Program Performance: Code Optimization

Course Logistics
• Exam 2 graded...
• Projects 3,4 graded
• Weighted total updated

• Makeup Exam: comprehensive
  • Held during final exam time Dec.12th
  • Your final exam score will be the average of all three exams
    ➢ You can improve on 1/3rd of your score
• Project 6 will be due Dec. 12th – absolutely no extensions
  • This is equivalent to a take home final exam

Code optimization for performance
• A quick look at some techniques that can improve the performance of your code
• Rewrite code to minimize processor cycles
  • But do not mess up the correctness!
  • Reduce number of instructions executed
  • Reduce the “complexity” of instructions
    ➢ in real processors, different arithmetic operations can take different times
• Locality
  • Will improve memory performance

Recall CPU time model

\[
\text{CPU time} = \text{Seconds} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Program}} \times \frac{\text{Seconds}}{\text{Instruction}} \times \frac{\text{Cycle}}{\text{Clock}}
\]

\[
\text{CPU} = IC \times CPI \times CLK
\]
Summary: Memory Access time optimization

- If each access to memory leads to a cache hit then time to fetch from memory is one cycle
  - Program performance is good!
- If each access to memory leads to a cache miss then time to fetch from memory is much larger than 1 cycle
  - Program performance is bad!
- Design Goal:
  How to arrange data/instructions so that we have as few cache misses as possible.

Who can ‘change’ each parameter

- $CPU = IC \times CPI \times Clk$
- $Clk$: ?
- $IC$ (number of instructions): ?
- $CPI$: ?
- $Clk$: completely under HW control
- $IC$: programmer and compiler
- $CPI$: compiler and HW:
- ....so what does a compiler do?

Compiler Tasks

- 1. Code Translation
  - Source language → target language
    FORTRAN → C
    C → x86, MIPS, Alpha machine code etc.
    MIPS binary → Alpha binary
- 2. Code Optimization
  - Code runs faster
  - Match dynamic code behavior to static machine structure

Compiler Structure

Front End \rightarrow Optimizer \rightarrow Back End

(high-level source code) \rightarrow (machine code)

Machine independent \rightarrow Machine dependent

(Dependence Analyzer) (IR= intermediate representation)
Front End

- Lexical Analysis
  - Misspelling an identifier, keyword, or operator
e.g. lex
done by a finite state machine!

- Syntax Analysis
  - Grammar errors, such as mismatched parentheses
e.g. yacc

- Semantic Analysis
  - Type checking

Formal Model for Code Optimization?

- Is it a hack job or is there a formal model underlying the various transformations that can help with designing a tool to optimize code?
- Need to make sure that transformed code is correct and does not change semantics of the original program.

- Power of abstraction.....
  - Graph theory: model program as a graph (Program dependence graph)
    - Model data and control dependencies
    - Code transformation = graph transformation

The Program Dependence Graph

- How to represent control and data flow of a program?
- The Program Dependence Graph (PDG) is the intermediate (abstract) representation of a program designed for use in optimizations

- It consists of two important graphs:
  - Control Dependence Graph captures control flow and control dependence
  - Data Dependence Graph captures data dependences
  - Analogous to a flow-chart of the program
  - Formal model for flow charts!

Control Flow Graph: Definition

A control flow graph $CFG = (N_c; E_c; T_c)$ consists of

- $N_c$, a set of nodes. A node represents a straight-line sequence of operations with no intervening control flow i.e. a basic block.
- $E_c \subseteq N_c \times N_c \times \text{Labels}$, a set of labeled edges.
Control Flow Graph

main:
  addi r2, r0, A
  addi r3, r0, B
  addi r4, r0, C
  ...      bge  r10,r5, end
  addi r5, r0, N
  loop:
    lw   r20, 0(r2)
    lw   r21, 0(r3)
    lw   r21, 0(r3)
    ...    
    sw   r20, 0(r4)
    ...    
    bge  r20,r21,T1
    b    T2
    ...    
T1:
  addi r10,r10,1
  addi r2, r2, 4
  addi r3, r3, 4
  ...    
  addi r4, r4, 4
  ...    
  slt  r10,r5, loop
end:

Can this control flow happen?

