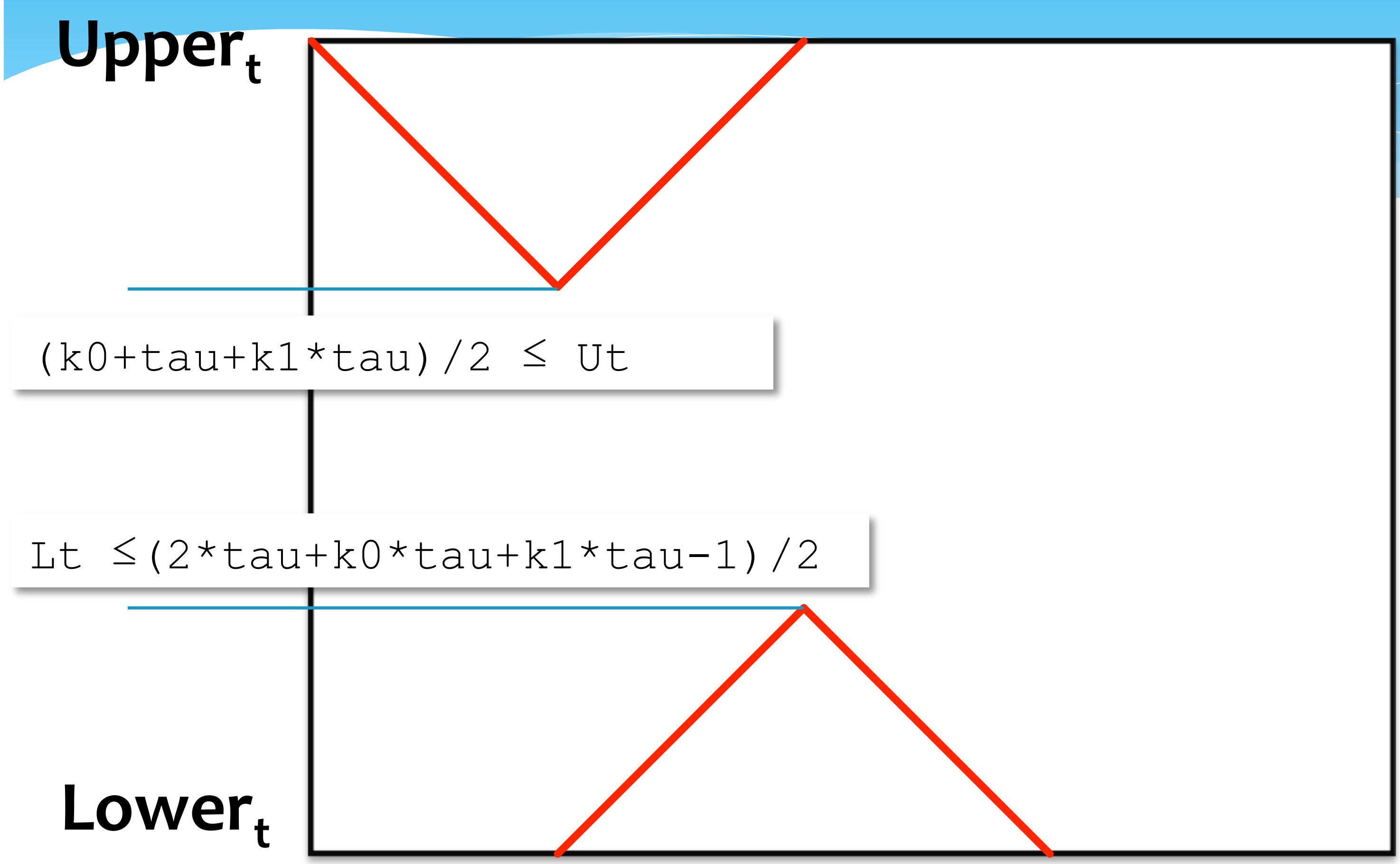
Parameterized Diamond Tiling




```
Li ≤ (tau*k0-tau-tau*k1)/2
```

```
(tau+tau*k0-tau*k1)/2 ≤ Ui
```

Parameterized Diamond Tiling



Parameterized Diamond Tiling

* Generalizable to higher dimensionality and different tiling hyperplanes.
* (ICS 2015) paper presents methodology to do this by hand.

