Benchmarks and Optimizing Compilers

CS 471/Math 471
Brian T. Smith, UNM
Andrew C. Pineda, UNM

Benchmarks

- Reference:
    - Chapter 11.3, IHPSC
    - http://www.netlib.org
    - http://www.nas.nasa.gov/RNR/Parallel/NPB
- Used to compare machines:
    - sometimes application related (numerical fp)
    - results published on netlib and other web sites
    - code often available to rerun the tests

Types Of Benchmarks

- CPU performance
- File server performance
- input/output interface performance
    - I/O libraries
- Network performance (control and data)
- Data communication (parallel computation)
- Others ...

Nature of Benchmarks

- Kernels
    - taken from from well-known and accepted applications (representative applications)
        - e.g. Livermore Loops
- Synthetic Tests
    - code designed to test performance of statements and instructions selected on a statistical passes
        - e.g. SPECmarks, Whetstones, Dhrystones, ...

Nature Of Benchmarks Continued

- Applications
    - LINPACK benchmarks
        - solve a system of linear equations of order 100
    - SPICE (Berkeley)
        - computations involving circuit designs
    - Shallow Water Simulation (NCAR)
    - NAS Parallel Benchmarks (NASA Ames)

Some Details Of Benchmarks

- Livermore loops (LLNL)
    - 24 kernels taken from application codes, for example
        - dot product, Monte Carlo, Cholesky factorization, fragment of a 2-D hydrodynamics solver, …
- Whetstones (NPL, UK)
    - selection of language constructs from numerical intensive computations (statistically sampled)
    - high level language constructs (Algol, Fortran)

Some More Details

- Dhrystones
  - benchmarks for system and integer operations
  - synthetic collections of tests based on a frequency analysis of programs such as compiler and system operations
  - written for C, Ada, and Pascal
- MIPS (millions of instructions per second)
  - comparisons with DEC 11/780 -- classic 1 MIP machine -- cache performance skews results

More Details

- SPECmarks
  - System Performance Evaluation Cooperative
  - a consortium of manufacturers (originally DEC, Sun, HP, MIPS (the vendor))
  - benchmarks are:
    - SPECintxx, SPECfpxx (xx is a year, 92, 95, …) to test integer and floating point performance
    - SDM (Unix software development workload)
    - SFS (Systems-level file server workload)

Compiler Optimizations

- References:
  - Chapter 11.4 of IHPSC
  - Optimization and Tuning Guide for Fortran, C, and C++, AIX Version 3.2 for RISC System/6000
- Essence:
  - minimize unnecessary instructions in machine code
  - rearrange the machine instructions to optimize the use of cache, pipelines, and code lookahead hardware

High Level View Of Compiler Optimization

- Rearrange the user code to yield a mathematically equivalent program that runs faster
  - equivalence in the sense of what is visible from outside the program
  - equivalence is defined by what is allowed by the definition of the language

General Kinds of Compiler Optimizations

- Common sub-expression elimination
  - do it once only
- Statement reordering and elimination
- Strength reduction
  - reduce the number of operations or kind of operations with the same mathematical results
- Loop optimization
- Array optimization and memory management

An Example

- Code taken from advection simulation

```
do j = 0, n-1
    do i = 0, n-1
      u(i,j) = 0.5d0*sin(2.0d0*pi*(i+j)*h)
      v(i,j) = -0.5d0*sin(2.0d0*pi*(i+j)*h)
      phi(I,j,b1) = exp(-16*h*h*((i-n/2.0)**2 + &
                        (j-n/2.0d0)**2))*sin(e*i*h)
    enddo
enddo
```

Common Sub-Expressions

- The expression `0.5d0*sin(2.0d0*pi*(i+j)*h)` appears twice
  - the compiler assigns it to a temporary variable
  - saves 4n*n multiplies, n*n additions, n*n sine evaluations

```
do j = 0, n-1
   do i = 0, n-1
     t1 = 0.5d0*sin(2.0d0*pi*(i+j)*h)
     u(i,j) = t1
     v(i,j) = -t1
     phi(I,j,b1) = exp(-16*h*h*((i-n/2.0)**2 + &
                          (j-n/2.0d0)**2))*sin(e*i*h)
   enddo
enddo
```


Common Expression and Code Movement

- Notice that:
  - common sub-expressions such as N/2.0, and
  - expressions (such as the above) that do not depend on the loop iterations
    - 2.0*pi*h
    - -16.0*h*h
    - e*h
- Move these expressions outside all DO loops (needs to recognize that * commutes)

Equivalent Optimized Code

```
t2 = 2.0d0*pi*h
t3 = -16*h*h
t4 = N/2.0d0; t5 = e*h
do j = 0, n-1
   t6 = (j-t4)**2
  do i = 0, n-1
     t1 = 0.5d0*sin(t2*(i+j))
     u(i,j) = t1; v(i,j) = -t2
     phi(i,j,b1) = exp(t3*((i-t4)**2 + t6))*sin(t5*h)
   enddo
enddo
```


Other Optimizations Possible

- This code has common sub-expressions and code movement optimizations that are possible
  - compute the vector (i-t4, i=0, n-1) once
  - compute the vector (i*t5, I=0, n-1) once
- Unlikely to be detected by the compiler
- Need to know what the compiler is likely to optimize and how to find out what the compiler has optimized
- BUT only for the inner (time-intensive) loops

Other Kinds Of Optimizations By The XLF Compiler

- Strength reduction
  - integer multiplies replaced by shifts and adds
  - exponentiation replaced by multiplies
  - use of Horner's scheme
  - floating point division replaced by reciprocal
    - especially for constants
    - division is 16-19 cycles
  - division rotation (division by multiplies)
  - factorization (x*a+x*b computed as x*(a+b))

Other Optimizations Continued

- Loop optimizations
  - loop unrolling to reduce loop overhead
  - loop simplification (code moved outside loop)
  - remove unneeded loop code
  - loop fusion (reduce loop overhead maybe)
  - loop distribution (better cache behavior)
  - loop interchange
  - removing IF code from the loop

Other Compiler Optimizations

– Array optimization and memory management
  - stride and TLB (next page) miss minimization
    – 6.7% of stride one performance in extreme case
    – cache miss is 8 cycles
    – table lookahead buffer (TLB) miss is 36 cycles
    – page fault is ???? (depends on disk speed)
  - avoiding misses
  - loop skewing and loop reversal
  - blocking and strip mining
  - triangular array manipulations

Page Faults and Large Strides

- One TLB miss per inner loop iteration

```
real(8)  a(513,512)
do i = 1, 512
   do j = 1, 512
      a(i,j) = a(i,j)*x
   enddo
enddo
```

- One TLB miss per outer loop iteration
  – switch the order of the loops