Program behaviour?

• Model as program dependence graph!
• What is a correct execution?
  • Execution will only follow valid paths in the program dependence graph!
    ➢ If code is written correctly, then force the program to only follow paths in the dependence graph!
• Connection to Software security/correctness
  • Only execute along paths in the graph = program cannot execute any malicious code

Formal Definition/Model: Code Optimization

• Need to make sure that transformed code is correct and does not change semantics of the original program.

• Model program as a graph (Program dependence graph)
  • Model data and control dependencies

• Any transformation should give us a homomorphic graph
  • Recall concept of Isomorphism/Homomorphism Discrete Structures courses !!!
• Bad news: checking graph isomorphism is NP-complete!
  • Therefore ... ???
Compiler optimizations - Heuristics
• Use ‘heuristics’ to solve the difficult problem
• All ‘useful’ compilers have code optimizers built into them
  • Optimize time….
  • other metrics: power? Code size?
    ➢ Why?
• Machine dependent optimizations
  • Need to know something about the processor details before we can optimize
• Machine independent optimizations
  • These are independent of processor specifics

Machine Dependent Optimizations
These need some knowledge of the processor
• Register Allocation
• Instruction Scheduling
• Peephole Optimizations

Peephole Optimizations
• Replacements of assembly instruction through template matching
  • Need to know a lot about the instruction set
• Eg. Replacing one addressing mode with another in a CISC

Instruction Scheduling
• Given a source program P, schedule the instructions so as to minimize the overall execution time on the functional units in the target machine
  • This is where processors with parallelism introduce complexity into the scheduling process
  • Schedule parallel instructions
• Finding a schedule with minimum execution time is an NP-complete problem
  • Need fast and effective heuristics
  • You will cover schedulers in Operating Systems course
Register Allocation

• Storing and accessing variables from registers is much faster than accessing data from memory.
  • Variables ought to be stored in registers
• It is useful to store variables as long as possible, once they are loaded into registers
• Registers are bounded in number
  • "register-sharing" is needed over time.
  • Some variables have to be ‘flushed’ to memory – i.e., STR
  • Reading from memory takes longer – i.e., LDR

Register Allocation – Is it important?

• Efficient register allocators were seen to improve performance by 25%
  • Poor allocation means repeatedly reading variables from memory
• How do we solve/model the register allocation problem?...Power of abstraction!!
• Formulate the problem of assigning variables to registers as a graph problem
  • The Graph coloring problem!
    ➢ Number of colors = Number of registers; number of variables= nodes
  • Use application domain (Instruction execution) to define the priority function
• Graph theory & CS – it is every &@@ place!

Register Allocation

{ ... 
  i=10;
  x= y*z +i;
  while (i<100) {
    a = a*100
    b = b +100
    i++;
  }
}

• Suppose you have 3 registers available...
• Can you place x and a into same register?
• Should you place a and b into the same register?

Machine Dependent Optimizations

• Need thorough knowledge of the architecture AND algorithms
• New architectures introduce new challenges…a LOT to be done
  • Multi-core, Multi-threaded, Embedded (need to optimize for power consumption), Security (compiler-HW tools to enforce software security)
  • Machine dependent optimizations can be done by a compiler writer....
• Huge demand in industry....
  • But few CS students want to study this stuff 😓
• …and, this is not our focus for now!
Our focus: Machine Independent Optimizations

• As SW developers, these should be a ‘default’ when you write code…
  • THIS is what separates you from those who take a single programming course and claim they know CS!!
• How does it work: a large ‘menu’ of optimization techniques
  • Some dependent on general architecture
    ➢ Ex: Pipelined processors and loop unrolling
  • We cover a small sample that works on all processors

