Benchmarking Sparse Matrix-Vector Multiply In 5 Minutes

Hormozd Gahvari, Mark Hoemmen, James Demmel, and Kathy Yelick

January 21, 2007

Outline

- What is Sparse Matrix-Vector Multiply (SpMV)? Why benchmark it?
- How to benchmark it?
  - Past approaches
  - Our approach
- Results
- Conclusions and directions for future work

SpMV

- Sparse Matrix-(dense)Vector Multiply
  - Multiply a dense vector by a sparse matrix (one whose entries are mostly zeroes)
- Why do we need a benchmark?
  - SpMV is an important kernel in scientific computation
  - Vendors need to know how well their machines perform it
  - Consumers need to know which machines to buy
  - Existing benchmarks do a poor job of approximating SpMV

Existing Benchmarks

- The most widely used method for ranking computers is still the LINPACK benchmark, used exclusively by the Top 500 supercomputer list
- Benchmark suites like the High Performance Computing Challenge (HPCC) Suite seek to change this by including other benchmarks
- Even the benchmarks in HPCC do not model SpMV however
- This work is proposed for inclusion into the HPCC suite

Benchmarking SpMV is hard!

- Issues to consider:
  - Matrix formats
  - Memory access patterns
  - Performance optimizations and why we need to benchmark them
- Preexisting benchmarks that perform SpMV do not take all of this into account

Matrix Formats

- We store only the nonzero entries in sparse matrices
- This leads to multiple ways of storing the data, based on how we index it
  - Coordinate, CSR, CSC, ELLPACK,…
- Use Compressed Sparse Row (CSR) as our baseline format as it provides best overall unoptimized performance across many architectures

CSR SpMV Example

$$\begin{pmatrix} 1 & 2 & & & \\ 3 & & 4 & & \\ & 5 & & 6 & \\ & & 7 & & 8 \end{pmatrix}$$

(M,N) = (4,5)

NNZ = 8

row_start:

(0,2,4,6,8)

col_idx:

(0,1,0,2,1,3,2,4)

values:

(1,2,3,4,5,6,7,8)

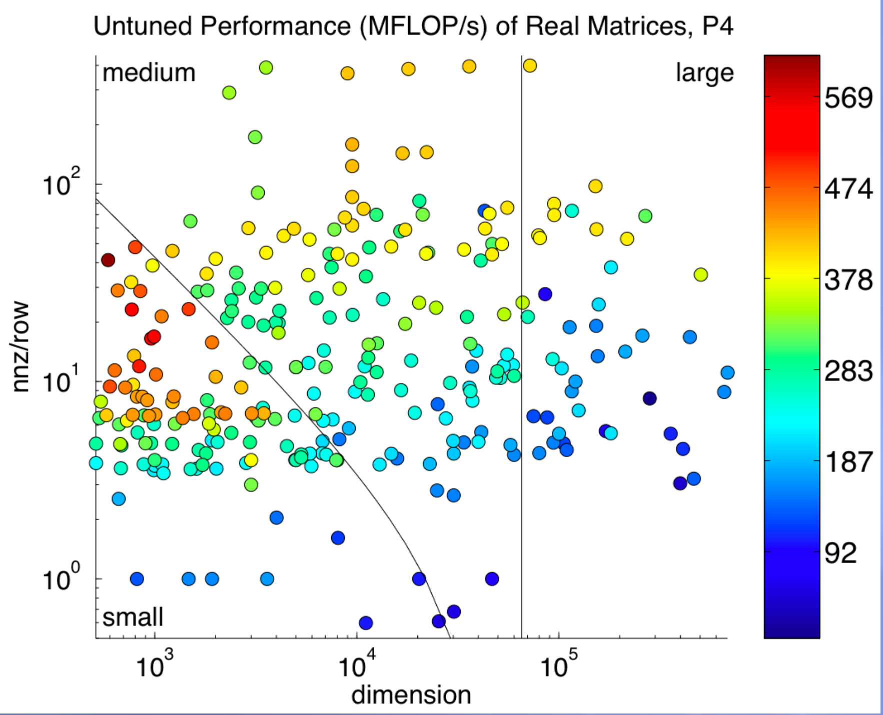
Memory Access Patterns

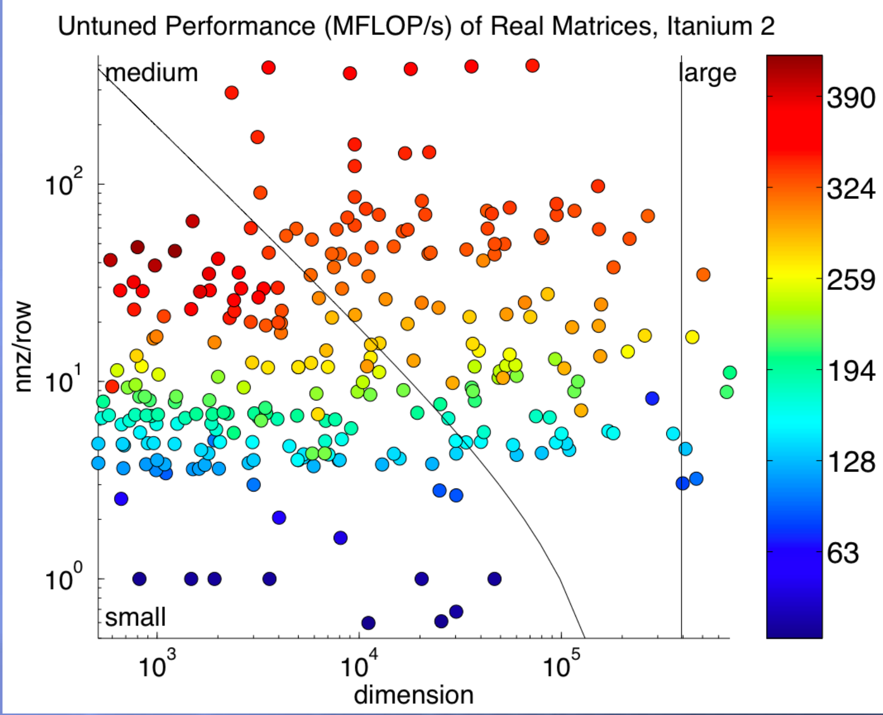
- Unlike dense case, memory access patterns differ for matrix and vector elements
  - Matrix elements: unit stride
  - Vector elements: indirect access for the source vector (the one multiplied by the matrix)
- This leads us to propose three categories for SpMV problems:
  - Small: everything fits in cache
  - Medium: source vector fits in cache, matrix does not
  - Large: source vector does not fit in cache
- These categories will exercise the memory hierarchy differently and so may perform differently

