Optimizing Integer→Integer Code with the OEIS

Matthew Davis, Benjamin Mikek
Georgia Institute of Technology {mdavis438,bmikek}@gatech.edu

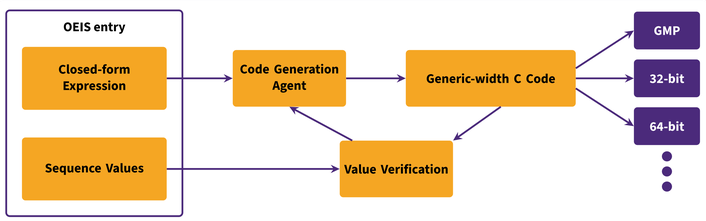
Introduction

Scientific software implements numerical algorithms. These use nested loops and sometimes recursion to count lattice points and triangular index spaces, compute binomial coefficients and perform reductions.

LLVM has limited support for recognizing these patterns. Scalar evolution handles only affine and low-degree polynomial recurrences, while idiom recognition covers only a fixed set of hand-written patterns.

Use the OEIS to help LLVM optimize numerical algorithms. Our pass analyzes input programs to match and replace regions with validated closed-form expressions drawn from the Online Encyclopedia of Integer Sequences (OEIS). Our approach handles both loops and recursion.

Part 1: Sequence Library Construction



We construct a database of 145k+ generic-width C implementations of sequences validated against all listed OEIS values. We support standard widths and arbitrary width with GMP.

Example: DART

Before:

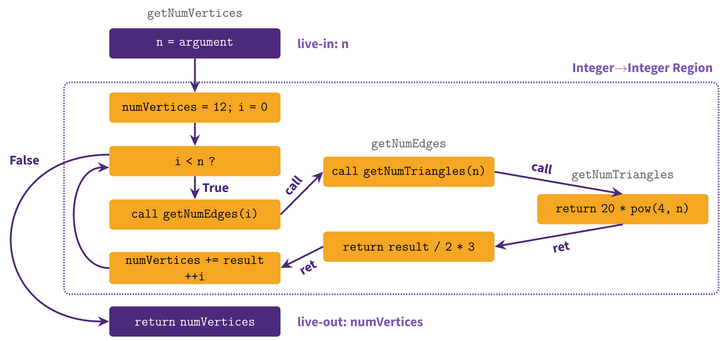
```
size_t getNumVertices(size_t n) {
  size_t numVertices = 12;
  for (auto i = 0u; i < n; ++i)
    numVertices += getNumEdges(i);
  return numVertices;
}
size_t getNumEdges(size_t n) {
  return getNumTriangles(n) / 2 * 3;
}
size_t getNumTriangles(size_t n) {
  return 20 * std::pow(4, n);
}
```

After: A122973

```
size_t getNumVertices(size_t n) {
  return 10 * (1 « (2 * n)) + 2;
}
size_t getNumEdges(size_t n) {
  return 30 * (1 « (2 * n));
}
size_t getNumTriangles(size_t n) {
  return 20 * (1 « (2 * n));
}
```

$$\texttt{getNumVertices}(n) = 12 + \sum_{i=0}^{n-1} 30 \cdot 4^i = 10 \cdot 4^n + 2$$

Part 2: Code Region Extraction

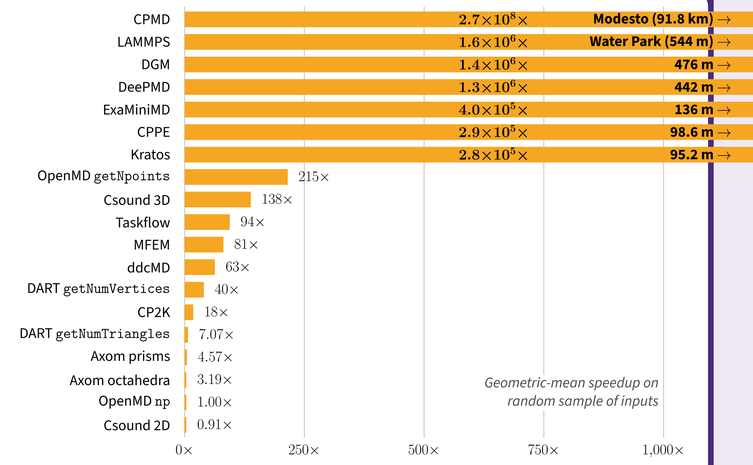


Our approach replaces pure single-entry subgraphs of the ICFG terminated by a post-dominating statement. We use a bidirectional dataflow analysis to detect regions with matched live-in and live-out integer variables.