Some Machine-Independent Optimizations

• Some easy/obvious ones: Dataflow Analysis and Optimizations
  • Constant folding, Copy propagation etc.
  • Elimination of common subexpression
  • Dead code elimination
  • Code motion
  • Strength reduction
  • Function/Procedure inlining
  • Improving memory locality

Code-Optimizing Transformations

• Constant folding
  
  \[(1 + 2) \implies 3\]
  
  \[(100 > 0) \implies \text{true}\]

• Copy propagation

  \[
x = b + c 
  \implies \begin{align*}
x &= b + c \\
z &= y \cdot x 
  \implies z &= y \cdot (b + c)
\end{align*}
\]

Why does this make a difference:
Recall how code is generated..
(b+c) is stored into a temp register R0; x is a local var loaded from memory
So we replace (for 2nd statement):
LDR R0, R5, #-2 ; Load x into R0
LDR R1, R5, #-3 ; load y into R1
MUL R2, R0, R1 ; multiply x,y and store into R2
With
LDR R1, R5, #-3 ; load y into R1
MUL R2, R0,R1 ; multiply with value (b+c) stored in R0
This saves one memory access
Code-Optimizing Transformations

- Common subexpression – reduce instruction count
  \[ x = b \cdot c + 4 \quad t = b \cdot c \]
  \[ z = b \cdot c - 1 \quad x = t + 4 \]
  \[ z = t - 1 \]

  - 2 mult, 1 add, 1 sub replaced by
  - 1 mult, 1 add, 1 sub

Code-Optimizing Transformations

- Dead code elimination
  \[ x = 1 \]
  \[ x = b + c \]
  or if \( x \) is not referred to at all

Code Optimization Example

\[
\begin{align*}
  x &= 1 \\
  y &= a \cdot b + 3 \\
  z &= a \cdot b + x + z + 2 \\
  x &= 3 \\
  y &= a \cdot b + 3 \\
  z &= a \cdot b + 3 + z \\
  x &= 3 \\
  y &= a \cdot b + 3 \\
  z &= a \cdot b + 3 + z \\
  x &= 3 \\
  t &= a \cdot b + 3 \\
  y &= t \\
  z &= t + z \\
  x &= 3
\end{align*}
\]

Code Motion

- Code Motion
  - Reduce frequency with which computation performed
    ➢ If it will always produce same result
    ➢ Especially moving code out of loop
  - Move code between blocks
    ➢ eg. move loop invariant computations outside of loops
  - What does this reduce?

\[
\begin{align*}
  t &= x \div y \\
  \text{while } (i < 100) \{ \\
  \quad *p = x \div y + i \\
  \quad i = i + 1 \\
  \} \\
  \text{while } (i < 100) \{ \\
  \quad *p = t + i \\
  \quad i = i + 1 \\
  \}
\end{align*}
\]
• Code Motion:
  • Most compilers do a good job with array code + simple loop structures
  • Code Generated by GCC

```c
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[n*i + j] = b[j];
```

• Replace costly operation with simpler one
  • Shift, add instead of multiply or divide
    
    \[16\times x \rightarrow x \ll 4\]
    
    ➢ Utility is machine dependent
    ➢ Depends on cost of multiply or divide instruction
    ➢ On Pentium x86, integer multiply only requires 4 CPU cycles

• Recognize sequence of products

```c
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[n*i + j] = b[j];
```

Strength Reduction

• Replace complex (and costly) expressions with simpler ones
  • What does this reduce?
  • E.g.
    
    \[a := b \times 17\]
    
    \[a := (b \ll 4) + b\]
  • E.g.
    
    \[p = \&a[i]\]
    \[t = i \times 100\]
    
    while (\(i < 100\)) {
      \[a[i] = i \times 100\]
      \[i = i + 1\]
    }
    
    loop invariant: \&a[i] = p, \(i \times 100 = t\)

Function Inlining

• What happens on a function call?
  • How are function calls implemented on the machine?
  • Is function call = one subroutine call?
  • Function call in C = number of instructions in machine code
    • Create activation records, allocate memory
    • Manipulate stack and frame pointers
  • What happens if we replace function call with body of function?
    • Inline the function
Function Call/Return