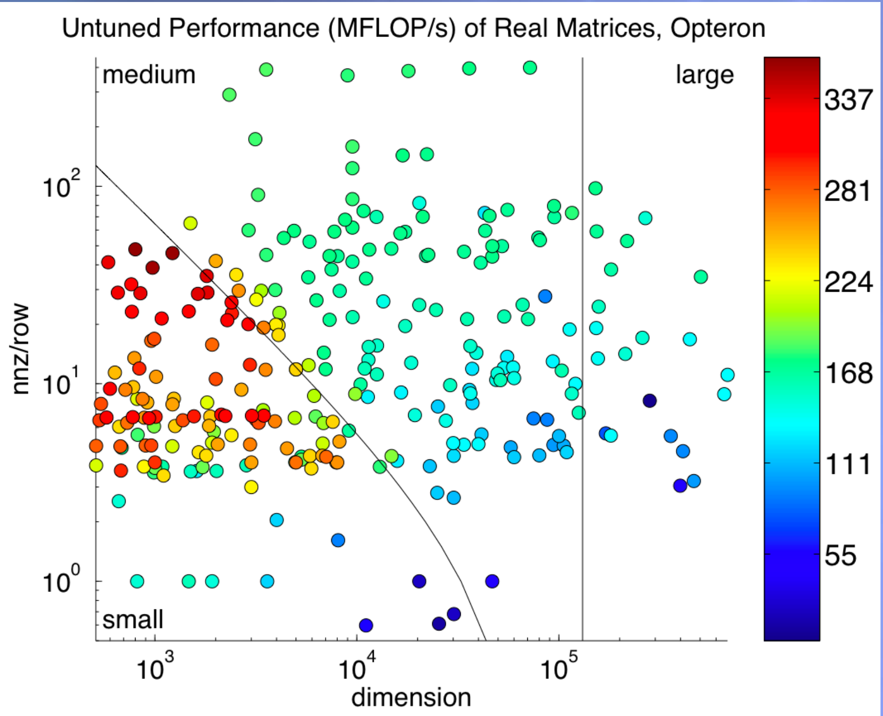
Examples from Three Platforms

- Intel Pentium 4
  - 2.4 GHz
  - 512 KB cache
- Intel Itanium 2
  - 1 GHz
  - 3 MB cache
- AMD Opteron
  - 1.4 GHz
  - 1 MB cache

- Data collected using a test suite of 275 matrices taken from the University of Florida Sparse Matrix Collection
- Performance is graphed vs. problem size







horizontal axis = matrix dimension or vector length

vertical axis = density in nnz/row

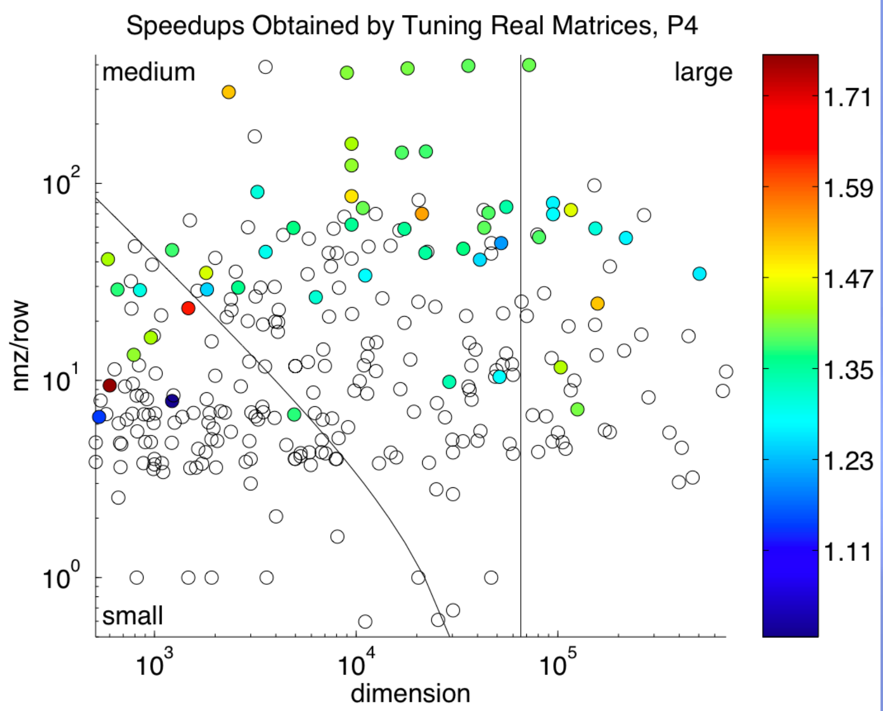
colored dots represent unoptimized performance of real matrices

