Orthogonal Scheduling of Stencil Computations with Chapel Iterators

Student:
Ian Bertolacci

Advisors:
Michelle Mills Strout
Catherine Olschanowsky

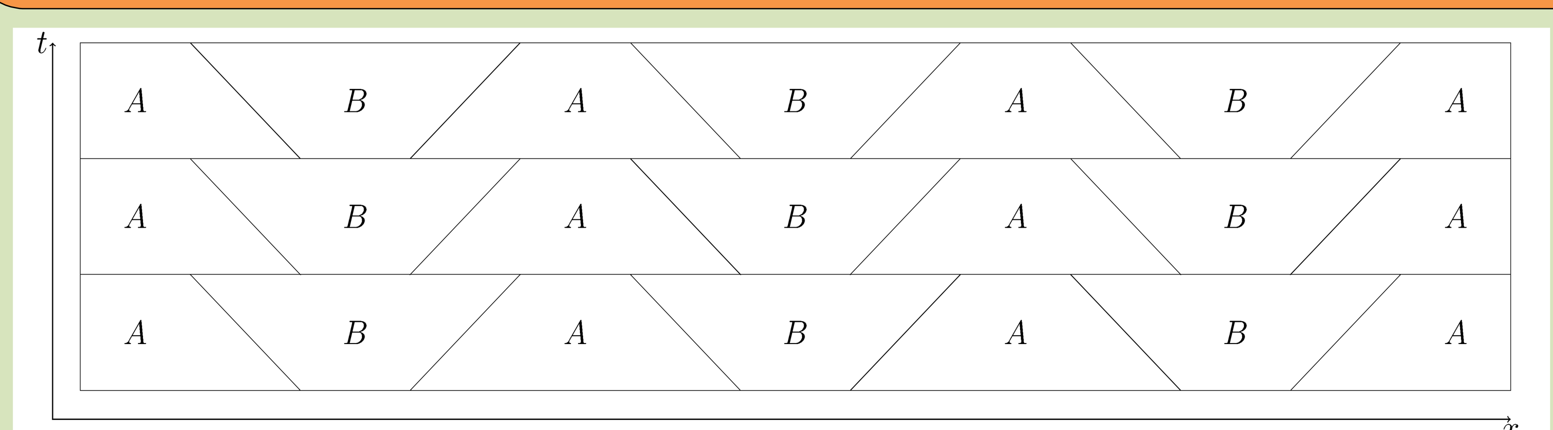
This project is supported by a Department of Energy Early Career Grant DE-SC0003956.

Problem

