Course Evaluation

- Email sent out today by Suzie Laurich-McIntyre
- Please fill out (is anonymous)
Midterm

What to learn:
- Understand $O$, $\Omega$, $\Theta$
- DFT properties explained in class
- Know the most important complexities
- Solve a recurrence using generating functions
- Be able to analyze the cost of a recursive transform algorithm given in terms of tensor products etc.
Feedback for 2nd Assignment
Plotting Graphs

- Almost always include data points on a line graph made up of discrete data
  - Without data points, a dip in the graph could have been because of one single deviated value, or because of multiple values
  - Curve of the line is arbitrarily decided by the plotting program

- Ensure that a line graph begins at an appropriate value (and not zero, unless that is an actual data value)
Plotting Graphs

Discuss and analyze:

- A plot by itself is usually of little value: What really matters is a meaningful discussion and analysis of the plot. Eg:
  - Why is a curve on the plot shaped a certain way?
  - What factors (apparent or hidden) influence or could potentially influence the plot
  - How would the extrapolated graph look (esp. important for MFLOPS plots)
  - What significance do the global maxima and minima have?
- At the very least, discuss and speculate
Plotting Graphs

- Use the correct kind of graph to illustrate your data:
  - Trends: line graph
  - Bars: values across categories
  - Pie charts: Contributions to a total value

- MFLOPS for this assignment: do not use bar graphs!

- Similarly, use a table when appropriate
Presenting data

- If there is a significant amount of variance in your data, either execute adequate iterations to get a meaningful mean, and/or choose to also present a measure of variance like standard deviation
Experiments

- If conducting a new experiment, (or deviating from the question in any manner):
  - First, clearly and explicitly present the objective or hypothesis
  - Next, present the experiment and how it verifies the hypothesis
Measuring time

- Understand the difference between wall clock time, user time, system time etc. This is important!
FLOPS and Peak performance calculation

- MFLOPS: Includes only FP +, *
- Does not include loads/stores (or you have to adjust peak performance)

- Peak performance: Do not assume this is the same as the clock frequency. This value depends on the computer and needs to be found out.
Some Plots from the 2nd Assignment
- CPU: PowerPC 750 ("G3") / 400 MHz / 32K L1-I,D caches / 1MB of L2 cache
- c2swap = i,j loops swapped
- CPU: Pentium M 2GHz / 1Gb / gcc 3.3.49
PowerBook G4 / Freescale PowerPC MPC7447A CPU at 1.5 GHz
Code3 reverse: loop order from ijk to jik
Pentium M / 1600 MHz / 32k L1 D,I caches / 1MB L2 cache
Pentium M / 1600 MHz / 32k L1 D,I caches / 1MB L2 cache
Pentium 4 / ICC

- Code-00: Triple loop naïve implementation
- Code-01: Block for Register
- Code-02: Block for Register Unroll 2
- Code-03: Block for Register Unroll 4
- Code-04: Block for Register + SSE
- Code-05: Block for Register + Block for L1 Cache
- Code-06: Block for Register + Block for L1 Cache + SSE
- **PowerBook G4 / Freescale PowerPC MPC7447A CPU at 1.5 GHz**
- **Code3 reverse**: code3+jik loop order
FFT Summary
FFT Algorithm Summary

- There is not just one FFT (Cooley-Tukey, Rader, etc.)

- Even if only Cooley-Tukey FFT is considered there are many ways of recursing (similar cost, but different dataflow)

- Several complexity results for the DFT are available. If \( c \) is bounded, then \( L_c(DFT_n) = \Theta(n \log(n)) \)
The FFT codelet generator in FFTW

M. Frigo, “A Fast Fourier Transform Compiler,”
Proc. PLDI 1999  link

FFTW homepage  link
Basic Block Optimizations for FFTs

- **Problem**: as in MMM, we do not want to recurse all the way down. Infrastructure destroys performance.
- **Solution**: Unrolled code for small size (<= 64)
- Optimization for these blocks is much harder than the micro/mini MMMs in MMM
- Again, compilers don’t do a good job on unrolled code
- **Solution**: Code generator/optimizer for small sizes

n → FFT codelet generator → Codelet for DFT<sub>n</sub>
Twiddle codelet for DFT<sub>n</sub>
Codelet Generator: Details

- DAG: directed acyclic graph
  - Represents a DFT algorithm (the dataflow)
  - Nodes: load, store, adds, mults by constant

- Give example on blackboard
DAG Generator

- Knows FFTs: Cooley-Tukey, split-radix, Good-Thomas, Rader, represented in sum notation

\[ y_{n_2 j_1 + j_2} = \sum_{k_1=0}^{n_1-1} (\omega_n^{j_2 k_1}) \left( \sum_{k_2=0}^{n_2-1} x_{n_1 k_2 + k_1} \omega_{n_2}^{j_2 k_2} \right) \omega_{n_1}^{j_1 k_1} \]

- For given n, suitable FFTs are recursively applied to yield n (real) expression trees for \( y_0, \ldots, y_{n-1} \)

- Trees are fused to an (unoptimized) DAG
Simplifier

- Applies: algebraic transformations, common subexpression elimination (CSE), DFT-specific optimizations

- **Algebraic transformations**
  - Simplify mults by 0, 1, -1
  - Distributivity law: \(kx + ky = k(x + y), kx + lx = (k + l)x\)
    
    May destroy common subexpressions and thus increase op count!
  - Canonicalization: \((x-y), (y-x)\) to \((x-y), -(x-y)\)

- **CSE: standard**
  - E.g., two occurrences of \(2x+y\): assign new temporary variable
Simplifier (cont’d)

- **DFT-specific optimizations**
  - All numeric constants are made positive
  - Reason: constants need to be loaded into registers, too
  - CSE on the transposed DAG (Blackboard)
Scheduler

- Determines in which sequence the DAG is unparsed to C (topological sort of the DAG)
  Goal: minimizer register spills

- If C register are available, then a 2-power FFT needs at least $\Omega(n\log(n)/C)$ register spills [1]

- Scheduler achieves this (asymptotic) bound independent of C

- Explain on blackboard