• Instructions to Push arguments to stack
• Instructions to Push frame pointer, return addr. Execute instructions of function
• Instructions to Pop return value, reset frame pointer, pop return address
• The bookkeeping instructions are essentially an “overhead”
  • They do not do the work of the function
• What happens if we replace function call with body of function?
  • Inline the function

Function Inlining

int myfunc(int m,n)

x = myfunc(i,j) {
  return(m+n);
}

After inlining:

x = m+n

• Improves performance
  • Removes bookkeeping instructions
  • but tradeoff with code readability
  • and code size

Finally….Memory Locality & Code Performance

Locality

• Recall Principle of Locality:
  • Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  • Temporal locality: Recently referenced items are likely to be referenced in the near future.
  • Spatial locality: Items with nearby addresses tend to be referenced close together in time.
Link with Memory organization...

- Let's use array data structures to guide our discussions
- Recall: accesses to cache better than accesses to main memory/disk
- Recall: Multidimensional Arrays

How does a two dimensional array work?

How would you store it?

<table>
<thead>
<tr>
<th>Column Major Order</th>
<th>Row Major Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0   1, 0   2, 0   0, 1   1, 1   2, 1   0, 2   1, 2   2, 2   0, 3   1, 3   2, 3</td>
<td></td>
</tr>
<tr>
<td>Column 0</td>
<td>Column 1</td>
</tr>
<tr>
<td>0, 0   0, 1   0, 2   0, 3   1, 0   1, 1   1, 2   1, 3   2, 0   2, 1   2, 2   2, 3</td>
<td></td>
</tr>
<tr>
<td>Row 0</td>
<td>Row 1</td>
</tr>
</tbody>
</table>
How would you store it?

C stores in row major order

<table>
<thead>
<tr>
<th>Column Major Order</th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,1</td>
<td>0,2</td>
<td>0,3</td>
<td></td>
</tr>
<tr>
<td>1,0</td>
<td>1,1</td>
<td>1,2</td>
<td>1,3</td>
<td></td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
<td>2,3</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Major Order</th>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,1</td>
<td>0,2</td>
<td>0,3</td>
</tr>
<tr>
<td>1,0</td>
<td>1,1</td>
<td>1,2</td>
<td>1,3</td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
<td>2,3</td>
</tr>
</tbody>
</table>

Locality of Access

• How are elements in the array accessed in your program?
  • Row major
  • How would you iterate over the 2-D array to maintain locality?

Locality and performance

• Recall: Memory = Cache + Main memory
  • Cache contains small number of bytes
• Recall: cache is arranged as a set of blocks
  • Can only fetch block at a time
• Example:
  • Assume each cache block has 4 words
  • If you fetch a block with addresses (0,1,2,3)
    • If four successive instructions use locations 0,1,2,3 then we only have one cache miss (first time to fetch block into cache)
    • If four successive instructions use locations 0,4,8,12 then each time we have to fetch a new cache block
  • Goal: have locality in memory accesses

Example

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1,2,3)</td>
</tr>
<tr>
<td>(4,5,6,7)</td>
</tr>
<tr>
<td>(8,9,10,11)</td>
</tr>
<tr>
<td>(12,13,14,15)</td>
</tr>
<tr>
<td>(16,17,18,19)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,2,3</td>
</tr>
<tr>
<td>4,5,6,7</td>
</tr>
<tr>
<td>8,9,10,11</td>
</tr>
<tr>
<td>12,13,14,15</td>
</tr>
</tbody>
</table>

4 bytes in each block
Addresses: (0,1,2,3)
Locality

- Being able to look at code and get a qualitative sense of its locality is a key skill for a professional software developer.