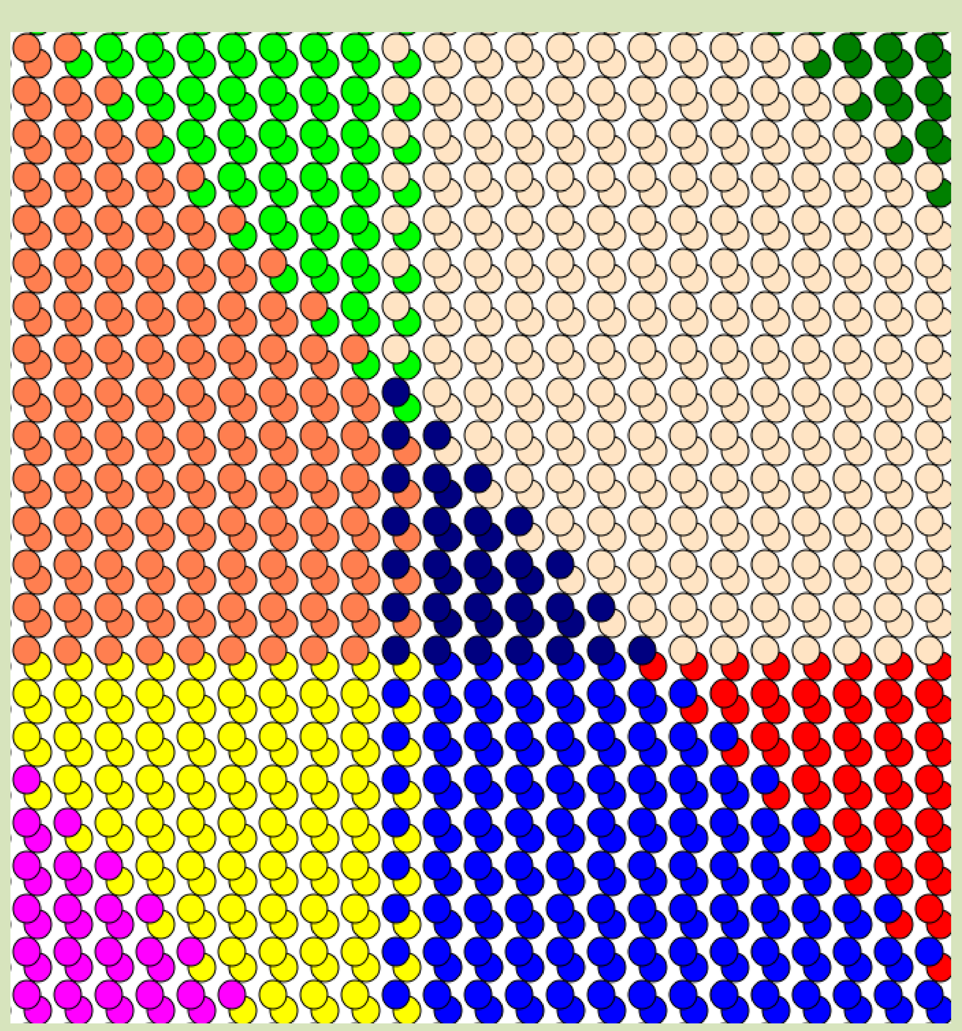
Implementing stencil computations is already a difficult task. The addition of specifying iteration schedules with both good parallelism and high data locality introduces a new dimension of difficulty, and increases the load on the developer.

