Interactive Source-to-source Code Optimization with OptiTrust

Thomas Kœhler   Arthur Charguéraud

Inria

Rencontres de la communauté française de compilation — Grenoble, March 2023

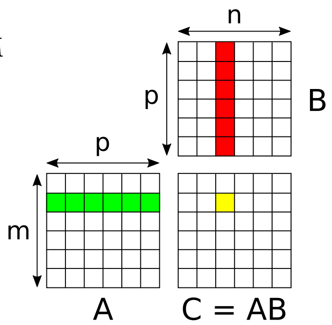
Matrix Multiplication Optimization Example

A standard benchmark to:

- showcase OptiTrust user experience
- compare to user-guided DSL compilers such as TVM

Unoptimized Matrix Multiplication

```
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; j++) {
    float sum = 0.0f;
    for (int k = 0; k < p; k++) {
      sum += A[i][k] * B[k][j];
    }
    C[i][j] = sum;
  }
}
```



Optimization by Hand

```
float* pB = (float*)malloc(sizeof(float[32][256][4][32]));
#pragma omp parallel for
for (int bj = 0; bj < 32; bj++) {
  for (int bk = 0; bk < 256; bk++) {
    for (int k = 0; k < 4; k++) {
      for (int j = 0; j < 32; j++) {
        pB[32768 * bj + 128 * bk + 32 * k + j] =
          B[1024 * (4 * bk + k) + 32 * bj + j]; }}}}
#pragma omp parallel for
for (int bi = 0; bi < 32; bi++) {
  for (int bj = 0; bj < 32; bj++) {
    float* sum = (float*)malloc(sizeof(float[32][32]));
    for (int i = 0; i < 32; i++) {
      for (int j = 0; j < 32; j++) {
        sum[32 * i + j] = 0.; }}
    for (int bk = 0; bk < 256; bk++) {
      for (int i = 0; i < 32; i++) {
        float s[32];
        memcpy(s, &sum[32 * i], sizeof(float[32]));
#pragma omp simd
        for (int j = 0; j < 32; j++) { // k = 0
          s[j] += A[1024 * (32 * bi + i) + 4 * bk + 0] *
                  pB[32768 * bj + 128 * bk + 32 * 0 + j]; }
        // [ ... k = 1, 2, 3 ]
        memcpy(&sum[32 * i], s, sizeof(float[32])); }}
    for (int i = 0; i < 32; i++) {
      for (int j = 0; j < 32; j++) {
        C[1024 * (32*bi + i) + 32*bj + j] = sum[32*i + j]; }}
// [ ... ]
```

- Standard optimizations:
  - improve data locality
    transform loops, change data layout
  - add parallelism
    vectorization, multi-threading
- Time consuming + error prone
- 5× more lines of code
- 150× faster
  4-core Intel CPU

Optimization with TVM

TVM Algorithm = what to compute

```
k = tvm.reduce_axis((0, P))
A = tvm.placeholder((M, P))
B = tvm.placeholder((P, N))

C = tvm.compute((M, N),
  lambda i, j:
  sum(A[i, k] * B[k, j], axis=k))
```

Rewritten Algorithm

```
pB = tvm.compute((N / 32, P, 32),
  lambda bj, k, j:
  B[k, bj * 32 + j])

C = tvm.te.compute((M, N),
  lambda i, j:
  sum(A[i, k] *
      pB[j // 32, k, j % 32],
      axis=k))
```

TVM Schedule = how to compute

```
CC = s.cache_write(C, "global")
bi, bj, i, j = s[C].tile(
  C.op.axis[0], C.op.axis[1], 32, 32)
s[CC].compute_at(s[C], bj)
i2, j2 = s[CC].op.axis
(kaxis,) = s[CC].op.reduce_axis
bk, k = s[CC].split(kaxis, factor=4)
s[CC].reorder(bk, i2, k, j2)
s[CC].vectorize(j2)
s[CC].unroll(k)
s[C].parallel(bi)
bj3, _, j3 = s[pB].op.axis
s[pB].vectorize(j3)
s[pB].parallel(bj3)
```

- Input: restricted DSL
- Transforms an unfamiliar IR
- Effect is hard to visualize

Optimization with OptiTrust

DEMO

Matrix Multiplication Performance

- Intel(R) Core(TM) i7-8665U CPU, AVX2 (8 floats), 4 cores (8 hyperthreads)
- Relative speedup on 1024^3 input:

version	single-thread	multi-thread
unoptimized	1×	1×
optimized	46×	150×
TVM	46×	150×
numpy (Intel MKL)[1]	71×	183×

