IBM Toronto Software Lab

Interprocedural Strength Reduction

Shimin Cui
Roch Archambault
Raul Silvera
Yaoqing Gao

IBM Toronto Software Lab

Outline

- Introduction
- Examples
- Algorithm in TPO
- Summary

Reduction in Strength

- Replace costly computations with less expensive ones.
- Example of operator strength reduction.
  - Shift, add instead of multiply or divide
  - 16*x → xx << 4
- Well known and widely used in compilers – largely restricted to optimize code within a single procedure.

Interprocedural Strength Reduction - Overview

A method to reduce the costly computations in strength interprocedurally to improve the code performance.

- Precompute costly computations in less frequently executed locations.
- Replace costly computations with less expensive ones.
- Reduce size of global objects referenced repeatedly in loops.

- Provide opportunities for further optimizations.
- Implemented in the link path of IBM product compiler.

Example 1: Global Scalars

- Pre-compute costly computations on global scalar variables.

```
init() {
  // execute count: 1
  a = …;
  b = …;

  c = …;
  d = …;

}
```

```
bar() {
  …
  for (i = 0; i < 1000; ++i) {
     …
     // execute count: 1000
     x += expr(a, b);
     y += c/d;
     …
     foo();
     …
  }
}
```

```
foo() {
  …
  if (cond) {
     // execute count: 10
     b = …;

     d = …;

  }
  …
  // execute count: 1000
  c = …;
  …
}
```

Example 1 *(cont.)*

- Pre-compute costly computations in less frequently executed locations.

```
init() {
  // execute count: 1
  a = …;
  b = …;
  isr1 = expr(a, b);
  c = …;
  d = …;

}
```

```
bar() {
  …
  for (i = 0; i < 1000; ++i) {
     …
     // execute count: 1000
     x += expr(a, b); isr1;
     y += c/d;
     …
     foo();
     …
  }
}
```

```
foo() {
  …
  if (cond) {
     // execute count: 10
     b = …;
     isr1 = expr(a, b);
     d = …;

  }
  …
  // execute count: 1000
  c = …;
  …
}
```

Example 1 *(cont.)*

- Replace costly computations with less expensive computations.

```
init() {
  // execute count: 1
  a = …;
  b = …;
  isr1 = expr(a, b);
  c = …;
  d = …;
  isr2 = (m(d), s(d));
}
```

```
bar() {
  …
  for (i = 0; i < 1000; ++i) {
    …
    // execute count: 1000
    x += expr(a, b) isr1;
    y += c/d; (c*isr2.m)>>isr2.s;
    …
    foo();
    …
  }
}
```

```
foo() {
  …
  if (cond) {
    // execute count: 10
    b = …;
    isr1 = expr(a, b);
    d = …;
    isr2 = (m(d), s(d));
  }
  …
  // execute count: 1000
  c = …;
  …
}
```

Example 2: Global Objects

- Pre-compute costly computations on global arrays and dynamic objects.

```
float **a;
int b[M][N];


int bar() {
   a = malloc(sizeof(float*)*M);

   for (i =0 ; i < M; i++) {
      a[i] = malloc(sizeof(float)*N);

   }
   …
}
```

```
int foo() {
   …
   for (i =0 ; i < M; i++) {
      for (j = 0 ; j < N; j++) {
         if (expra(a[i][j]) > 1) {
            x += 1;
         }
         y += exprb(b[i][j])%256;
         …
      }
      …
   }
   …
}
```

