Low-Level Software Optimization

Sommerakademie in Leysin
AG 2 – Effizientes Rechnen

Christoph Strößner

Technische Universität München

25th August 2016

Studienstiftung des deutschen Volkes

Low Level Software Optimization

Goals

make software run faster, less energy consuming

Low Level Software Optimization

Goals

make software run faster, less energy consuming

auxiliary condition: keep the result identical

Low Level Software Optimization

Goals

make software run faster, less energy consuming
auxiliary condition: keep the result identical

Ansatz

use compiler optimization methods

1. Issues causing slow execution of code

2. Overview of the various possible techniques
   - Compile Time Optimization
     - Different Optimization Levels
     - Simplification of Equations
     - Loop Unrolling
     - Further Code Optimization
     - More Technical Compliler Optimization
     - Memory Optimization
     - Compiler Optimization Example GCC

Issues causing slow execution of code

Most code would be slow if executed like it is written:

Issues causing slow execution of code

Most code would be slow if executed like it is written:

- many unnecessary/redundant assignments
  - inefficient loop usage
  - inefficient equation form
- "modern programmers" care little about memory loading
- parallelism is only exploited partially

Different Optimization Levels

1. general design
2. algorithms and datastructures
3. **source code level**
4. **build level**
5. **compile level**
6. **assembly level**
7. run time

Simplification of Equations

Example: 3 Point Interpolation

given points $x_1,\ x_2,\ x_3\ \in X \subset \mathbb{R}$, function $f : X \to \mathbb{R}$ and values $y_i = f(x_i)$
find polynomial $p(x) = a_2x^2 + a_1x + a_0$ such that $p(x_i) = y_i$

Simplification of Equations

Example: 3 Point Interpolation

given points $x_1,\ x_2,\ x_3\ \in X \subset \mathbb{R}$, function $f : X \to \mathbb{R}$ and values $y_i = f(x_i)$
find polynomial $p(x) = a_2x^2 + a_1x + a_0$ such that $p(x_i) = y_i$

$$a_0 = \frac{-x_2^2x_3y_1+x_2x_3^2y_1+x_1^2x_3y_2-x_1x_2^2y_3+x_1x_2^2y_3-x_1x_3^2y_2}{(x_1-x_2)(x_1x_2-x_1x_3-x_2x_3+x_3^2)}$$
$$a_1 = \frac{x_2^2y_1-x_3^2y_1-x_1^2y_2+x_2^2y_2+x_1^2y_3-x_2^2y_3}{(x_1-x_2)(x_1x_2-x_1x_3-x_2x_3+x_3^2)}$$
$$a_2 = -\frac{-(x_3-x_1)(y_1-y_2)+(-x_1+x_2)(y_1-y_3)}{-(x_1^2+x_2^2)(-x_1+x_3)+(-x_1+x_2)(-x_1^2+x_3^2)}$$

Simplification of Equations

$$a_0 = \frac{-x_2^2x_3y_1+x_2x_3^2y_1+x_1^2x_3y_2-x_1x_2^2y_3+x_1x_2^2y_3-x_1x_3^2y_2}{(x_1-x_2)(x_1x_2-x_1x_3-x_2x_3+x_3^2)}$$

$$a_1 = \frac{x_2^2y_1-x_3^2y_1-x_1^2y_2+x_2^2y_2+x_1^2y_3-x_2^2y_3}{(x_1-x_2)(x_1x_2-x_1x_3-x_2x_3+x_3^2)}$$

$$a_2 = -\frac{-(x_3-x_1)(y_1-y_2)+(-x_1+x_2)(y_1-y_3)}{-(x_1^2+x_2^2)(-x_1+x_3)+(-x_1+x_2)(-x_1^2+x_3^2)}$$

Simplification of Equations

$$a_0 = \frac{-x_2^2 x_3 y_1 + x_2 x_3^2 y_1 + x_1^2 x_3 y_2 - x_1 x_2^2 y_3 + x_1 x_2^2 y_3 - x_1 x_3^2 y_2}{(x_1 - x_2)(x_1 x_2 - x_1 x_3 - x_2 x_3 + x_3^2)}$$
$$a_1 = \frac{x_2^2 y_1 - x_3^2 y_1 - x_1^2 y_2 + x_2^2 y_2 + x_1^2 y_3 - x_2^2 y_3}{(x_1 - x_2)(x_1 x_2 - x_1 x_3 - x_2 x_3 + x_3^2)}$$
$$a_2 = -\frac{-(x_3 - x_1)(y_1 - y_2) + (-x_1 + x_2)(y_1 - y_3)}{-(x_1^2 + x_2^2)(-x_1 + x_3) + (-x_1 + x_2)(-x_1^2 + x_3^2)}$$

$$q_1 = x_1^2, q_2 = x_3^2, q_3 = x_2^2, q_4 = x_1 - x_3, q_5 = \frac{1}{q_4(x_1 - x_2)(x_2 - x_3)}$$
$$a_0 = q_5(-q_4 x_1 x_3 y_2 + x_2(-q_2 y_1 + q_1 y_3) + q_3(x_3 y_1 - x_1 y_3))$$
$$a_1 = q_5(q_2(y_1 - y_2) + q_1(y_2 - y_3) + q_3(-y_1 + y_3))$$
$$a_2 = q_5(x_3(-y_1 + y_2) + x_2(y_1 - y_3) + x_1(-y_2 + y_3))$$

Simplification of Equations

$$a_0 = \frac{-x_2^2 x_3 y_1 + x_2 x_3^2 y_1 + x_1^2 x_3 y_2 - x_1 x_2^2 y_3 + x_1 x_2^2 y_3 - x_1 x_3^2 y_2}{(x_1 - x_2)(x_1 x_2 - x_1 x_3 - x_2 x_3 + x_3^2)}$$

$$a_1 = \frac{x_2^2 y_1 - x_3^2 y_1 - x_1^2 y_2 + x_2^2 y_2 + x_1^2 y_3 - x_2^2 y_3}{(x_1 - x_2)(x_1 x_2 - x_1 x_3 - x_2 x_3 + x_3^2)}$$

$$a_2 = -\frac{-(x_3 - x_1)(y_1 - y_2) + (-x_1 + x_2)(y_1 - y_3)}{-(x_1^2 + x_2^2)(-x_1 + x_3) + (-x_1 + x_2)(-x_1^2 + x_3^2)}$$

$$q_1 = x_1^2, q_2 = x_3^2, q_3 = x_2^2, q_4 = x_1 - x_3, q_5 = \frac{1}{q_4(x_1 - x_2)(x_2 - x_3)}$$

$$a_0 = q_5(-q_4 x_1 x_3 y_2 + x_2(-q_2 y_1 + q_1 y_3) + q_3(x_3 y_1 - x_1 y_3))$$

$$a_1 = q_5(q_2(y_1 - y_2) + q_1(y_2 - y_3) + q_3(-y_1 + y_3))$$

$$a_2 = q_5(x_3(-y_1 + y_2) + x_2(y_1 - y_3) + x_1(-y_2 + y_3))$$

116 flops $\rightarrow$ 55 flops

Simplification of Equations

avoids multiple identical operations

More Code

faster but less readable

Simplification of Equations

avoids multiple identical operations

More Code

faster but less readable

Computer Algebra Systems

automated optimization possible

Numerical Stability

thorough analysis required if more than **common subexpression elimination** is applied
e.g.: $a_2x^2 + a_1x + a_0$ vs. $x(a_2x + a_1) + a_0$

Loop Unrolling

Example: Matrix Multiplication $M = AB$ (for fixed dimensions)

for $i = 1 : n$
  for $j = 1 : n$
    $M_{i,j} = 0$
    for $k = 1 : n$
      $M_{i,j} = M_{i,j} + A_{i,k} B_{k,j}$

Loop Unrolling

for $i = 1 : n$
  for $j = 1 : n$
    $M_{i,j} = 0$
    for $k = 1 : n$
      $M_{i,j} = M_{i,j} + A_{i,k} B_{k,j}$

for fixed dimension (e.g. $n = 3$):
$M_{1,1} = A_{1,1} B_{1,1} + A_{1,2} B_{2,1} + A_{1,3} B_{3,1}$,
$\ldots$
$M_{3,3} = A_{3,1} B_{1,3} + A_{3,2} B_{2,3} + A_{3,3} B_{3,3}$

Loop Unrolling

Faster Execution

no iteration variables needed

Way More Code

less readable
can be done partially

Loop Unrolling

Faster Execution
no iteration variables needed

Way More Code
less readable
can be done partially

Automatic Code Generation
can be generated easily using the original code
could even by done during runtime within "free periods"

Size must be fixed and known
often true for embedded systems

Can be improved by common subexpression elimination

Further Code Optimization

inlining

code instead of function call

calculate constants

use values directly instead of formula expressions

Further Code Optimization

inlining

code instead of function call

calculate constants

use values directly instead of formula expressions

loop inversion

if do while instead of for $\rightarrow$ saves 2 jump statements

factoring out invariants

move invariant lines out of loops

Further Code Optimization

inlining

code instead of function call

calculate constants

use values directly instead of formula expressions

loop inversion

if do while instead of for $\rightarrow$ saves 2 jump statements

factoring out invariants

move invariant lines out of loops

remove recursion

iteration is faster when possible

many more

Compiler Phases

1. lexical analysis
2. parsing
3. semantic analysis
4. optimization
5. code generation

Optimization Timing

Abstract Syntax Tree Optimization

machine independent, too complex for certain applications

Assembly Language Optimization

machine dependent, suitable

Intermediate language

machine independent, suitable

Intermediate Language

1. register adresses: $x, y, a, b, \cdots$
2. basic operations: $+, *, \cdots$
3. jumps to jump labels: $L:$
4. conditional jumps: *if A goto L*

Intermediate Language

1. register adresses: $x, y, a, b, \cdots$
2. basic operations: $+, *, \cdots$
3. jumps to jump labels: $L:$
4. conditional jumps: *if A goto L*

basic blocks: sequence without interior jumps or labels
$\rightarrow$ can be optimized by simplifying equations

Control Flow Graph

Control Flow Graph

basic blocks as vertices
jumps as edges

Control Flow Graph

Control Flow Graph

basic blocks as vertices

jumps as edges

Usage

prefetching and branch prediction

Optimization Types

Local Optimization

single basic block

Global Optimization

control flow graph

Optimization Types

Local Optimization

single basic block

Global Optimization

control flow graph

goal

find a good mix of optimization methods

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

$x = 4,\ y = 2,\ z = x + y \rightarrow$ compute at compile time

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

$x = 4,\ y = 2,\ z = x + y \rightarrow$ compute at compile time

eliminate unreachable blocks

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

$x = 4,\ y = 2,\ z = x + y \rightarrow$ compute at compile time

eliminate unreachable blocks

Single Assignment Form

assign each register only once in the block
$x =$ always the first use f x

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

$x = 4,\ y = 2,\ z = x + y \rightarrow$ compute at compile time

eliminate unreachable blocks

Single Assignment Form

assign each register only once in the block
$x =$ always the first use f x
$\rightarrow$ direct common subexpression elimination
$\rightarrow$ simplifies other optimization (e.g. dead code elimination)

Example: Local Optimizaton

$x = x + 0 \rightarrow$ delete

$x = x * 0 \rightarrow$ simplify: $x = 0$

$x = x * 2 \leftrightarrow$ bit-shift $x << 1$ (machine dependent)

$x = 4,\ y = 2,\ z = x + y \rightarrow$ compute at compile time

eliminate unreachable blocks

Single Assignment Form

assign each register only once in the block
$x =$ always the first use f x
$\rightarrow$ direct common subexpression elimination
$\rightarrow$ simplifies other optimization (e.g. dead code elimination)

Optimization

apply several methods until they do not change the result

Example: Peephole Optimization

Concept on Assembly level

consider a very small section of assembly code

Example: Peephole Optimization

Concept on Assembly level

- consider a very small section of assembly code
- if possible replace this section with a known better equivalent expression

Example: Peephole Optimization

Concept on Assembly level

- consider a very small section of assembly code
- if possible replace this section with a known better equivalent expression
- repeat this over and over again