[1] uses assembly code, explicit vectorization, custom thread library

OptiTrust Script vs TVM Schedule

TVM Schedule = how to compute

```
CC = s.cache_write(C, "global")
bi, bj, i, j = s[C].tile(
  C.op.axis[0], C.op.axis[1], 32, 32)
s[CC].compute_at(s[C], bj)
i2, j2 = s[CC].op.axis
(kaxis,) = s[CC].op.reduce_axis
bk, k = s[CC].split(kaxis, factor=4)
s[CC].reorder(bk, i2, k, j2)
s[CC].vectorize(j2)
s[CC].unroll(k)
s[C].parallel(bi)
bj3, _, j3 = s[pB].op.axis
s[pB].vectorize(j3)
s[pB].parallel(bj3)
```

- Input: restricted DSL
- Transforms an unfamiliar IR
- Effect is hard to visualize

OptiTrust transformation script

```
~~List.iter [("i", 32); ("j", 32); ("k", 4)]
  (fun (loop_id, tile_size) ->
    Loop.tile (trm_int tile_size) ~index:("b" ^ loop_id)
      ~bound:TileDivides [cFor loop_id]);
Loop.reorder_at ~order:["bi"; "bj"; "bk"; "i"; "k"; "j"]
  [cPlusEqVar "sum"];
Loop.hoist_expr ~dest:[tBefore; cFor "bi"] "pB"
  ~indep:["bi"; "i"] [cArrayRead "B"];
Function.inline ~delete:true [cFun "mm"];
Matrix.stack_copy ~var:"sum" ~copy_var:"s" ~copy_dims:1
  [cFor ~body:[cPlusEqVar "sum"] "k"];
Matrix.elim_mops [];
Loop.unroll [cFor ~body:[cPlusEqVar "s"] "k"];
Omp.simd [nbMulti; cFor ~body:[cPlusEqVar "s"] "j"];
Omp.parallel_for [nbMulti;
  cFunDef "mm1024"; dBody; cStrict; cFor ""];
```

- Input: general-purpose language
- Transforms familiar C code
- Effect is visualized on C diffs

Ongoing Work: Justify Correctness with Separation Logic

- Require user annotations on the input program
- Annotations are expressed in a subset of Separation Logic

  local reasoning on read/write permissions
- Compute Separation Logic permissions at every code location
- Leverage annotations to decide whether a transformation is valid
- Transformations may also transform annotations

Ongoing Work: Justify Correctness with Separation Logic

Initial Program Annotations

```
void mm(float* C, float* A, float* B, int m, int n, int p) {
  modifies("C -> matrix(m, n)");
  reads("A -> matrix(m, p) * B -> matrix(p, n)");
  // [...]
}
```

Intermediate Program Annotations

```
for (int bi = 0; bi < exact_div(m, 32); bi++) {

// Omp.parallel_for ~modifies:"C -> matrix_tile ((bi, 32), (0, n)) (m, n)" [...]

matrix_tile_open("C (32, n)"); // consumes 'C -> matrix(m, n)', produces C tiles
#pragma omp parallel for
for (int bi = 0; bi < exact_div(m, 32); bi++) {
  only_this_iteration_modifies("C -> matrix_tile ((bi, 32), (0, n)) (m, n)");
```

Conclusion

- OptiTrust case studies:
  - Particle-In-Cell : numerical simulation in 140 script steps
  - Matrix Multiplication : same performance as TVM in 9 script steps
- Plan to justify transformation correctness with separation logic
- Future work:
  - investigate ease of extensibility *(e.g. image processing optimizations, GPU offloading)*
  - combine C and OptiTrust libraries to implement and optimize DSLs *(languages as libraries)*
  - investigate and improve ease of use *(e.g. semi-automation, optimization hints)*
- Next Talk: optimization of formally verified programs, with formal guarantees

Conclusion

- OptiTrust case studies:
  - Particle-In-Cell : numerical simulation in 140 script steps
  - Matrix Multiplication : same performance as TVM in 9 script steps
- Plan to justify transformation correctness with separation logic
- Future work:
  - investigate ease of extensibility *(e.g. image processing optimizations, GPU offloading)*
  - combine C and OptiTrust libraries to implement and optimize DSLs *(languages as libraries)*
  - investigate and improve ease of use *(e.g. semi-automation, optimization hints)*
- Next Talk: optimization of formally verified programs, with formal guarantees

---

thomas.koehler@thok.eu
thok.eu

Thanks!
