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 code. Our research investigates whether a library can be developed that gives the programmer a more optimal code.

We want to transform our original schedule:

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

into a faster schedule:

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

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

```plaintext
forall c1 in c1lb..c1ub {
    var c2lb: int = -floord(Lj + 1, tau) + floord(-Ui-(tau_OVER_3-3),tau) - 1;
    var c2ub: int = floord(Uj + (tau_OVER_3)-2, tau);
   forall c2 in c2lb..c2ub {
        var c1ub: int = floord(Uj -5, tau);
        var c0 = -1;
       forall c3 in start_s..min( T-toffset+start_s-1, stop_s) {
            var c3lb: int = floord(Uj + (tau_OVER_3)-2, tau) - Lj;
            var c3ub: int = floord(Uj -5, tau) - Lj;
           forall c4 in maxs(-tau * c1 - tau * c2 + 2 * c3 - (2*tau-2),
                        mins(tau * c0 - tau * c1 - tau * c2 - c3 + (tau-1),
                        mins(Uj - 1, -tau * c2 + c3 - c4, Lj + (tau_OVER_3)+4, tau * c0 - tau * c1 - tau * c2 - c3, tau * c1 - c3 + (tau-1),
                        mins(-tau * c2 + c3 - c4 - (tau-1),
                        mins(-tau * c2 + c3, Lj, -tau * c2 + c3 - c4 - (tau-1) ))
                        tau * c1 - c3 + (tau-1))) {
                var c4lb: int = floord(-Ui - Uj + 3, tau);
                var c4ub: int = -floord(Lj + 1, tau) + floord(-Ui-(tau_OVER_3-3),tau); +1;
               forall c5 in maxs(tau * c1 - c3, Lj, t - S + c0) {
                    var write: int = 1;
                    var read = 1-write;
                    computation ( read, write, c4, c5);
                }
            }
        }
    }
}
```

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: