Patrick Woodrum - pwoodru

CPSC 3300

Project 2 – Loop Optimizations

Due Date: 11:59 PM, April 5 on Canvas

Part 1: Describe 3 (working individually) loop optimization techniques and use code examples to show the transformations before and after the optimizations.

Technique 1: **Loop Unrolling**

Loop unrolling is a loop optimization technique that optimizes the execution time of a program by removing loop control instructions as well as loop test instructions. It essentially removes or reduces iterations in a loop and provides the code directly to the compiler as intended [4].

Example Initial Code:

```
for (int a = 0; a < 3; a++) {
        printf("Example\n");
}
```

Example Unrolled Loop Code:

```
printf("Example\n");
printf("Example\n");
printf("Example\n");
```

(based on sample code from [4])

Technique 2: **Loop Jamming**

Loop Jamming is when multiple loops are combined to form a single loop that accomplishes the goals of both of its predecessors. Loop jamming can reduce the time it takes to compile many loops and make looping groups of information more efficient [4].

Example Initial Code:

```
for (int a = 0; a < 3; a++) {
        a += 3;
}
for (int b = 0; b < 3; b++) {
        b -= 3;
}
```

Example Loop Jammed Code:

```
for (int i = 0; i < 3; i++) {
        a += 3;
        b-= 3;
}
```

(based on sample code from [4])

Technique 3: **Loop Peeling**

Loop peeling is a form of loop splitting which simplifies a loop's problematic first iteration by performing that iterations outside of the loop before entering it. Loop splitting is a technique that breaks loops into multiple parts to help reduce dependencies. Loop peeling helps that further by manually removing problematic iterations of a loop [5].

Example Initial Code:

```
for (int i = 0; i < 50; i++) {
        if(i==0)
                a[i] = 1;
        else
                b[i] = 1;
}
```

Example Loop Peeled Code:

```
a[0] = 1;
for (int i = 1; i < 50; i++) {
        b[i] = 1;
}
```

(based on sample code from [5])

Part 2: Definition and explanation of terms

Loop-carried dependencies exist when a statement in one iteration of a loop depends in some way on a statement in a different iteration of the same loop. When one statement in an iteration depends on another, that statement cannot proceed until its dependent statement has completed processing [1] [3].

Example:

```
for (int i = 0; I < 4; i++) {
        S1: b[i] = 4;
        S2: a[i] = b[i-1] + 10;
} [1]
```

Statement S2 cannot proceed until statement S1 completes because S2 is dependent on S1. Loop-carried dependencies can affect the ability to optimize loops based on the dependencies present inside the iterations. Loop splitting techniques may not be possible as different parts of the iterations rely on each other. Alongside this, dependencies require that statements occur together or directly following one another and ensures that they compile together.

Array Element Aliasing means using a type other than the type of value that the variable has stored in it, and with arrays means potentially using an index value that's out of bounds of the accessed array and accessing an object stored next to the array [2]. Aliasing to array elements allows for optimizations as far as accessing incompatible types, distinct objects, and distinct elements of the

array. Array element aliasing makes arrays optimized to be able to return values or types for elements that would otherwise throw warnings, errors, or return values unintended. These optimizations can further improve loop optimization techniques by speeding up incompatible type returns and array accessing.

Part 3: GNU Compiler Hands on Test Results

Compiler Invocation and Flags	Ex. Time (in seconds)
gcc whetstone.c -lm	3.654
gcc whetstone.c -lm -O1	2.153
gcc whetstone.c -lm -O2	1.379
gcc whetstone.c -lm -O3	1.314
gcc whetstone.c -lm -Os	2.102
gcc whetstone.c -lm -Ofast	0.354
gcc whetstone.c -lm -Og	2.419
gcc whetstone.c -lm -funroll-loops	3.672
gcc whetstone.c -lm -Ofast -funroll-loops	0.295
gcc whetstone.c -lm -Ofast -funroll-loops -fmerge-all-constants	0.296
gcc whetstone.c -lm -Ofast -funroll-loops -fwhole-program	0.295
gcc whetstone.c -lm -Ofast -funroll-loops -fgcse-las	0.289
gcc whetstone.c -lm -Ofast -funroll-loops -fgcse-sm	0.290
gcc whetstone.c -lm -Ofast -funroll-loops -ftree-loop-im	0.284

SoC computer: babbage32.computing.clemson.edu

Processor: Intel® Core™ i7-4770 CPU @ 3.40 GHz

OS: Ubuntu 20.04.2 LTS

GCC Version: 9.3.0 (Ubuntu 9.3.0-17ubuntu1-20.04)

Sources

[1] https://people.engr.ncsu.edu/efg/506/s10/www/lectures/notes/lec5.pdf

[2] 2, Martin Sebor June. “The Joys and Perils of C and C++ Aliasing, Part 1.” *Red Hat Developer*, developers.redhat.com/blog/2020/06/02/the-joys-and-perils-of-c-and-c-aliasing-part-1/#:~:text=Typically%2C%20aliasing%20means%20using%20a,stored%20adjacent%20to%20the%20array.

[3] “Loop Dependence Analysis.” *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Loop_dependence_analysis#:~:text=Loop%2Dcarried%20dependencies%20and%20loop,a%20loop%2Dcarried%20dependence%20exists.

[4] *Loop Optimization in Compiler Design*. 21 Nov. 2019, www.geeksforgeeks.org/loop-optimization-in-compiler-design/.

[5] “Loop Optimization Techniques: Set 2.” *GeeksforGeeks*, www.geeksforgeeks.org/loop-optimization-techniques-set-2/.