Automatic Code Generation for Graph Algorithms

Research Problem

Turning mathematical graph algorithms into actual implementations that run at speed is complicated. It requires:

1. **algorithmic design** to identify the appropriate implementable algorithms
2. **tuned implementations** that consider data storage formats and available hardware features

Target Problem

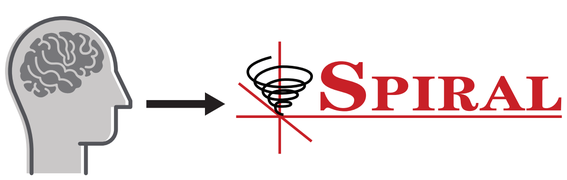
Using triangle counting as an example, we demonstrate our approach to generating graph algorithms from their mathematical specification.

Mathematical Specification

$$\Delta = \frac{1}{6}\Gamma(A^3)$$

Proposed Solution

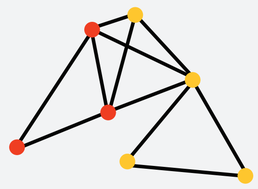
Encode expert knowledge about algorithm design and optimization into an automated system (SPIRAL) to generate tuned implementations automatically. Allow the use of GraphBLAS formulae for providing mathematical specifications.



Algorithmic Design

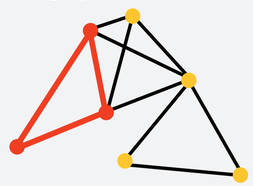
Formally deriving algorithms from the mathematical specification.

Example: Triangle Counting

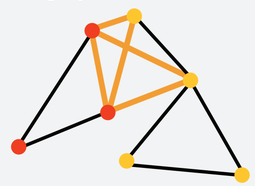


Original graph split into two subgraphs: processed (red) and unprocessed (yellow)

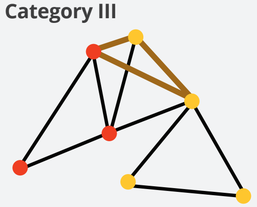
Category I



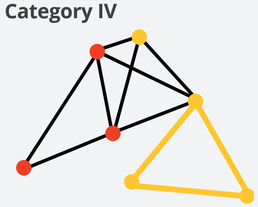
Category II



Category III



Category IV

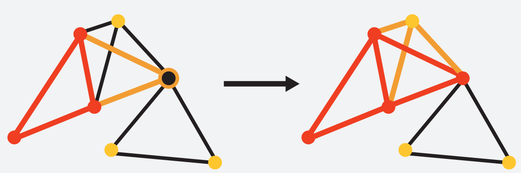


Intuition

Counting different categories of triangles as we iterate over the different vertices yields different algorithms.

Illustration

Counting Category I and II triangles, where red vertices have been processed.



Automatic Code Generation

Formalize the algorithm and implementation techniques into SPIRAL

Program Specification

```
TriangleCount()
```

$$\Delta = \frac{1}{6}\Gamma(A^3)$$

Algorithm Choice

```
Accum_VMV( TriangleCount() )
```

$$L = tril(A)$$
$$C = LL \circ L$$
$$\Delta = Reduce(C)$$

Algorithm Derivation

```
BB(
  Accum(i4, 1, X.N-1,
    Accum_X(i6, [i4, 0], i4,
      Dot([i6, add(i4, V(1)) ], [ i4, add(i4, V(1))],
                sub(sub(X.N, i4), V(1)))
))
```

$$\Delta = \Delta + \frac{1}{2}\alpha_{10} A_{00} \alpha_{01}$$

Abstract Code

```
program(
    func(TVoid, "transform", [ res, IJ ],
        decl([ i6, j131, j1765, j1m31, j231, j2m31, jm32, rf63, rf64 ],
            chain(
                assign(deref(res), V(0)),
                loopf(i4, 1, 262110,
                    chain(
                        assign(rf63, V(0)),
                        assign(j1765, add(V(262112), IJ, nth(IJ, i4))),
                        assign(jm32, add(V(262112), IJ, nth(IJ, add(i4, V(1))))),
                        loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), V(0))),
                            assign(j1765, add(j1765, V(1)))
                        ),
                        loopw(logic_and(lt(j1765, jm32), lt(deref(j1765), i4)),
                            // dot product
                        ),
                        assign(deref(res), add(deref(res), rf63))
) ) ) ) ) )
```

C Code

```
void tc(int *res, int *IJ) {
  for(...) {
    // VMV product
  }
}
```


Dr. Scott McMillan | smcmillan@sei.cmu.edu
with Paul Brouwer, Dr. Daniele Spampinato, Dr. Jason Larkin, Dr. Tze Meng Low, and Dr. Franz Franchetti