Example 2 *(cont.)*

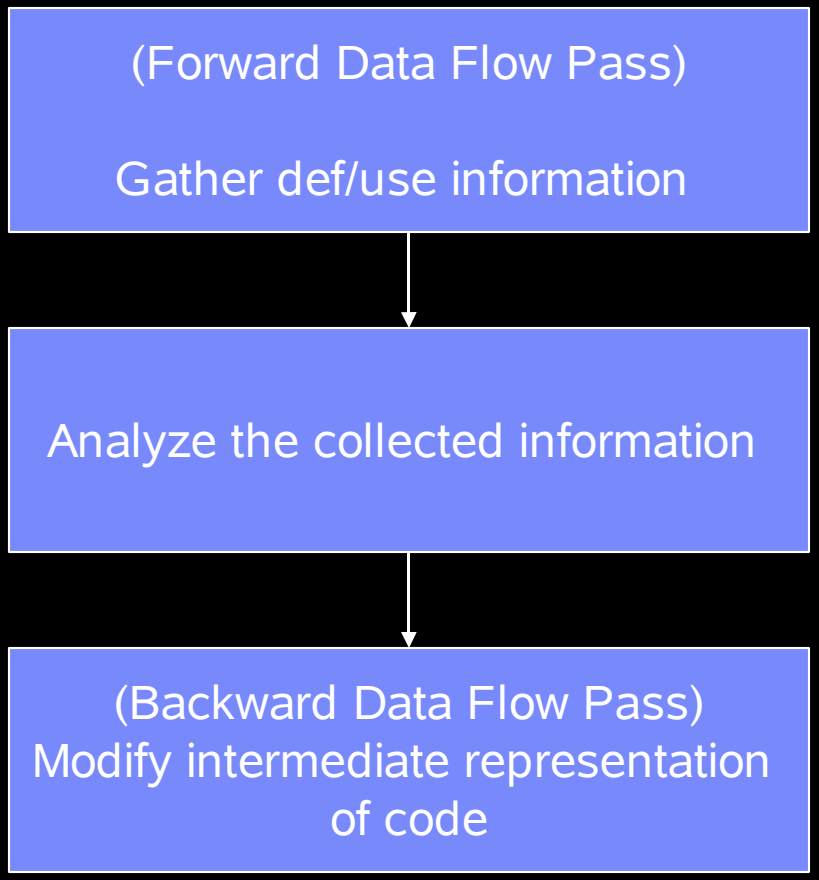
- Reduce size of global objects referenced repeatedly in loops to improve the data cache performance.

```
float **a;
int b[M][N];
bool **isra;
char isrb[M][N];
int bar() {
   a = malloc(sizeof(float*)*M);
   isra = malloc(sizeof(bool*)*M);
   for (i =0 ; i < M; i++) {
      a[i] = malloc(sizeof(float)*N);
      isra[i] = malloc(sizeof(bool)*N);
   }
   …
}
```

```
int foo() {
   …
   for (i =0 ; i < M; i++) {
      for (j = 0 ; j < N; j++) {
         if (expra(a[i][j]) > 1 isra[i][j]) {
            x += 1;
         }
         y += exprb(b[i][j])%256 isrb[i][j];
         …
      }
      …
   }
   …
}
```

Algorithm in IPA Link Phase

- Only supported at -O5 (IPA level 2).



Gather Information

- A global variable is considered only when all of its possible definition locations are known.
  - Aliases of global objects
  - Size of the global objects (static or runtime profile)
- Identify the costly computations which operates only on global variables and collect execution cost related information.
  - Costly computation
  - Computation that can be mapped to an object reference of smaller size data type
- Identify the stores where global variables are modified and collect execution cost related information.

Cost analysis

- Select the candidate computations for reduction in strength based on the cost analysis for the whole program.
  - The total cost for both computations and precomputations
  - The total size of the global objects
- Create a global variable for each selected computation.
  - Initialization of the global variable
  - Indexing or indirect symbols
  - Aliasing symbols

Code Transformation

- Replace the candidate computation by a weaker computation.
  - Load of the created global variable
    - direct, indirect, indexing
  - Multiply-shift sequence using magic number
- Insert the store of global variables (and their aliases) at definition points of all the variables used in the selected computations.
  - Store operation
  - Memory allocation
  - Pointer assignment

Summary

- Precompute costly computations in less frequently executed place
  - To reduce total number of costly computation
- Replace costly computations with less expensive ones
  - To reduce the strength of operation
- Reduce size of global objects referenced repeatedly in loops
  - To improve data cache utilization

Questions?