Parameterized Diamond Tiling Code

C + OpenMP:

Jacobi 2D Example

```
for (kt=ceild(3,tau)-3; kt<=floord(3*T,tau); kt++) {
 int k1_lb = ceild(3*Lj+2+(kt-2)*tau,tau*3);
 int k1_ub = floord(3*Uj+(kt+2)*tau,tau*3);
 int k2_lb = floord((2*kt-2)*tau-3*Ui+2,tau*3);
 int k2_ub = floord((2+2*kt)*tau-3*Li-2,tau*3);
 #pragma omp parallel for
 for (k1 = k1_lb; k1 <= k1_ub; k1++) {
  for (x = k2_lb; x <= k2_ub; x++) {
   k2 = x - k1;
   for (t = max(1, floord(kt*tau-1, 3));
        t < min(T+1, tau + floord(kt*tau, 3));
        t++) {
    write = t & 1;
    read = 1 - write;
    for (x = max(Li,max((kt-k1-k2)*tau-t, 2*t-(2+k1+k2)*tau+2));
         x <= min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-(k1+k2)*tau));
        x++) {
     for (y = max(Lj,max(tau*k1-t, t-i-(1+k2)*tau+1));
          y <= min(Uj,min((1+k1)*tau-t-1, t-i-k2*tau));
          y++) {
      A[write][x][y] = ( A[read][x-1][y] + A[read][x][y-1]+
                         A[read][x+1][y] + A[read][x][y+1]+
                         A[read][x  ][y] )/5;
} } } } } }
```

Tile Wave-fronts

Tile Coordinates

Time-Steps in Tile

Read/Write buffer

X,Y Coordinates

Stencil Operation

Parameterized Diamond Tiling Code

Jacobi 2D Example

Chapel:

```
for kt in -2 .. floord(3*T,tau) {
 var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
 var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
 var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
 var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);

 forall k1 in k1_lb .. k1_ub {
  for x in k2_lb .. k2_ub {
   var k2 = x-k1;
   for t in max(1,floord(kt*tau,3))
         .. min(T,floord((3+kt)*tau-3,3)){
    write = t & 1;
    read = 1 - write;
    for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
          .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-(k1+k2)*tau)) {

     for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
           .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){
      A[write,x,y]=( A[read, x-1, y] + A[read, x, y-1]+
                     A[read, x+1, y] + A[read, x, y+1]+
                     A[read, x  , y] )/5;
} } } } } }
```

Tile Wave-fronts

Tile Coordinates

Time-Steps in Tile

Read/Write buffer

X,Y Coordinates

Stencil Operation

Chapel Iterator Review

```
iter my_iter( N: int ): int {
  for i in 1..N do yield i;
  for i in N..1 by -1 do yield i;
}

for j in my_iter( 10 ) do
  writeln( j );
```

Iterator Abstraction

Chapel

```
iter DiamondTileIterator( lowerBound: int, upperBound: int, T: int,
                          tau: int,
                          param tag: iterKind): 4*int
                          where tag == iterKind.standalone {
 for kt in -2 .. floord(3*T,tau) {
  var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
  var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
  var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
  var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);

  forall k1 in k1_lb .. k1_ub {
   for x in k2_lb .. k2_ub {
    var k2 = x-k1;
    for t in max(1,floord(kt*tau,3))
          .. min(T,floord((3+kt)*tau-3,3)){

     write = t & 1;
     read = 1 - write;
     for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
           .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-(k1+k2)*tau)) {

      for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
            .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){

       yield (read, write, x, y);
                    A[read, x+1, y] + A[read, x, y+1]+
                    A[read, x  , y] )/5;
} } } } } } }
```

Main Result: Reduction of Code Complexity

Without Iterator

```
for kt in -2 .. floord(3*T,tau) {
 var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
 var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
 var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
 var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);
 forall k1 in k1_lb .. k1_ub {
  for x in k2_lb .. k2_ub {
   var k2 = x-k1;
   for t in max(1,floord(kt*tau,3))
         .. min(T,floord((3+kt)*tau-3,3)){
    write = t & 1;  read = 1 - write;
    for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
          .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-
                 (k1+k2)*tau)) {
     for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
           .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){
      A[write,x,y]=(A[read,x-1,y]+A[read, x, y-1]+ A[read,x+1,y]
                    + A[read,x,y+1]+ A[read,x,y] )/5;
} } } } } }
for kt in -2 .. floord(3*T,tau) {
 var k1_lb: int = floord(3*Lj+2+(kt-2)*tau,tau_times_3);
 var k1_ub: int = floord(3*Uj+(kt+2)*tau-2,tau_times_3);
 var k2_lb: int = floord((2*kt-2)*tau-3*Ui+2,tau_times_3);
 var k2_ub: int = floord((2+2*kt)*tau-2-3*Li,tau_times_3);
 forall k1 in k1_lb .. k1_ub {
  for x in k2_lb .. k2_ub {
   var k2 = x-k1;
   for t in max(1,floord(kt*tau,3))
         .. min(T,floord((3+kt)*tau-3,3)){
    write = t & 1;  read = 1 - write;
    for x in max(Li,(kt-k1-k2)*tau-t,2*t-(2+k1+k2)*tau+2))
          .. min(Ui,min((1+kt-k1-k2)*tau-t-1, 2*t-
                 (k1+k2)*tau)) {
     for y in max(Lj,tau*k1-t,t-x-(1+k2)*tau+1)
           .. min(Uj,(1+k1)*tau-t-1,t-x-k2*tau){
      B[write,x,y]= B[read,x-1,y]+ B[read, x, y-1]+B[read,x+1,y]
                    + B[read, x, y+1]+ 4*B[read,x,y];
} } } } } }
```

