Optimization

Compiler Construction

2025-01-14

Prof. Dr. Michael Kuhn
michael.kuhn@ovgu.de

Parallel Computing and I/O
Institute for Intelligent Cooperating Systems
Faculty of Computer Science
Otto von Guericke University Magdeburg
https://parcio.ovgu.de

Outline

Optimization

Overview

Optimization

Summary

Optimization

- Optimizations are important to improve performance
  - Straightforward translations are often not very fast
- Optimizations can be applied at different levels
  - Local within a basic block, global within a full function, interprocedural across functions
- Optimizations cannot fix everything
  - $O(n^2)$ algorithms will not be simply replaced by $O(n \cdot \log(n))$ ones

Outline

Optimization

Overview

Optimization

Summary

Group Exercise

- Can the following expressions be optimized?

```
1  i: integer;
2  j: integer;
3  k: float;
4  l: float;
5
6  i = i * 9;
7  i = i * j + i * j;
8  k = k * 9;
9  k = k * l + k * l;
```

Group Exercise

- Can the following expressions be optimized?

```
1  i: integer;
2  j: integer;
3  k: float;
4  l: float;
5
6  i = i * 9;
7  i = i * j + i * j;
8  k = k * 9;
9  k = k * l + k * l;
```

```
1  i = (i << 3) + i;
2  i = (i * j) << 1;
3  k = ???;
4  k = ???;
```

Group Exercise

- Perform loop unrolling for the following snippet. Discuss advantages/disadvantages.

```
int a[];
int b[];
int c[];

for (int i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
}
```

Group Exercise

- Perform loop unrolling for the following snippet. Discuss advantages/disadvantages.

```
int i;
for (i = 0; i < N - 4; i += 4) {
    c[i] = a[i] + b[i];
    c[i + 1] = a[i + 1] + b[i + 1];
    c[i + 2] = a[i + 2] + b[i + 2];
    c[i + 3] = a[i + 3] + b[i + 3];
}
for (; i < N; i++) {
    c[i] = a[i] + b[i];
}
```

- Code size increases
- Can provide opportunities for vectorization

Group Exercise

- Perform function inlining for the following code snippet.

```
1  int foo (int a, int b) {
2      return a + b;
3  }
4
5  int a = 0;
6  for (int i = 0; i < N; i++) {
7      a = foo(a, i) * foo(a, i + 1);
8  }
```

Group Exercise

- Perform function inlining for the following code snippet.

```
int a = 0;
for (int i = 0; i < N; i++) {
    a = (a + i) * (a + i + 1);
}
```

- Code size increases

Group Exercise

- Perform function inlining for the following code snippet.

```
1  int foo (int a, int b) {
2      b = a;
3      return a + b;
4  }
5
6  int a = 0;
7  for (int i = 0; i < N; i++) {
8      a = foo(a, i) * foo(a, i + 1);
9  }
```

Group Exercise

- Perform function inlining for the following code snippet.

```
int a = 0;
for (int i = 0; i < N; i++) {
    int foo1, foo2;
    i = a;
    foo1 = a + i;
    i + 1 = a;
    foo2 = a + i + 1;
    a = foo1 * foo2;
}
```

- Side effects have to be considered

Group Exercise

- Perform function inlining for the following code snippet.

```
1 int a = 0;
2 for (int i = 0; i < N; i++) {
3     a = (a + a) * (a + a);
4 }
```

Group Exercise

- Perform code hoisting for the following code snippet.

```
while (term_iteration > 0) {
    for (i = 1; i < N; i++) {
        for (j = 1; j < N; j++) {
            star = 0.25 * (Matrix[m2][i - 1][j] + Matrix[m2][i][j - 1] +
                ↪ Matrix[m2][i][j + 1] + Matrix[m2][i + 1][j]);
            star += 0.25 * (2 * M_PI * M_PI) * h * h * sin(M_PI * h *
                ↪ (double)i) * sin(M_PI * h * (double)j);

            Matrix[m1][i][j] = star;
        }
    }
}
```

Group Exercise

- Perform code hoisting for the following code snippet.

```
double pih = M_PI * h;
double fpisin = 0.25 * (2 * M_PI * M_PI) * h * h;
while (term_iteration > 0) {
    for (i = 1; i < N; i++) {
        double fpisin_i = fpisin * sin(pih * (double)i);
        for (j = 1; j < N; j++) {
            star = 0.25 * (Matrix[m2][i - 1][j] + Matrix[m2][i][j - 1] +
                ↪ Matrix[m2][i][j + 1] + Matrix[m2][i + 1][j]);
            star += fpisin_i * sin(pih * (double)j);

            Matrix[m1][i][j] = star;
        }
    }
}
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
int foo (int* bar) {
    bool b = false;
    if (b && bar == NULL) {
        ...
        return 1;
    } else {
        ...
        return 2;
    }
    return 3;
}
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
1  int foo (int* bar) {
2      bool b = false;
3      if (b && bar == NULL) {
4          ...
5          return 1;
6      } else {
7          ...
8          return 2;
9      }
10 }
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
1 int foo (int* bar) {
2     ...
3     return 2;
4 }
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
1 int foo (int* bar) {
2     int v = *bar;
3     if (bar == NULL) {
4         return NULL;
5     }
6     return *bar;
7 }
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
int foo (int* bar) {
    if (bar == NULL) {
        return NULL;
    }
    return *bar;
}
```

Group Exercise

- Perform dead code elimination for the following code snippet.

```
1 int foo (int* bar) {
2     return *bar;
3 }
```

- Might lead to undefined behavior

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly
- `-O1`: 787 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly
- `-O1`: 787 lines of assembly
- `-O2`: 830 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly
- `-O1`: 787 lines of assembly
- `-O2`: 830 lines of assembly
- `-O3`: 1,016 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly
- `-O1`: 787 lines of assembly
- `-O2`: 830 lines of assembly
- `-O3`: 1,016 lines of assembly
- `-Ofast`: 1,126 lines of assembly

Group Exercise

- Real-world compiler optimizations using partdiff and GCC 14
- `-O0`: 1,447 lines of assembly
- `-Og`: 923 lines of assembly
- `-O1`: 787 lines of assembly
- `-O2`: 830 lines of assembly
- `-O3`: 1,016 lines of assembly
- `-Ofast`: 1,126 lines of assembly
- `-Os`: 679 lines of assembly

Outline

Optimization

Overview

Optimization

Summary

Summary

- Be careful about correctness
  - For example, order of floating point operations
- Be careful about external side effects
  - For example, system calls like `write`
- Be careful about changes to debugging
  - For example, breakpoints in inlined functions
- Interactions can be positive or negative
  - For example, constant propagation before reachability analysis

References

[Thain, 2020] Thain, D. (2020). ***Introduction to Compilers and Language Design: Second Edition.*** http://compilerbook.org/.