Compilers already exist to automate the task of optimization, but may lack the information required to generate the most optimal code. Our research investigates whether a library can be developed that gives the programmer a more optimal schedule in a plug-and-play manner without introducing complexity into their code.

1D & 2D Sliced Diamond Tiling [1,2,3]



Depiction of 1D Sliced Diamond Tiling



Depiction of 2D Sliced Diamond Tiling

Programmability

We want to transform our original schedule:

```
for t in timeRange do
  forall (x,y) in spaceDomain do
    computation( t, x, y );
```

into a faster schedule:

```
forall (t,x,y) in slicedDiamondIterator() do
  computation( t, x, y );
```

while avoiding code with equivalent performance, but agony to develop, maintain, or understand:

```
var tau_OVER_3: int = (tau/3);
var s: int = tau_OVER_3 - 2;
if subset_s > s || subset_s < 2
  exit(-1);
var start_s: int = (s/2)-(subset_s/2);
if start_s % 2 == 0 then start_s -=1;
if start_s < 0 then start_s = 1;
var stop_s:int = start_s + subset_s - 1;
var Li: int = 1,
  Lj: int = 1,
  Ui: int = upperBound + 1,
  Uj: int = upperBound + 1;
for toffset in 0 .. T by subset_s {
 var c0: int = -2;
 var c1lb: int = floord(Lj + 1, tau);
 var c1ub: int = floord(Uj -5, tau);
 var c2lb: int = floord(-Ui - Uj + 3, tau);
 var c2ub: int = -floord(Lj + 1, tau) + floord(-Li - 2, tau) - 1;
 forall c1 in c1lb..c1ub {
  forall c2 in c2lb..c2ub {
   for c3 in start_s..min( T-toffset+start_s-1, stop_s) {
     var t: int = c3 + toffset - start_s + 1;
     var write = t & 1;
     var read = 1-write;
    for c4 in maxs(-tau * c1 - tau * c2 + 2 * c3 - (2*tau-2),
          -Uj - tau * c2 + c3 - (tau-2),
          tau * c0 - tau * c1 - tau * c2 - c3,
          Li)
         ..
          mins(tau * c0 - tau * c1 - tau * c2 - c3 + (tau-1),
             -tau * c1 - tau * c2 + 2 * c3,
             -Lj - tau * c2 + c3,
             Ui - 1){
    for c5 in maxs(tau * c1 - c3,
           Lj,
           -tau * c2 + c3 - c4 - (tau-1))
        ..
        mins(Uj - 1,
           -tau * c2 + c3 - c4,
           tau * c1 - c3 + (tau-1)) {
     computation ( read, write, c4, c5);
    }}}}}
 c0 = -1;
 c1lb = floord(Lj + 1, tau);
 c1ub = floord(Uj + (tau_OVER_3)-2, tau);
 c2lb = floord(-Ui - Uj + 3, tau);
 c2ub = floord(-Li - Lj + (tau_OVER_3)-3, tau);
 forall c1 in c1lb..c1ub {
  forall c2 in c2lb..c2ub {
   for c3 in start_s..min(T-toffset+start_s-1,stop_s) {
    var t: int = c3 + toffset - start_s + 1;
    var write = t & 1;
    var read = 1-write;
    for c4 in maxs(-tau * c1 - tau * c2 + 2 * c3 - (2*tau-2),
            -Uj - tau * c2 + c3 - (tau-2),
            tau * c0 - tau * c1 - tau * c2 - c3,
            Li)
         ..
         mins(tau * c0 - tau * c1 - tau * c2 - c3 + (tau-1),
            -tau * c1 - tau * c2 + 2 * c3,
            -Lj - tau * c2 + c3, Ui - 1){
        for c5 in maxs(tau * c1 - c3, Lj,
              -tau * c2 + c3 - c4 - (tau-1))
            ..
            mins(Uj - 1, -tau * c2 + c3 - c4,
              tau * c1 - c3 + (tau-1)){
      computation ( read, write, c4, c5);
     }}}}}
 c0 = 0;
 c1lb = floord(Lj + (tau_OVER_3)+4, tau);
 c1ub = floord(Uj + (tau_OVER_3)-2, tau);
 c2lb = -floord(Uj + (tau_OVER_3)-2, tau) + floord(-Ui-(tau_OVER_3-3),tau)
+1;
 c2ub = floord(-Li - Lj + (tau_OVER_3-3), tau);
 forall c1 in c1lb..c1ub {
  var read: int = 0;
  var write: int = 1;
  forall c2 in c2lb..c2ub {
   for c3 in start_s..min(T-toffset+start_s-1,stop_s) {
    var t: int = c3 + toffset - start_s + 1;
    var write = t & 1;
    var read = 1-write;
    for c4 in maxs(-tau * c1 - tau * c2 + 2 * c3 - (2*tau-2),
          -Uj - tau * c2 + c3 - (tau-2),
          tau * c0 - tau * c1 - tau * c2 - c3,
          Li)
         ..
         mins(tau * c0 - tau * c1 - tau * c2 - c3 + (tau-1),
            -tau * c1 - tau * c2 + 2 * c3,
            -Lj - tau * c2 + c3,
            Ui - 1){
     for c5 in maxs(tau * c1 - c3, Lj,
            -tau * c2 + c3 - c4 - (tau-1) )
          ..
          mins(Uj - 1,
            -tau * c2 + c3 - c4,
            tau * c1 - c3 + (tau-1) ){
       computation ( read, write, c4, c5);
     }}}}}}
```