With Iterator

```
forall (read, write, x ,y) in
 DiamondTileIterator(L, U, T, tau) {
  A[write, x, y] = ( A[read, x-1, y] +
                     A[read, x, y-1] +
                     A[read, x, y  ] +
                     A[read, x, y+1] +
                     A[read, x+1, y] )/5;
}

forall (read, write, x ,y) in
 DiamondTileIterator(L, U, T, tau) {
  B[write, x, y] =   B[read, x-1, y] +
                     B[read, x, y-1] +
                     4* B[read, x, y  ] +
                     B[read, x, y+1] +
                     B[read, x+1, y];
}
```

Metrics of Success

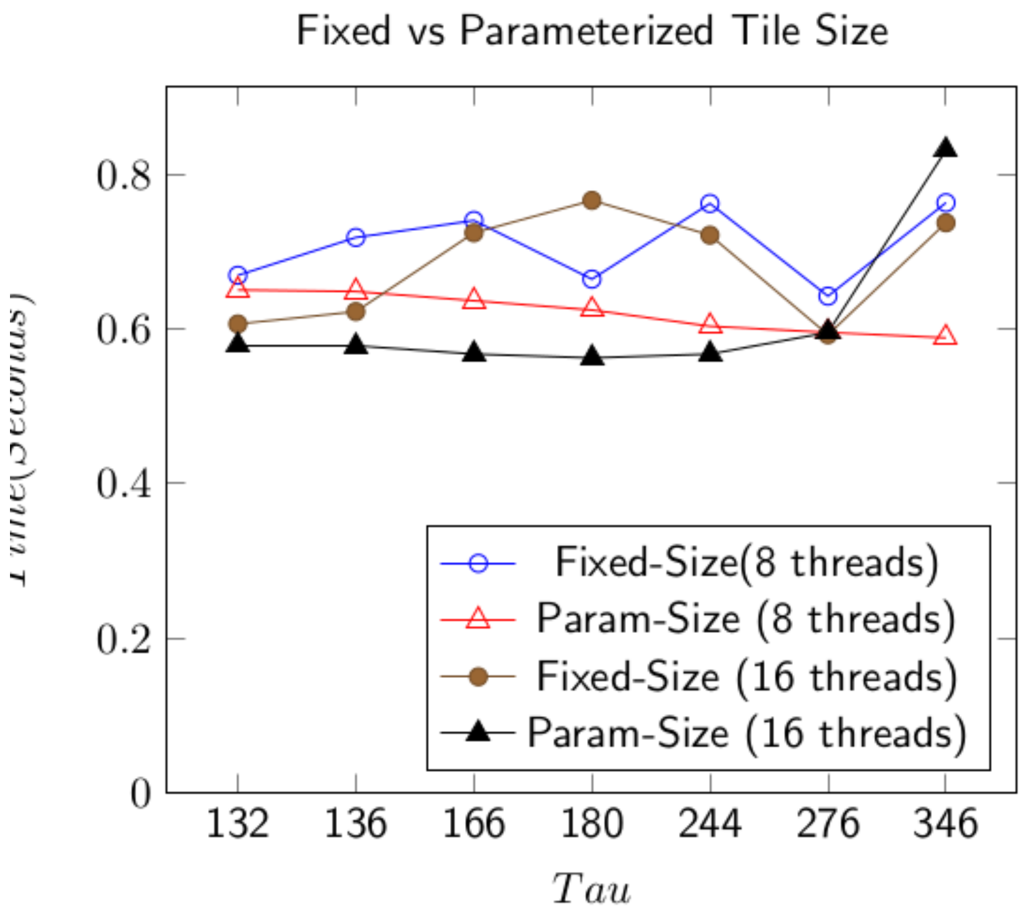
* Parameterized Diamond Tiling is competitive with fixed size Diamond Tiling.