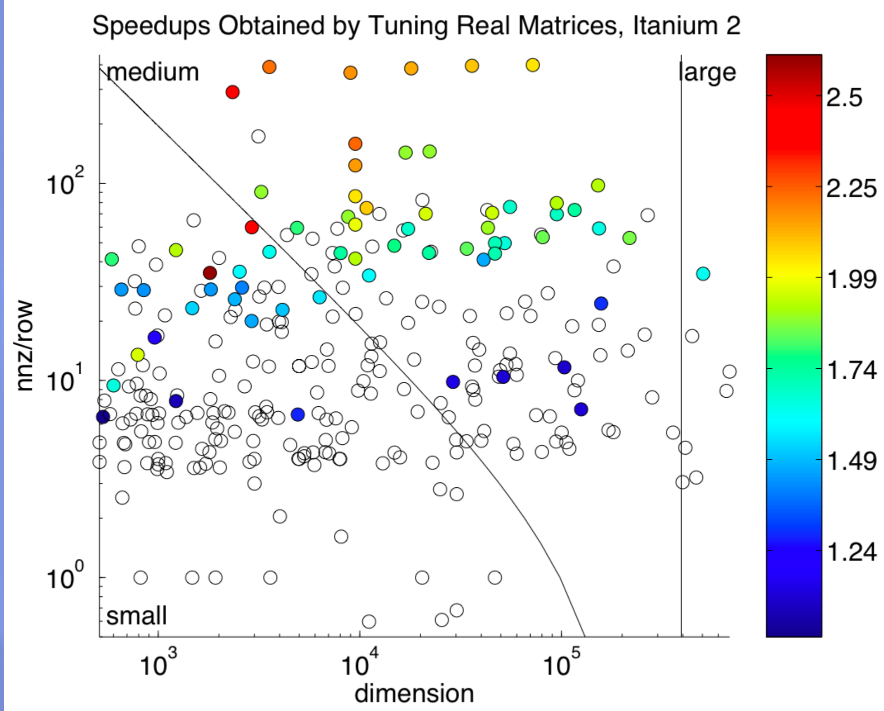
Performance Optimizations

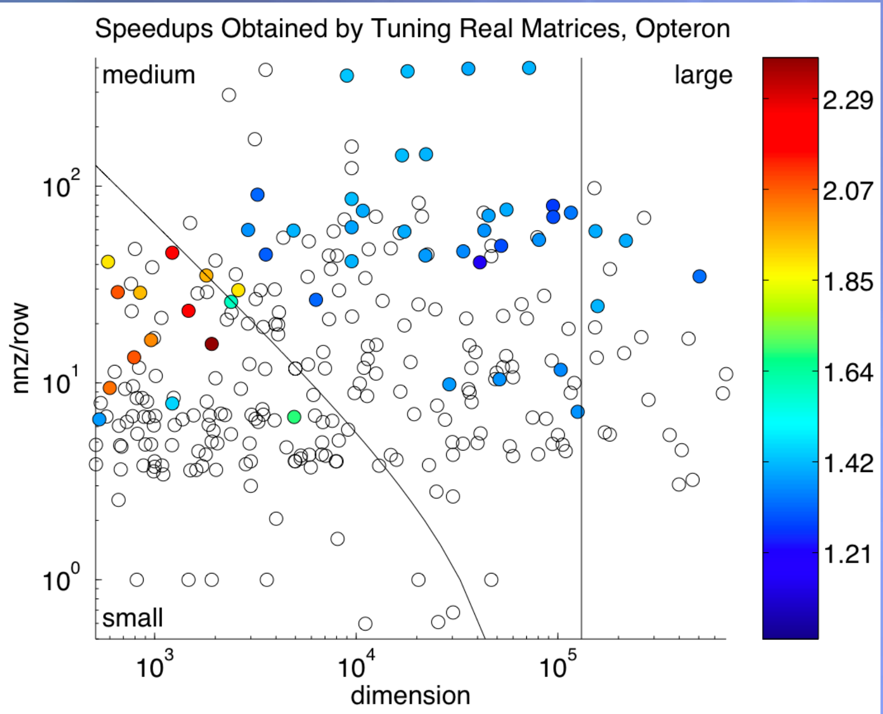
- Many different optimizations possible
- One family of optimizations involves blocking the matrix to improve reuse at a particular level of the memory hierarchy
  - Register blocking - very often useful
  - Cache blocking - not as useful
- Which optimizations to use?
  - HPCC framework allows significant optimization by the user - we don’t want to go as far
  - Automatic tuning at runtime permits a reasonable comparison of architectures, by trying the same optimizations on each one
  - We will use only the register-blocking optimization (BCSR), which is implemented in the OSKI automatic tuning system for sparse matrix kernels developed at Berkeley
  - Prior research has found register blocking to be applicable to a number of real-world matrices, particularly ones from finite element applications

Both unoptimized and optimized SpMV matter

- Why we need to measure optimized SpMV:
  - Some platforms benefit more from performance tuning than others
  - In the case of the tested platforms, Itanium 2 and Opteron gain vs. P4 when we tune using OSKI
- Why we need to measure unoptimized SpMV:
  - Some SpMV problems are more resistant to optimization
  - To be effective, register blocking needs a matrix with a dense block structure
  - Not all sparse matrices have one
- Graphs on next slide illustrate this







horizontal axis = matrix dimension or vector length

vertical axis = density in nnz/row

blank dots represent real matrices that OSKI could not tune due to lack of a dense block structure

colored dots represent speedups obtained by OSKI's tuning

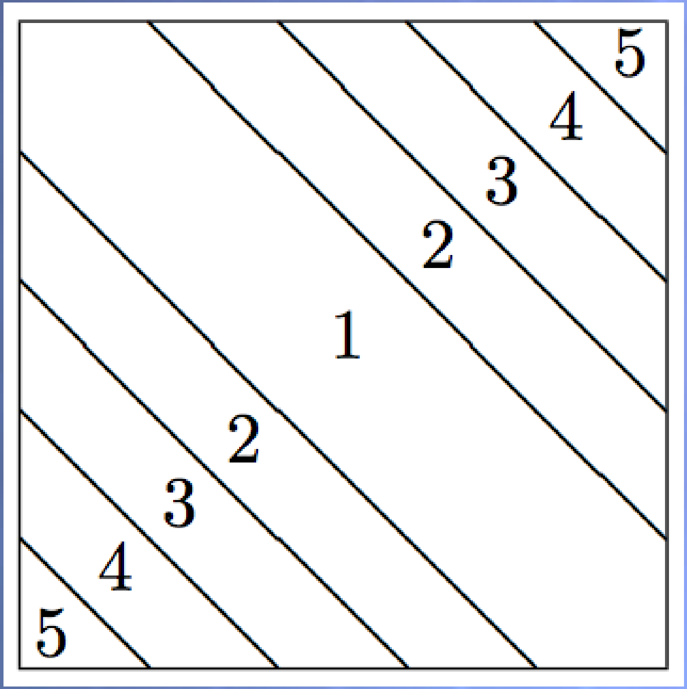
So what do we do?

- We have a large search space of matrices to examine
- We could just do lots of SpMV on real-world matrices. However
  - It’s not portable. Several GB to store and transport. Our test suite takes up 8.34 GB of space
  - Appropriate set of matrices is always changing as machines grow larger
- Instead, we can randomly generate sparse matrices that mirror real-world matrices by matching certain properties of these matrices

Matching Real Matrices With Synthetic Ones

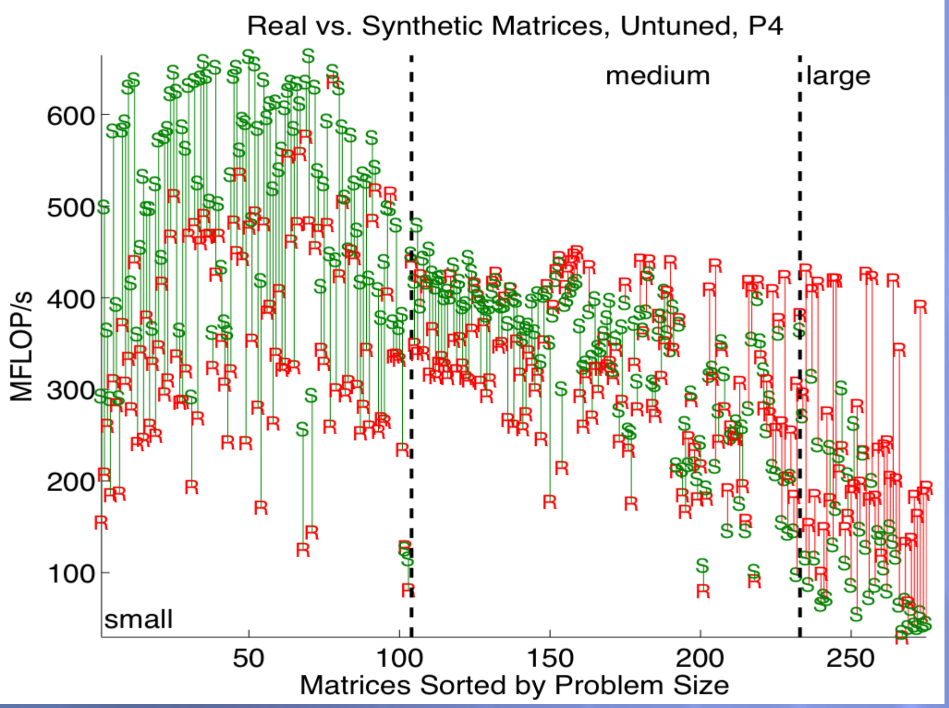
- Randomly generated matrices for each of 275 matrices taken from the Florida collection
- Matched real matrices in dimension, density (measured in NNZ/row), blocksize, and distribution of nonzero entries
- Nonzero distribution was measured for each matrix by looking at what fraction of nonzero entries are in bands a certain percentage away from the main diagonal

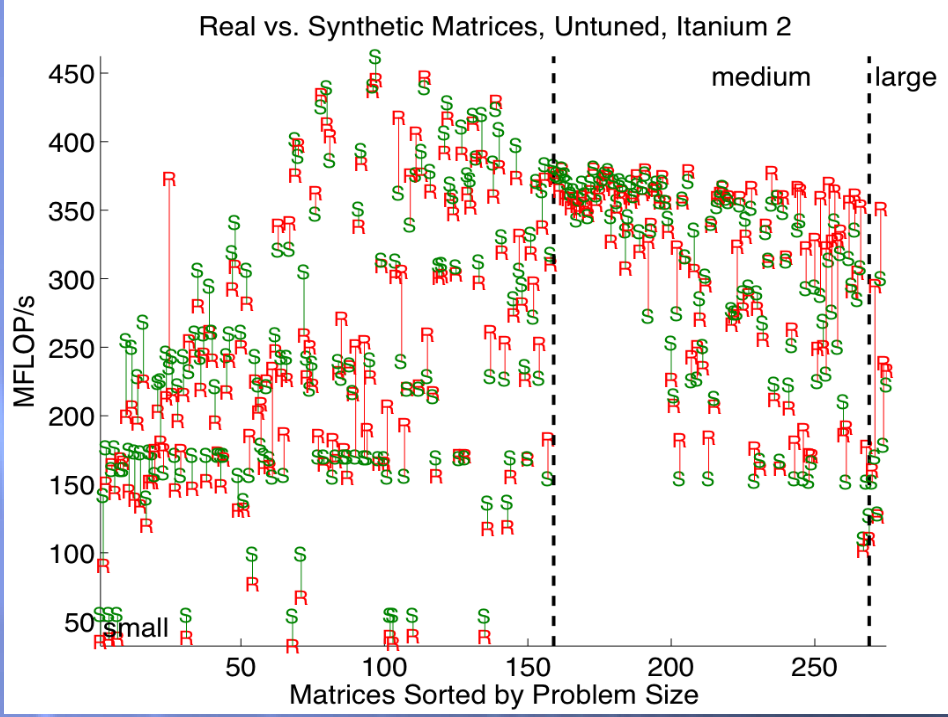
Band Distribution Illustration

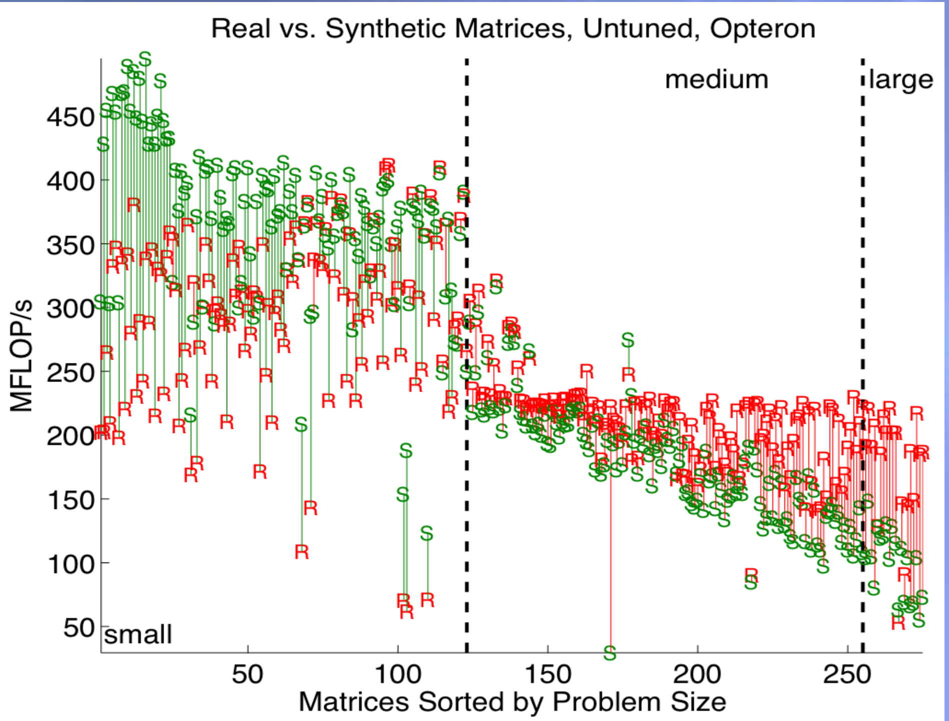


What proportion of the nonzero entries fall into each of these bands 1-5?

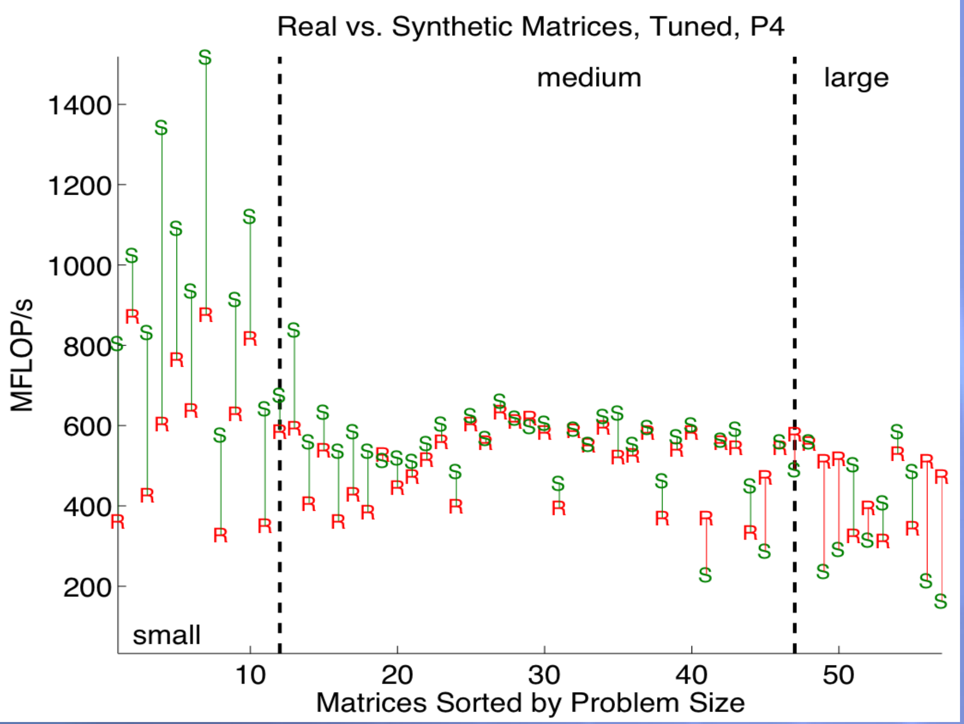
We use 10 bands instead of 5, but have shown 5 for simplicity.

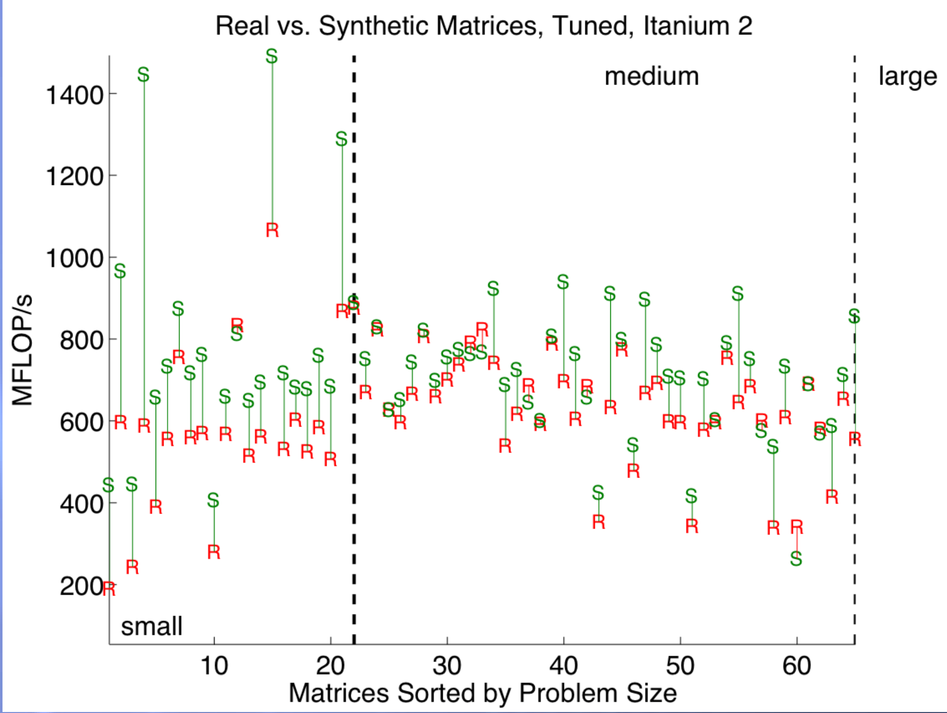


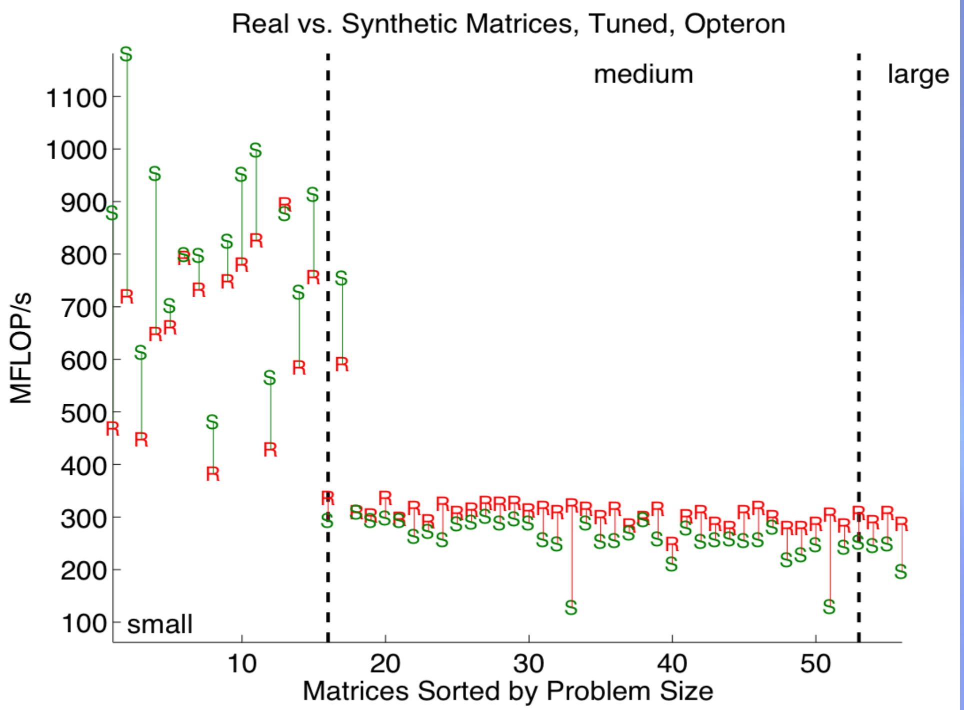




In these graphs, real matrices are denoted by a red R, and synthetic matrices by a green S. Real matrices are connected by a line whose color indicates which matrix was faster to the synthetic matrices created to approximate them.


Real vs. Synthetic Matrices, Tuned, P4
medium
large
small
MFLOP/s
Matrices Sorted by Problem Size
S
R


Real vs. Synthetic Matrices, Tuned, Itanium 2
medium
large
small
MFLOP/s
Matrices Sorted by Problem Size
S
R


Real vs. Synthetic Matrices, Tuned, Opteron
medium
large
small
MFLOP/s
Matrices Sorted by Problem Size
S
R

Remaining Issues

- We’ve found a reasonable way to model real matrices, but benchmark suites want less output. HPCC wants us to report only a few numbers, preferably just one
- Challenges in getting there
  - As we’ve seen, SpMV performance depends greatly on the matrix, and there is a large range of problem sizes. How do we capture this all? Stats on Florida matrices:
    - Dimension ranges from a few hundred to over a million
    - NNZ/row ranges from 1 to a few hundred
  - How to capture performance of matrices with small dense blocks that benefit from register blocking?
- What we’ll do:
  - Bound the set of synthetic matrices we generate
  - Determine which numbers to report that we feel capture the data best

Bounding the Benchmark Set

- Limit to square matrices
- Look over only a certain range of problem dimensions and NNZ/row
  - Since dimension range is so huge, restrict dimension to powers of 2
- Limit blocksizes tested to ones in {1,2,3,4,6,8} x {1,2,3,4,6,8}
  - These were the most common ones encountered in prior research with matrices that mostly had dense block structures
- Here are the limits based on the matrix test suite:
  - Dimension <= 2^20 (a little over one million)
  - 24 <= NNZ/row <= 34 (avg. NNZ/row for real matrix test suite is 29)
- Generate matrices with nonzero entries distributed (band distribution) based on statistics for the test suite as a whole

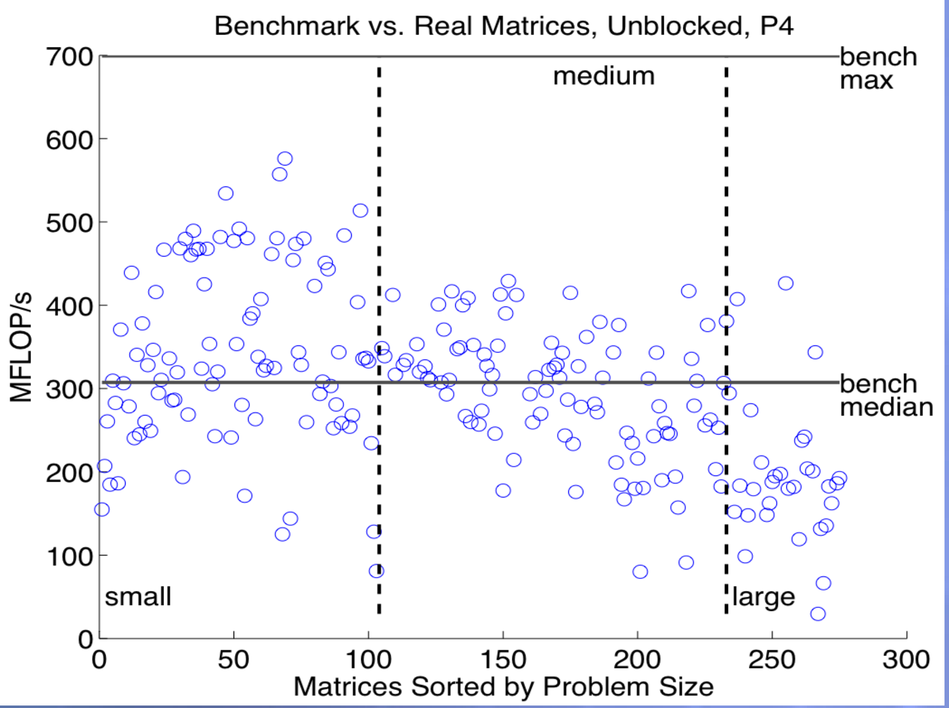
Condensing the Data

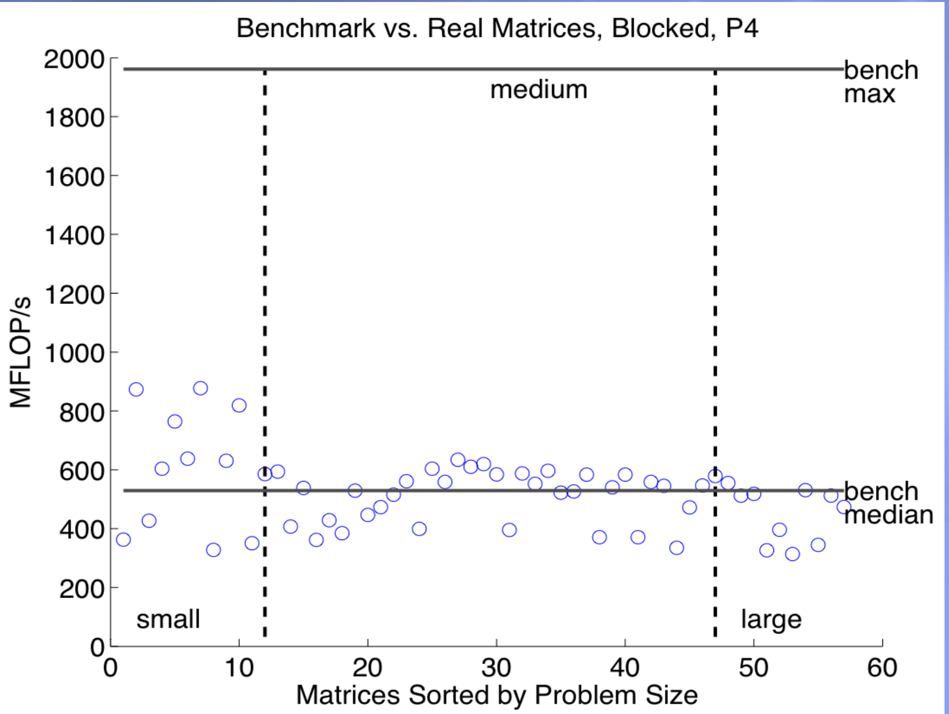
- This is a lot of data
  - 11 x 12 x 36 = 4752 matrices to run
- Tuned and untuned cases are separated, as they highlight differences between platforms
  - Untuned data will only come from unblocked matrices
  - Tuned data will come from the remaining (blocked) matrices
- In each case (blocked and unblocked), report the maximum and median MFLOP rates to capture small/medium/large behavior
- When forced to report one number, report the blocked median

Output

	Unblocked		Blocked	
	Max	Median	Max	Median
Pentium 4	699	307	1961	530
Itanium 2	443	343	2177	753
Opteron	396	170	1178	273

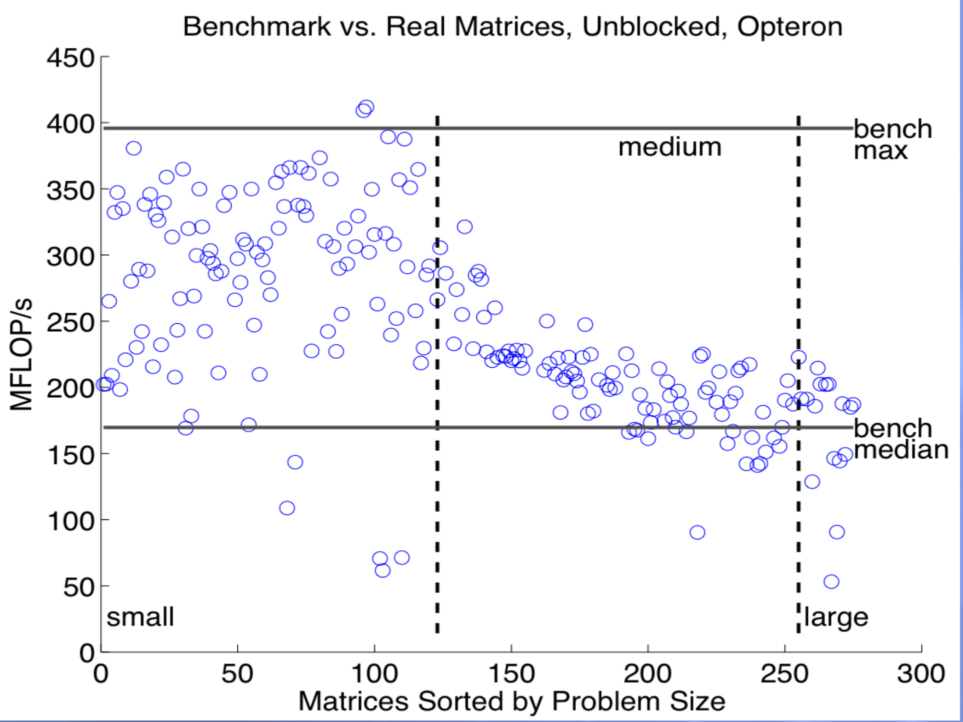
(all numbers MFLOP/s)

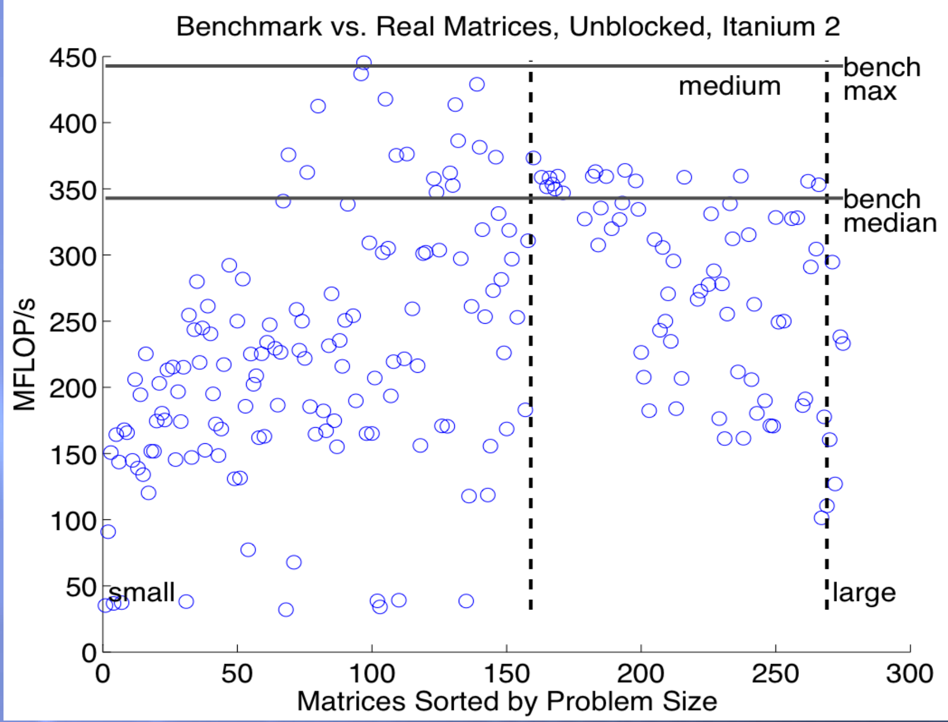


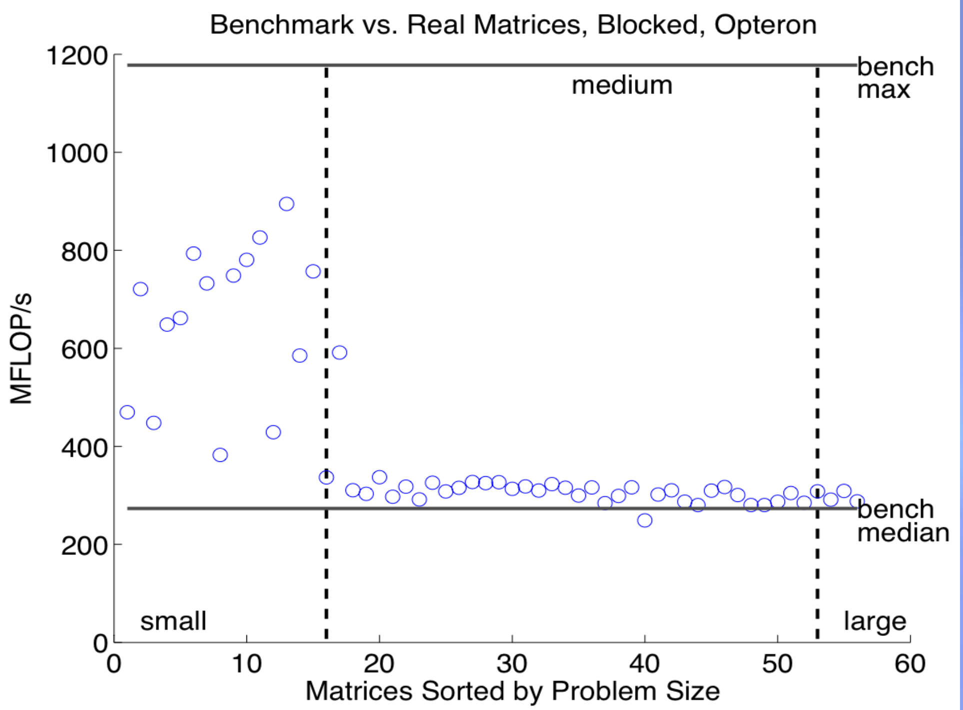


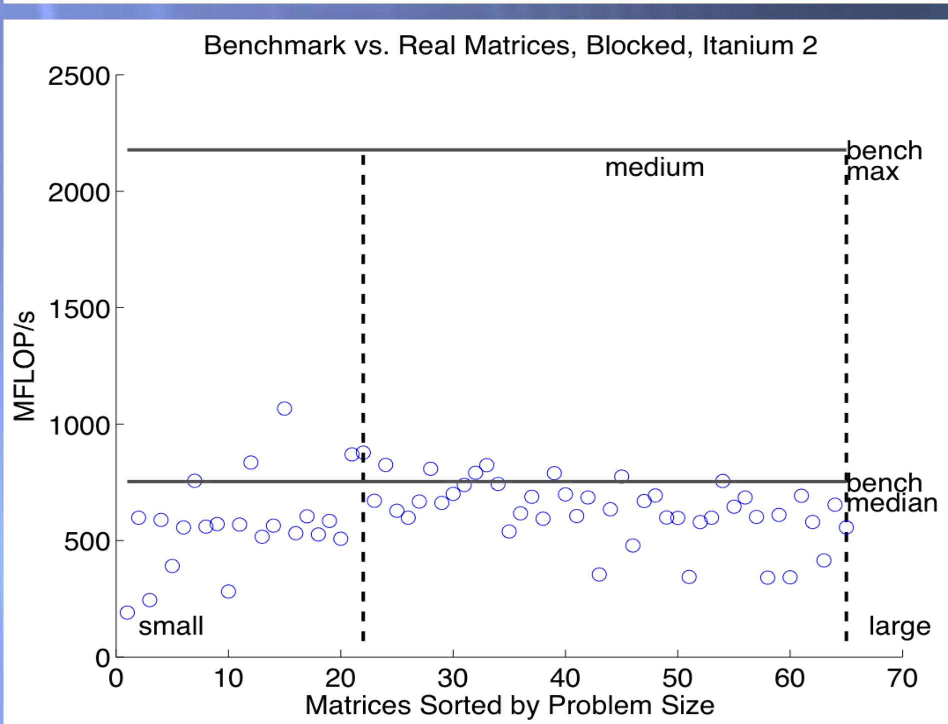
How well does the benchmark approximate real SpMV performance?