Locality Example

Question: Does this function have good locality?

```c
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum
}
```

```
Access pattern 1
```

Improving Memory Access Times (Cache Performance) by Compiler Optimizations

- McFarling [1989] improve perf. By rewriting the software
- Instructions
  - Reorder procedures in memory so as to reduce cache misses
  - Code Profiling to look at cache misses (using tools they developed)
- Data
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop Interchange: change nesting of loops to access data in order stored in memory
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Compiler optimizations – merging arrays

- This works by improving spatial locality
- For example, some programs may reference multiple arrays of the same size at the same time
  - Could be bad – not enough locality
    - Accesses may interfere with one another in the cache – conflict misses
- A solution: Generate a single, compound array...

```c
/* Before:*/
int tag[SIZE]
int byte1[SIZE]
int byte2[SIZE]
int dirty[size]
/* After */
struct merge {
    int tag;
    int byte1;
    int byte2;
    int dirty;
}  
struct merge cache_block_entry[SIZE]
```

Merging Arrays Example

/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of structures */
struct merge {
    int val;
    int key;
}  
struct merge merged_array[SIZE];

Reducing conflicts between val & key; improve spatial locality

Compiler optimizations – loop interchange

- Some programs have nested loops that access memory in non-sequential order
  - Simply changing the order of the loops may make them access the data in sequential order...
- What’s an example of this?
  - Recall: C stores 2-D arrays in row-major format

```c
/* Before */
for (k = 0; k < 100; k = k+1)
for (j = 0; j < 100; j = j+1)
for (i = 0; i < 5000; i = i+1)
x[i][j] = 2 * x[i][j];
```

Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
for (j = 0; j < 100; j = j+1)
for (i = 0; i < 5000; i = i+1)
x[i][j] = 2 * x[i][j];
Loop Interchange Example

/* After */
for (k = 0; k < 100; k = k+1)
    for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
            x[i][j] = 2 * x[i][j];

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Compiler optimizations – loop fusion

• This one’s pretty obvious once you hear what it is…
• Seeks to take advantage of:
  • Programs that have separate sections of code that access the same arrays in different loops
  • Especially when the loops use common data
  • The idea is to “fuse” the loops into one common loop
• What’s the target of this optimization?

Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];

2 misses per access to a & c vs. one miss per access; improve spatial locality & temporal locality
A more general concept: Memory Blocking.
• Can you keep locality in all memory operations
• This is probably the most “famous” of compiler optimizations to improve cache performance
• Another common concept: blocking
  • Rewrite code to process blocks of data at a time
  • Size of block = ??? Size of cache block!!

Compiler optimizations – blocking
• Tries to reduce misses by improving temporal locality and spatial locality
• To get a handle on this, you have to work through code on your own
• this is used mainly with arrays!
• Simplest case??
  • Row-major access

Naïve Matrix Multiply
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}
  for j = 1 to n
    {read C(i,j) into fast memory}
    {read column j of B into fast memory}
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    {write C(i,j) back to slow memory}

Blocked (Tiled) Matrix Multiply
Consider A,B,C to be N-by-N matrices of b-by-b subblocks where b=n / N is called the block size
for i = 1 to N
  for j = 1 to N
    {read block C(i,j) into fast memory}
    for k = 1 to N
      {read block A(i,k) into fast memory}
      {read block B(k,j) into fast memory}
      C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory}

Work these details out….need it for the project!
**Code Optimization and Compilers**

- Modern compilers provide a menu of code optimization features
  - Inlining, strength reduction, register allocation, loop optimizations, etc.
- Some provide default optimization levels
  - Example: gcc -O3 test.c
- Bottom Line: Everyone wants to run optimized code
  - Being smart with your solution!

---

**Example of Code Optimization: (Final) Project 6**

- Topic: Code Performance Optimization
  - Given code for Image Smoothing and Image Rotation, rewrite the code to make it run faster.
    - Use only techniques covered in class.
- Description will be posted Thursday and code be posted Friday 2pm
- Due Dec. 12th, 2 pm. (official final exam date)
  - Should take you 6-10 hours to complete
- Involves:
  - Code rewriting
  - Report writing: summarize your experiments, explain why the code ran faster (or slower).
  - Grade will depend on your analysis – simply turning in code (with documentation) that runs faster will only get you max of 50%