Algorithms and Computation in Signal Processing

special topic course 18-799B
spring 2005
5^{th} Lecture Jan. 25, 2005

Instructor: Markus Pueschel

TA: Srinivas Chellappa

Guide to Benchmarking

Guide to Benchmarking: How?

- **First: Verify your code!**

- **Measure runtime, compare against the best available code**
  - compile other code correctly (as good as possible)
  - use same timing method
  - be fair
  - always sanity check: compare to published results etc.

- **Measure performance: flops (number floating point ops/second), compare to peak performance**
  - needs peak performance
  - get instruction count statically (cost analysis) or dynamically (tool that counts, or replace ops by counters through macros)
  - Careful: Different algorithms may have different op count, i.e., best flops is not always best runtime

Guide to benchmarking: How to measure runtime?

- **C clock()**
  - process specific, low resolution, very portable
- **gettimeofday**
  - measures wall clock time, higher resolution, somewhat portable
- **Performance counter (e.g., TSC on Pentiums)**
  - measures cycles (i.e., also wall clock time), highest resolution, not portable
- **Careful:**
  - measure only what you want to measure (maybe subtract overhead)
  - proper machine state (e.g., cold/warm cache)
  - measure enough repetitions
  - check how reproducible; if not reproducible: fix it

Guide to Benchmarking: How to present results (in writing)?

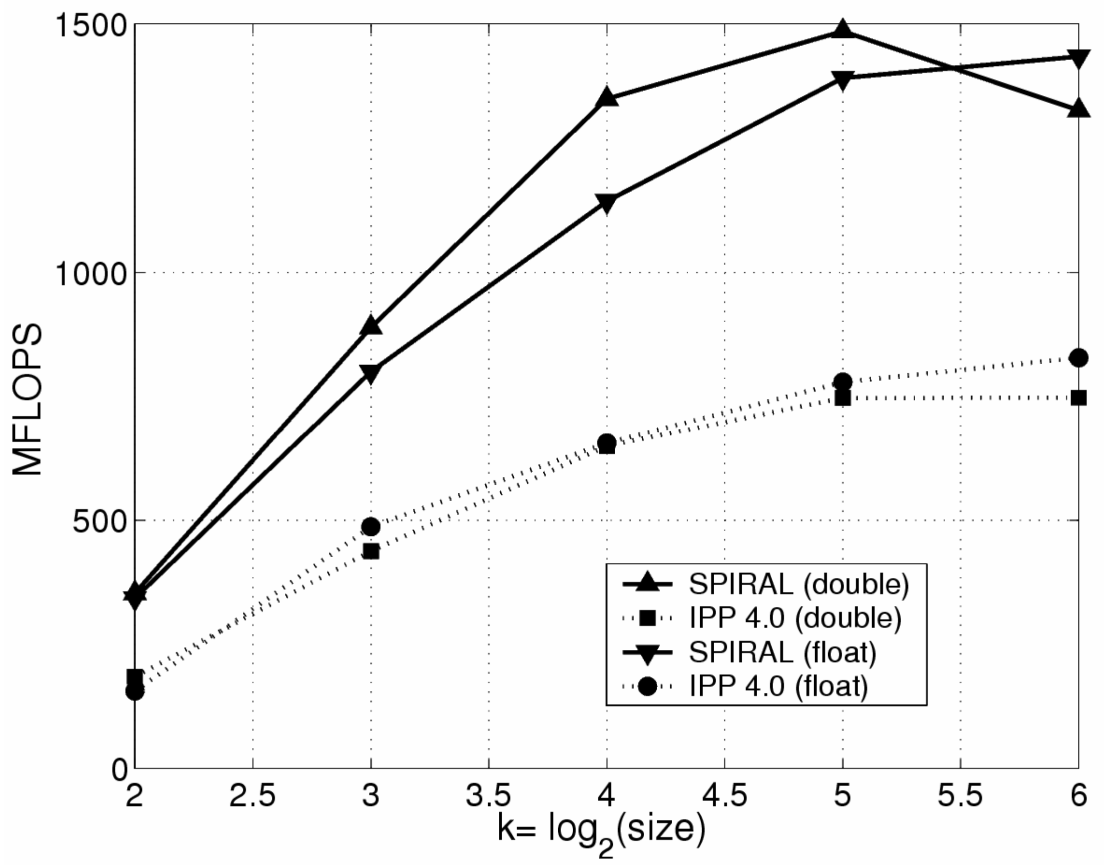
- **Specify machine**
  - processor type, frequency
  - relevant caches and their sizes
  - operating system
- **Specify compilation**
  - compiler incl. version
  - flags
- **Explain timing method**
- **Plot**
  - Has to be very readable (colors, lines, fonts, etc.)
  - Choose proper type of plot: message as visible as possible

Guide to Benchmarking: How to present results (talking)?

