Code Optimization

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
    o In real processors, different arithmetic operations can take different times
• Locality
  • Will improve memory performance
Recall CPU time model

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

\[
CPU = IC \times CPI \times Clk
\]

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:
  
  \textit{How to arrange data/instructions so that we have as few cache misses as possible.}
Who can ‘change’ each parameter

- CPU time = IC * CPI * Clk
- Clock: 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 → MIPS, x86, PowerPC or Alpha machine code
    MIPS binary → x86 binary

- 2. Code Optimization
  - Code runs faster
  - Match dynamic code behavior to static machine structure
Compiler Structure

- **Front End**
  - high-level source code
  - Machine independent
- **Optimizer**
  - IR
- **Back End**
  - machine code
  - Machine dependent

(IR= intermediate representation)

Compiler Front End tasks

- **Lexical Analysis**
  - Misspelling an identifier, keyword, or operator
    - e.g. lex
    - done by a finite state machine (i.e., *deterministic finite automata*)!

- **Syntax Analysis**
  - Grammar errors, such as mismatched parentheses
  - Define syntax using *Context Free Grammar*...then build parser
    - e.g. yacc

- **Semantic Analysis**
  - Type checking, check formal and actual arguments to function match, etc.

- code generation...you’ve been doing this for C to LC3!!
  - to target ISA or intermediate code, llvm-code
**Code Optimization**

- After front end analysis $\Rightarrow$ an executable program $P$

- $P$ has some performance $T(P)$
  - $T(P) = IC \times CPI \times Clock$

- Goal: Improve $T(P)$
  - Reduce time
  - How? Reduce CPI and/or IC

- Rewrite/transform $P$ to equivalent program $Q$ such that
  1. $T(Q) < T(P)$ and
  2. $Q$ and $P$ are equivalent, i.e., do exactly the same thing
     - For all inputs, $Q$ and $P$ produce the same result and compute the same function

---

**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!

Definition: Control Flow Graph

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.
• \( E_c \subseteq N_c \times N_c \times Labels \), a set of labeled edges.

• Example: the code below has two basic blocks

```
ADD R0, R0, #0
BRn here1
LDR R1, R0, #0
ADD R2, R1, R2
BRzp here2
```
Data Dependence Graph

- Within each basic block capture the data dependencies between instructions
  - In RISC processor, follow the data in the registers
  - Value computed in a register is needed by an instruction in the future
  - Ex:
    - Value computed by LDR is needed by next instruction
    - But no dependence between AND and the other instructions
- Can capture these dependencies using a graph
  - Nodes are instructions and edges dependencies
- Data dependencies important in
  - Scheduling instructions
  - Parallelizing the code

```
LDR R1, R0, #0
ADD R2, R1, R2
AND R3, R3, #4
BRzp here2
```

Control Flow Graph

```
main:
addi r2, r0, A
addi r3, r0, B
addi r4, r0, C
addi r5, r0, N
add r10,r0, r0
bge r10,r5, end

loop:
lw r20, 0(r2) BB 1
lw r21, 0(r3)
bge r20,r21,T1 BB 2
sw r21, 0(r4) BB 3
b T2

T1: sw r20, 0(r4) BB 4
T2: addi r10,r10,1
addi r2, r2, 4
addi r3, r3, 4 BB 5
addi r4, r4, 4
blt r10,r5, loop

end:
```
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

• 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

• 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
  • Reading from memory takes longer
• how important is Register allocation to performance?
  • efficient register allocators improved performance 25%
  • Poor allocation means repeatedly reading variables from memory

Register Allocation

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

• Suppose you have 3 registers available…
• should you place a and b into same register ?
• Can you place x and a into same register ?
Register Allocation

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

• “live range” \( LR(j) \) for each variable \( j \) – where is it accessed
• Do live ranges of \( x \) and a “interfere” : \( LR(x) \) & \( LR(a) = 0 \)?
• Do live ranges of \( a \) and \( b \) interfere ? : \( LR(a) \) & \( LR(b) = 0 \)?
• If ranges interfere, then assign to different registers

Register Allocation: Problem Formulation and Solution

• Determine live ranges for each variable, and determine conflicts/interference between variables/live ranges
  • Using dataflow analysis compute live ranges for each variable
• How do we model the register allocation problem?
  • Power of abstraction!!
• Formulate the problem of assigning variables to registers as a graph problem: The Graph coloring problem !
  o Number of colors = Number of registers;
  o Nodes in graph = number of variables (live ranges)
  o Edges in graph = edge between \( x,y \) if live ranges \( x,y \) interfere
• Use application domain (Instruction execution) to define the priority function
• Graph theory & CS – it is every &@#$@ place!
  • My curriculum advice (that nobody takes…except 2): take a graph theory course!
Machine Dependent Optimizations

- Need thorough knowledge of the architecture AND algorithms
- New architectures introduce new challenges…
  - Multi-core, Multi-threaded, Embedded (need to optimize for power consumption), Security (compiler-HW tools to enforce software security)
  - Compiling for FPGA co-processors to accelerate (ex: AWS, Microsoft)
- Compiling for Security – leverage FPGAs & extra HW to place verification and encryption circuits
- Compiling for power optimization
  - Control memory power using compiler….layout the data so we can switch off memory modules
- 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
  
  \[
  \begin{aligned}
  (1 + 2) & \Rightarrow 3 \\
  (100 > 0) & \Rightarrow \text{true}
  \end{aligned}
  \]