* Chapel iterator performance is competitive with C + OpenMP implementation.

Methodology

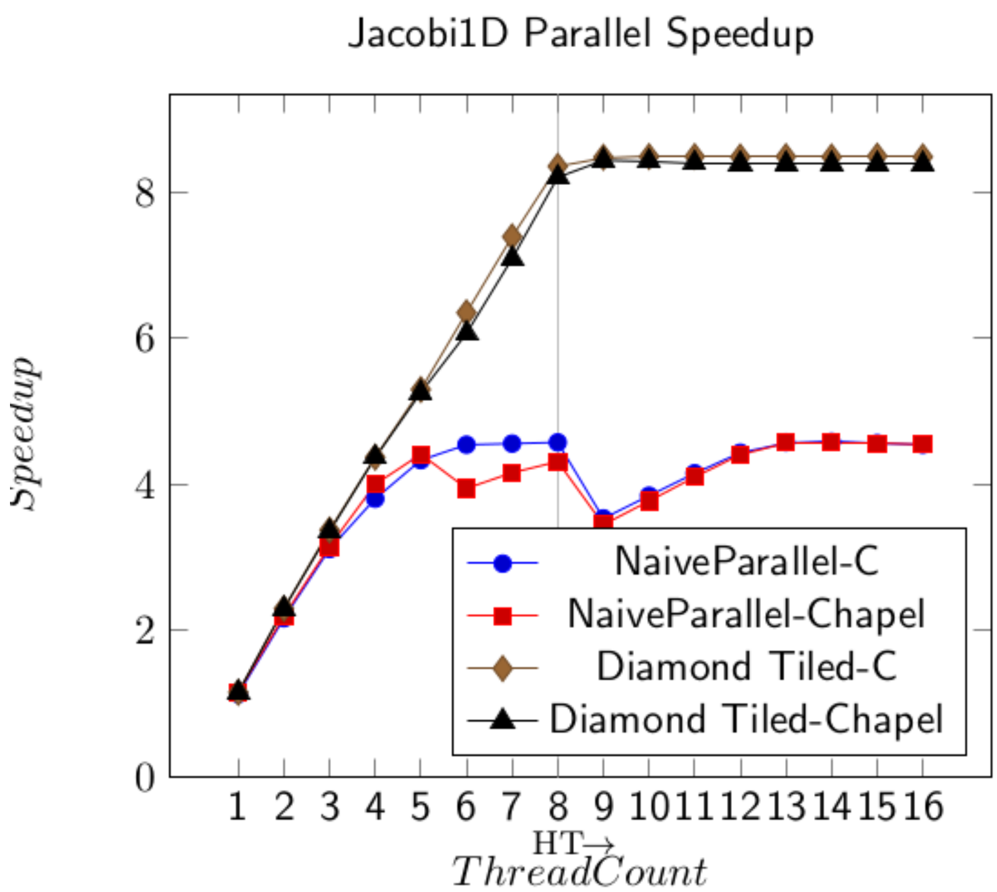
* Hardware:
  * Workstation Machine
  * Single socket Intel Xeon E5
    * 8 Core (16 Hyper-Threads) 2.6GHz
    * 32Kb L1 data, 256Kb L2, 20Mb L3 Cache
  * 32 Gb RAM
* Benchmarks:
  * Jacobi 1D & 2D
  * Problem sizes 2x L3 cache