These graphs show the benchmark numbers as horizontal lines versus the real matrices which are denoted by circles.


Benchmark vs. Real Matrices, Unblocked, Opteron
MFLOP/s
Matrices Sorted by Problem Size
bench max
bench median
small
medium
large


Benchmark vs. Real Matrices, Unblocked, Itanium 2
MFLOP/s
Matrices Sorted by Problem Size
bench max
bench median
small
medium
large


Benchmark vs. Real Matrices, Blocked, Opteron
MFLOP/s
Matrices Sorted by Problem Size
bench max
bench median
small
medium
large


Benchmark vs. Real Matrices, Blocked, Itanium 2
MFLOP/s
Matrices Sorted by Problem Size
bench max
bench median
small
medium
large

Output

- Matrices generated by the benchmark fall into small/medium/large categories as follows:

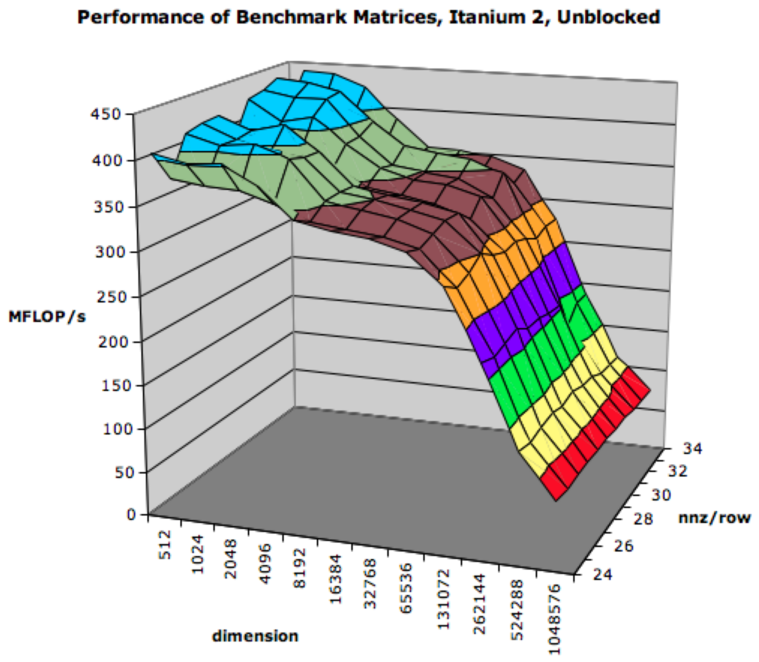
	Pentium 4	Itanium 2	Opteron
Small	17%	33%	23%
Medium	42%	50%	44%
Large	42%	17%	33%

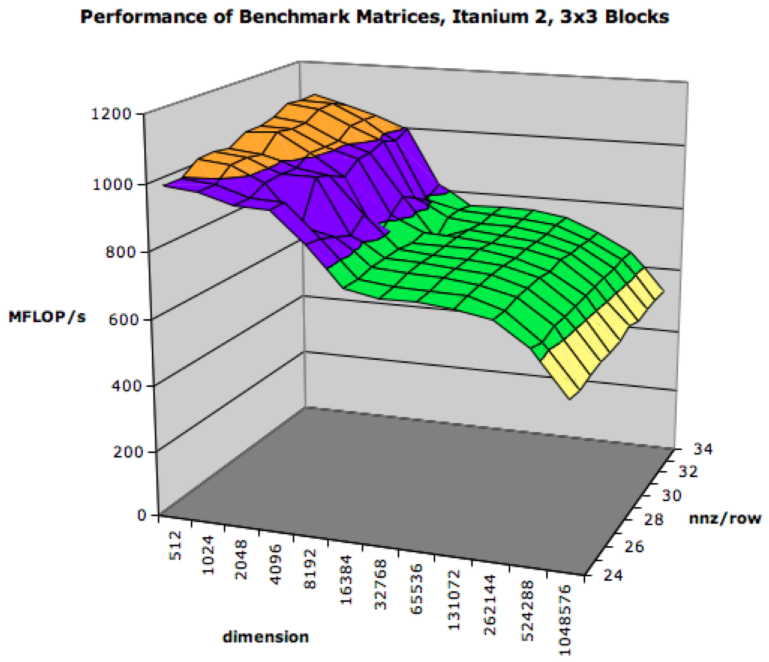
One More Problem

- Takes too long to run:
  - Pentium 4: 150 minutes
  - Itanium 2: 128 minutes
  - Opteron: 149 minutes
- How to cut down on this? HPCC would like our benchmark to run in 5 minutes

Cutting Runtime

- Test fewer problem dimensions
  - The largest ones do not give any extra information
- Test fewer NNZ/row
  - Once dimension gets large enough, small variations in NNZ/row have little effect
- These decisions are all made by a runtime estimation algorithm
- Benchmark SpMV data supports this





Sample graphs of benchmark SpMV for 1x1 and 3x3 blocked matrices

Output Comparison

	Unblocked		Blocked	
	Max	Median	Max	Median
Pentium 4	692 (699)	362 (307)	1937 (1961)	555 (530)
Itanium 2	442 (443)	343 (343)	2181 (2177)	803 (753)
Opteron	394 (396)	188 (170)	1178 (1178)	286 (273)

Runtime Comparison

	Full	Shortened
Pentium 4	150 min	3 min
Itanium 2	128 min	3 min
Opteron	149 min	3 min

Conclusions and Directions for the Future

- SpMV is hard to benchmark because performance varies greatly depending on the matrix
- Carefully chosen synthetic matrices can be used to approximate SpMV
- A benchmark that reports one number and runs quickly is harder, but we can do reasonably well by looking at the median
- In the future:
  - Tighter maximum numbers
  - Parallel version
- Software available at http://bebop.cs.berkeley.edu