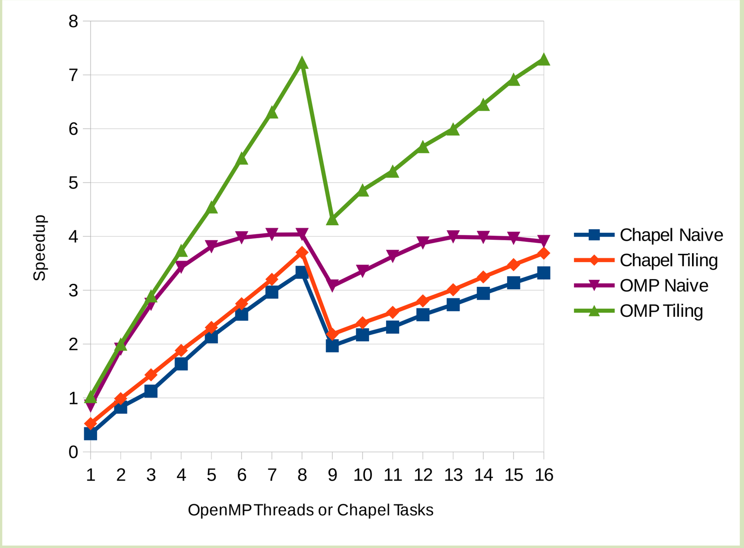
Performance

Speedup for Jacobi

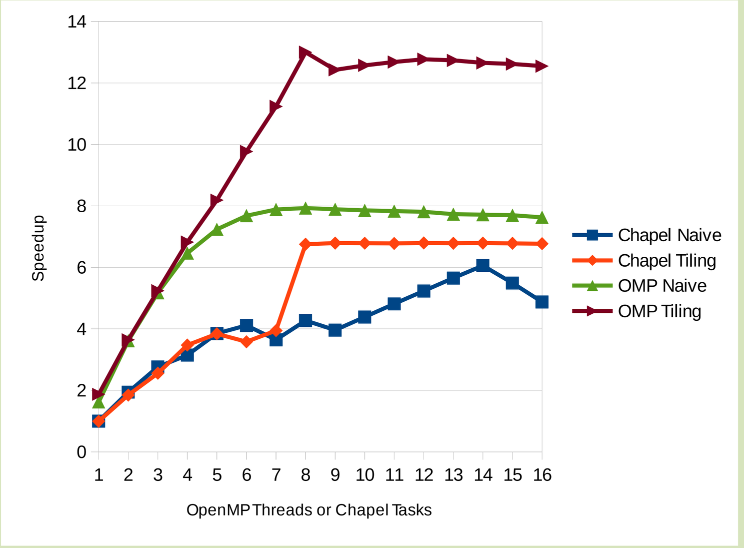
	1D Data		2D Data	
Language	Naive Parallel	Sliced Diamond Tiling	Naive Parallel	Sliced Diamond Tiling
Chapel	3.32x	3.69x	5.96x	6.85x
OpenMP	3.91x	7.29x	7.70x	13.05x

- Tested on 8 core machine with HyperThreading
- Fixing the performance gap between Chapel and OpenMP is a work in progress

Speedup Of Jacobi Across Tasks/Threads

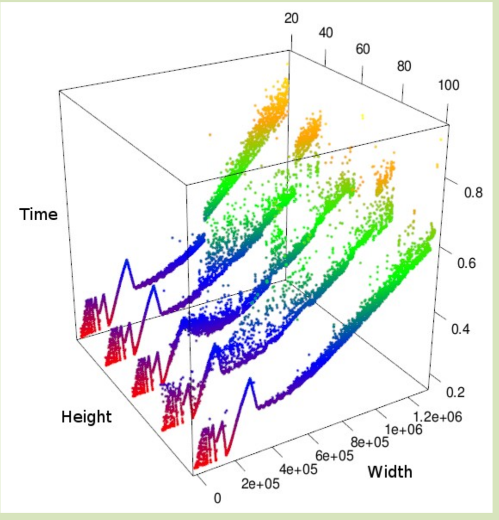


1D Data

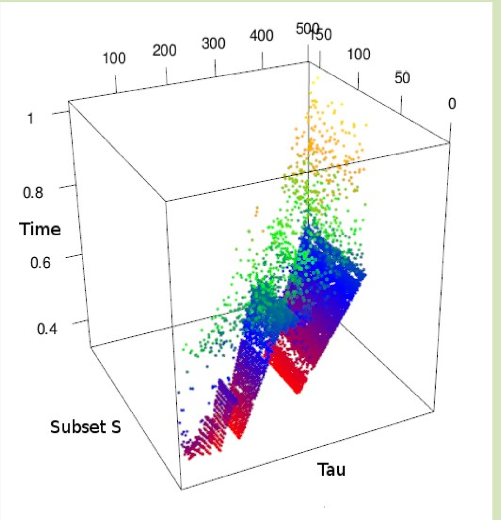


2D Data

Execution-Time Over Tile Size



1D Data



2D Data

Future Work

- Fix performance gap between Chapel and OpenMP
- Creation of Chapel tiling iterators library
- Optimum tile size discovery algorithm
- Generic, N-dimensional, dependency realizing iterators
- OpenMP C parallel iterators

References

1. M. Frigo and V. Strumpen. Cache oblivious stencil computations. *In Proceedings of the 19th annual international conference on Supercomputing - ICS '05*, page 361, New York, New York, USA, June 2005. ACM Press.
2. R. Strzodka, M. Shaheen, and D. Pajak. Time skewing made simple. In *PPOPP*, pages 295–296, 2011.
3. 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.