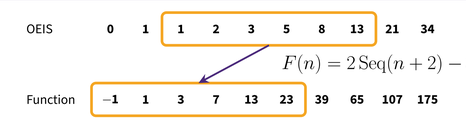
Substituted Functions

Repo	Stars	Function	LOC	Complexity
LAMMPS	3.1k	`compute_ncomps`	7→3	$O(n^3) \to O(1)$
DeePMD	2.1k	`coupling_record_count`	15→6	$O(n^3) \to O(1)$
ExaMiniMD	33	`build_indexlist`	8→7	$O(n^3) \to O(1)$
DGM	0	`ncells`	11→2	$O(n^3) \to O(1)$
CP2K	1.2k	`do_bsse_calculation`	18→3	$O(n^2) \to O(1)$
OpenMD	93	`getNpoints`	4→4	$O(n^2) \to O(1)$
MFEM	2.2k	`InitVolume`	4→1	$O(n) \to O(1)$
Csound	1.5k	`early_init` (3D)	10→8	$O(n) \to O(1)$
Csound	1.5k	`early_init` (2D)	2→1	$O(n) \to O(1)$
Kratos	1.4k	`NumberOfShapeFunctionRows`	6→2	$O(n) \to O(1)$
DART	1.2k	`getNumVertices`	4→1	$O(n) \to O(1)$
Axom	197	`count_sphere_octahedra`	11→2	$O(n) \to O(1)$
Axom	197	`count_segment_prisms`	18→2	$O(n) \to O(1)$
OpenMD	93	`np`	12→2	$O(n) \to O(1)$
CPMD	53	`mm_solv_const`	6→5	$O(n) \to O(1)$
CPPE	24	`Nterms`	5→1	$O(n) \to O(1)$
Taskflow	12.2k	`coprime`	6→1	$O(\log n) \to O(1)$
ddcMD	36	`treesize`	10→1	$O(\log n) \to O(1)$
DART	1.2k	`getNumTriangles`	1→1	$O(1) \to O(1)$
flow5	235	`factorial`	1→1	BUG FIX

Speedup over clang/flang -O3



Part 3: Sequence Matching



Function outputs are matched for arbitrary offset and linear transform: $F(n) = \frac{\text{num} \cdot \text{Seq}(n+\text{off})}{\text{den}} + b$.

Example: OpenMD

Before:

```
int getNpoints(int n) {
  int count = 0;
  for (int i = 0; i <= n; i++)
    count += np(i);
  return count;
}
int np(int n) {
  if (n < 0) return -1;
  else if (n == 0) return 1;
  else if (n == 1) return 12;
  else if (n == 2) return 42;
  else {
    int count = 0;
    count += 12;
    count += (n - 1) * 30;
    for (int i = 1; i <= n - 2; i++)
      count += i * 20;
    return count;
  }
}
```

After: A005902, A005901

```
int getNpoints(int n) {
  if (n <= 0) return 0;
  return ((2 * n + 1) *
    (5 * n * n + 5 * n + 3)) / 3;
}
int np(int n) {
  if (n < 0) return -1;
  if (n == 0) return 1;
  return 10 * n * n + 2;
}
```

$$\texttt{np}(n) = 12 + 30(n-1) + \sum_{i=1}^{n-2} 20i = 10n^2 + 2$$

$$\texttt{getNpoints}(n) = \sum_{i=0}^{n} \texttt{np}(i) = \frac{(2n+1)(5n^2+5n+3)}{3}$$