  This save one instruction – reduce IC
Code-Optimizing Transformations

• Copy propagation
  \[ x = b + c \quad \Rightarrow \quad x = b + c \]
  \[ z = y \cdot x \quad \Rightarrow \quad z = y \cdot (b + c) \]

Why does this make a difference: Recall how code is generated.
(b+c) is stored into a temp register R0 and then STR R0, R5, #-2 to store
local var x. Code generated for the 2nd statement \( z = y \cdot x \) is:

```
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
```

Replace above 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..reduces IC and CPI*

Code-Optimizing Transformations

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

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

• Reduces IC
Code-Optimizing Transformations

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

  Saves one instruction…reduce IC

Code Optimization Example

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

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

Original: 2 Mult, 4 Add, 7 Read/Write Mem

New: 1 Mult, 2 Add, 5 Read/Write
**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?
    - Number of times x/y is computed...reduce IC

\[
t = \frac{x}{y}
\]

```c
while (i < 100) {
  \*p = \frac{x}{y} + i
  i = i + 1
}
```

- **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];
```

```c
for (i = 0; i < n; i++)
{
  int ni = n*i;
  int \*p = a+ni;
  for (j = 0; j < n; j++)
    \*p++ = b[j];
}
```
Strength Reduction

- 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

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

```markdown
int ni = 0;
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++)
    a[ni + j] = b[j];
  ni = ni + n;
}
```

Strength Reduction

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

loop invariant: &a[i]=p, i*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
  • i.e., 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
  • Remove the function call and return overhead instructions
  • …reduce IC
Function Inlining

```c
int myfunc(int m,n)
{
    return(m+n);
}
```

After inlining:

```c
...
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
**Declaration**

```c
int ia[3][4];
```

**Number of Rows**

**Number of Columns**

**Type**

**Address**

*Declaration at compile time i.e. size must be known*

---

**How does a two dimensional array work?**

```
0 1 2 3
0
1
2
```

*How would you store it?*
### How would you store it?

#### Column Major Order

<table>
<thead>
<tr>
<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>1,0</td>
<td>2,0</td>
<td>0,1</td>
</tr>
<tr>
<td>0,1</td>
<td>1,1</td>
<td>2,1</td>
<td>0,2</td>
</tr>
<tr>
<td>0,2</td>
<td>1,2</td>
<td>2,2</td>
<td>0,3</td>
</tr>
<tr>
<td>0,3</td>
<td>1,3</td>
<td>2,3</td>
<td></td>
</tr>
</tbody>
</table>

#### Row Major Order

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1,1</td>
<td>1,2</td>
<td>1,3</td>
</tr>
<tr>
<td>1,2</td>
<td>1,3</td>
<td>2,0</td>
</tr>
<tr>
<td>1,3</td>
<td>2,0</td>
<td>2,1</td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
</tr>
<tr>
<td>2,1</td>
<td>2,2</td>
<td>2,3</td>
</tr>
</tbody>
</table>

---

### C stores in row major order

#### Column Major Order

<table>
<thead>
<tr>
<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>1,0</td>
<td>2,0</td>
<td>0,1</td>
</tr>
<tr>
<td>0,1</td>
<td>1,1</td>
<td>2,1</td>
<td>0,2</td>
</tr>
<tr>
<td>0,2</td>
<td>1,2</td>
<td>2,2</td>
<td>0,3</td>
</tr>
<tr>
<td>0,3</td>
<td>1,3</td>
<td>2,3</td>
<td></td>
</tr>
</tbody>
</table>

#### Row Major Order

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1,1</td>
<td>1,2</td>
<td>1,3</td>
</tr>
<tr>
<td>1,2</td>
<td>1,3</td>
<td>2,0</td>
</tr>
<tr>
<td>1,3</td>
<td>2,0</td>
<td>2,1</td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
</tr>
<tr>
<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 or column major or other?
  - 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
  - Each memory access is an access to main memory

- Goal: have locality in memory accesses
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: Column-major

```plaintext
0 1 2 3
0 1 2
```

**Locality Example**

- **Question:** Does this function have good locality?

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

Access pattern: Row-major

```plaintext
0 1 2 3
0 1
0 1 2
```
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
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
    o 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?
  • Locality – reduce memory access times
  • IC – by reducing number of branches
    o Important in pipelined processors

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];

Move inside first loop
Loop Fusion Example

/* After */
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];
        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

```plaintext
{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; note column major access!}
    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}
```

Good locality in access to matrix A; poor locality in access to B
Blocked (Tiled) Matrix Multiply

Consider A,B,C to be N-by-N matrices of b-by-b subblocks where
b= N/m is called the block size
for i = 1 to N
    for j = 1 to N
        {read block of C(i,j) into fast memory}
        for k = 1 to N
            {read block of A(i,k) into fast memory}
            {read block of 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 -03 test.c

• Bottom Line: Everyone wants to run optimized code
  • Being smart with your solution!
• Have we seen everything there is to code optimization?....not by a long shot !!
  • Lots and lots more optimization techniques
    • The “cooler” ones need architecture knowledge
Example of Code Optimization:
(Final) Project 6

• Topic: Code Performance Optimization
  • Given code for Image operations, rewrite the code to make it run faster.
    o Use only techniques covered in class.

• Description will be posted last day of classes and due official final exam date: Thursday Dec. 17\textsuperscript{th} midnight.
  Should take you 6-10 hours to complete

• Involves:
  • Code rewriting
  • Report writing: summarize your experiments, explain why the code ran faster (or slower).

• Very Important: Grade will depend on your analysis – simply turning in code (with documentation) that runs faster will only get you up to 50%