Parameterized vs Fixed Tile Sizes



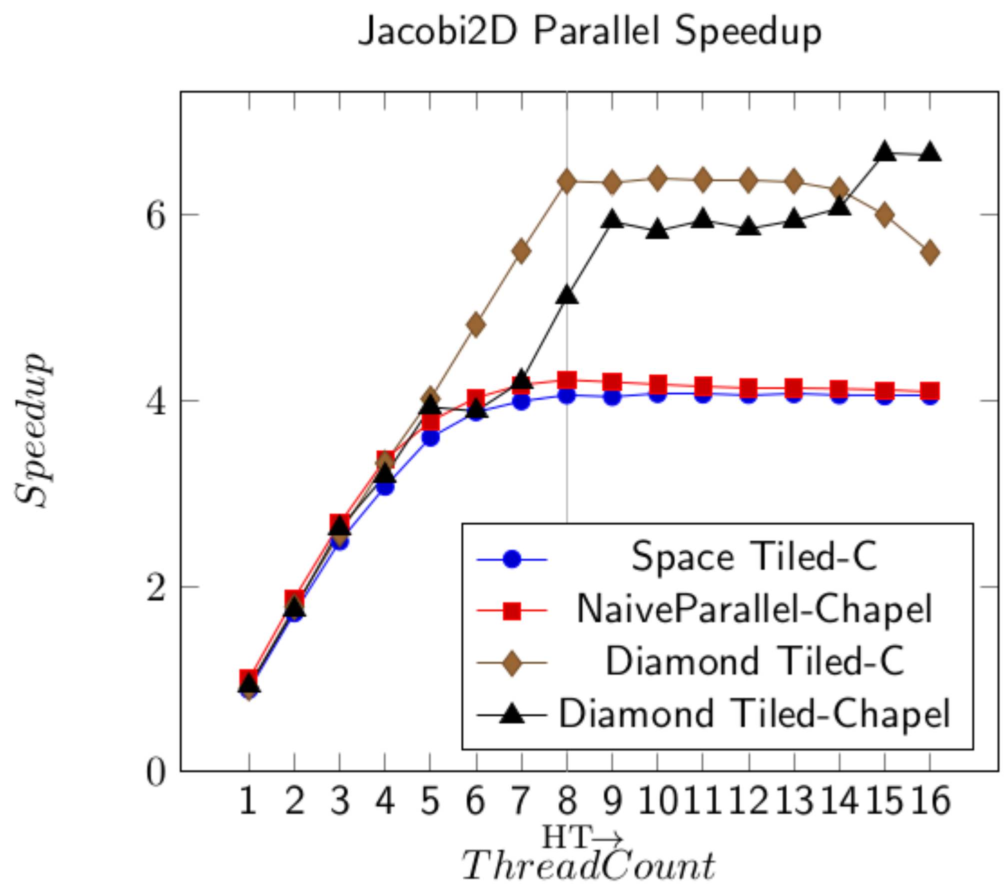
* Parameterized tile size does *better* than fixed.

Competitive Performance



* Maximum Speedup:
  * Chapel: 8.4x
  * C + OpenMP: 8.5x

Competitive Performance



- Maximum Speedup:
  - Chapel: 6.7x
  - C + OpenMP: 6.4x

Conclusion

* Parameterized tile size Diamond Tiling is just as effective as fixed tile size Diamond Tiling.
* Diamond Tiling implemented in Chapel iterators is competitive with Diamond Tiling in C + OpenMP.
* Chapel iterators make advanced tiling schedules much easier to adopt and use.

Bibliography, page 1

[Bandishti et al. 2012] V. Bandishti, I. Pananilath, and U. Bondhugula. Tiling stencil computations to maximize parallelism. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), 2012.

[Bondhugula et al. 2008] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral pro- gram optimization system. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), June 2008.

[Cloog] C. Bastoul. Code generation in the polyhedral model is easier than you think. In Proceedings of the 13th Interntional Conference on Parallel Architecture and Compilation Techniques (PACT), 2004.

[ISCC] Sven Verdoolaege. http://polyhedral.info/2014/01/21/ISCC-demo-online.html, 2014.

[ISL] S. Verdoolaege. ISL: An integer set library for the polyhedral model. In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, Lecture Notes in Computer Science, 2010.

[Omega+] Chun Chen and Mary Hall and Anand Venkat. http://ctop.cs.utah.edu/ctop/?page_id=21, 2012.

[Christen et al. 2011] Matthias Christen and Olaf Schenk and Helmar Burkhart. PATUS: A Code Generation and Autotuning Framework For Parallel Iterative Stencil Computations on Modern Microarchitectures. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2011.

Bibilography, page 2

[Grosser et al. 2014] T. Grosser, S. Verdoolaege, A. Cohen, and P. Sadayappan. The relation between diamond tiling and hexagonal tiling. Parallel Processing Letters, 24(03), 2014.

[Holewinski et al. 2012] J. Holewinski, L. Pouchet, and P. Sadayappan. High-performance code generation for stencil computations on GPU architectures. In Proceedings of the 26th ACM International Conference on Supercomputing, 2012.

[Ragan-Kelley 2013] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, 2013.

[Tang et al. 2011] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The pochoir stencil compiler. In Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures, 2011.

[Yuki et al. 2012] T. Yuki, V. Basupalli, G. Gupta, G. Iooss, D. Kim, T. Pathan, P. Srinivasa, Y. Zou, and S. Rajopadhye. Alphaz: A system for analysis, transformation, and code generation in the polyhedral equational model. Technical report, Technical Report CS-12-101, Colorado State University, 2012.