- Briefly explain the experiment
- Explain x- and y-axis
- Say, e.g., “higher is better” if appropriate
- If many lines, maybe explain one as example
- Extract a message in the end

Example

Performance of code for the discrete cosine transform (DCT):



Platform:
P4 (HT), 3GHz,
8KB L1, 512KB L2,
WinXP

Compiler:
icc 8.0

Compiler flags:
/QxKW /G7 /O3

Spiral-generated code is a factor of 2 faster
reaches up to 50% of the peak performance

Linear Algebra Software: LAPACK and BLAS

Linear Algebra Algorithms: Examples

- Solving systems of linear equations
- Computation of eigenvalues
- Singular value decomposition
- LU/Cholesky/QR/… decompositions
- … and many others

- Make up most of the numerical computation across disciplines (sciences, computer science, engineering)
- Efficient software is extremely relevant

The Path to LAPACK

- **1960s/70s: EISPACK and LINPACK**
  - libraries for linear algebra algorithms
  - Cleve Moler et al.
- **Problem:**
  - Implementation “vector-based,” i.e., no locality in data access
  - Low performance on computers with deep memory hierarchy
  - Became apparent in the 80s
- **Solution: LAPACK**
  - Reimplement the algorithms “block-based,” i.e., with locality
  - End of 1980s, early 1990s
  - Jim Demmel, Jack Dongarra et al.

LAPACK and BLAS

- **Basic Idea:**

- **BLAS = Basic Linear Algebra Subroutines** *link*
  - BLAS1: vector-vector operations (e.g., vector sum)
  - BLAS2: matrix-vector operations (e.g., matrix-vector product)
  - BLAS3: matrix-matrix operations (mainly matrix-matrix product)

- **LAPACK implemented on top of BLAS** *link*
  - as much as possible using block matrix operations (locality) = BLAS 3
  - Implemented in F77 (enables good compilation)
  - Open source

- **BLAS recreated for each platform to port performance**

Why is BLAS3 so important?

- BLAS1: O(n) data, O(n) operations
- BLAS2: $O(n^2)$ data, $O(n^2)$ operations
- BLAS3: $O(n^2)$ data, $O(n^3)$ operations = data reuse = locality!

- Give example of blocking for MMM (blackboard)

Blocking (for the memory hierarchy) is the single most important optimization for linear algebra algorithms

Matrix-Matrix Multiplication (MMM): Algorithms and Complexity

MMM by Definition

- **Cost as computed before**
  - n^3 multiplications
  - n^3-n^2 additions
  - $= 2n^3-n^2$ floating point operations
  - $=O(n^3)$ runtime
- **Blocking**
  - Increases locality (see previous example)
  - Does not decrease cost
- **Can we do better?**

Strassen's Algorithm

- **Strassen, V. "Gaussian Elimination is Not Optimal."** *Numerische Mathematik* **13, 354-356, 1969**
- **Multiplies two n x n matrices in $O(n^{\log_2(7)}) \approx O(n^{2.808})$**
- **Similarities to Karatsuba**
- **Check out algorithm at Mathworld** link
- **Breakover point, in terms of cost: n=654, but …**
  - Structure more complex
  - Numerical stability inferior
- **Can we do better?**

MMM Complexity: What is known

- Coppersmith, D. and Winograd, S. "Matrix Multiplication via Arithmetic Programming." *J. Symb. Comput.* 9, 251-280, 1990
- MMM is $O(n^{2.376})$ and $\Omega(n^2)$
- It could well be $\Theta(n^2)$
- Compare this to matrix-vector multiplication, which is $\Theta(n^2)$ (Winograd), i.e., boring
- MMM is the single most important computational kernel in linear algebra (probably in whole numerical computing)