Conclusion Compile Time Optimization

many possibilities to improve the code

concepts ranging from source code to assembly level

can be performed one after another

Conclusion Compile Time Optimization

many possibilities to improve the code

concepts ranging from source code to assembly level

can be performed one after another

depend on actual optimization goal (memory / runtime)

depend on the target system e.g.: memory access optimization

Conclusion Compile Time Optimization

- many possibilities to improve the code
  - concepts ranging from source code to assembly level
  - can be performed one after another
  - depend on actual optimization goal (memory / runtime)
  - depend on the target system e.g.: memory access optimization
- target systems can be set within the build phase
  - allows specific compile optimization
- most compilers have options to perform optimization

Conclusion Compile Time Optimization

many possibilities to improve the code

concepts ranging from source code to assembly level
can be performed one after another
depend on actual optimization goal (memory / runtime)
depend on the target system e.g.: memory access optimization

target systems can be set within the build phase

allows specific compile optimization

most compilers have options to perform optimization

but they are not perfect

Memory Optimization

loop optimization

parallelizable $c_i = a_i + b_i$

loop-carried dependencies $c_i = a_i + c_{i-1}$

loop nests $\rightarrow$ loop trafos (interchange indizes)

such that parallelizable and data elements positioned "close together"

Memory Optimization

loop optimization

parallelizable $c_i = a_i + b_i$

loop-carried dependencies $c_i = a_i + c_{i-1}$

loop nests $\rightarrow$ loop trafos (interchange indizes)

such that parallelizable and data elements positioned "close together"

$\rightarrow$ less memory energy consumption

Memory Optimization

Main memory optimization

Memory Optimization

Main memory optimization

goals

use the advantages of:

- burst access mode:
  - access a sequence of memory locations
- paged memory:
  - access the same page
- banked memory
  - parallel access to different components

Memory Optimization

Cache optimization

Memory Optimization

Cache optimization

goals

- keep important variables accessible
- improve access speed
  - pull several values with a single cache

Memory Optimization

Cache optimization

goals

- keep important variables accessible
- improve access speed
  - pull several values with a single cache

methods

- place arrays in cache lines in access order
- move different data to different cache locations
- count relative access numbers → placement heuristics

Compiler Optimization Example

Procedure Restructuring:
inlining, recursion

High-level data flow optimization:
loop invariants, operation strength

Partial evaluation:
calculate constants

Memory optimization:
loop trafos, data placement

Compiler Optimization Example

GCC Compiler Settings

- O1
- O2
- O3
- O0 (almost none $\rightarrow$ debugging)
- Os (code size)
- Ofast
- ...

Compiler Optimization Example

GCC Compiler Settings

- O1
- O2
- O3
- O0 (almost none $\rightarrow$ debugging)
- Os (code size)
- Ofast
- ...
- plus a ton of specific optimization settings (vectorize)

Compiler Optimization Example

GCC Compiler Setting O1

fauto-inc-dec	fbranch-count-reg	fcombine-stack-adjustments
fcompare-elim	fcprop-registers	**fdce**
fdefer-pop	fdelayed-branch	fdse
fforward-propagate	**fguess-branch-probability**	fif-conversion2
fif-conversion	**finline-functions-called-once**	fipa-pure-const
fipa-profile	fipa-reference	**fmerge-constants**
fmove-loop-invariants	freorder-blocks	fshrink-wrap
fsplit-wide-types	fssa-backprop	fssa-phiopt
ftree-bit-ccp	ftree-ccp	ftree-ch
ftree-coalesce-vars	ftree-copy-prop	ftree-dce
ftree-dominator-opts	ftree-dse	ftree-forwprop
ftree-fre	ftree-phiprop	ftree-sink
ftree-slsr	ftree-sra	ftree-pta
ftree-ter	funit-at-a-time	

Compiler Optimization Example

GCC Loop Unrolling

faggressive-loop-optimizations is always enabled.

GCC O2 Example: fgcse

Perform a **global** common subexpression elimination pass.
This pass also performs global constant and copy propagation.

GCC O3 Example: fpredictive-commoning

Perform predictive commoning optimization, i.e., reusing computations (especially memory loads and stores) performed in previous iterations of loops.