Adaptable benchmarks for register blocked sparse matrix-vector multiplication

- Berkeley Benchmarking and Optimization group (BeBOP)
- Hormozd Gahvari and Mark Hoemmen
- Based on research of:
  - Eun-Jin Im
  - Rich Vuduc

Outline

- Sparse matrix-vector multiplication
- (Optimized) sparse data structures
- Current benchmarks and ours

Sparse matrix-vector multiplication (SpMV)

- “Stream” through nonzero A(i,j)
  - Do y(i) += A(i,j) * x(j)
- SpMV expenses:
  - Limited data reuse
  - Index lookups / computations
  - Already low FP/memory balance
- Performance < 10% peak
  - DGEMV: 10-30%

Sparse formats for SpMV

- Many!
  - SPARSKIT: supports 13
- No “best,” but for SpMV:
- Compressed Sparse Row (/Column)
  - Modern cache-based processors
- Others:
  - Ellpack/Itpack, Jagged Diagonal (JAD)
    - Vector machines

Compressed Sparse Row (CSR)

- Two integers:
  - M: number of rows
  - NNZ: number of nonzero elements
- Three arrays:
  - int row_start[M+1];
  - int col_idx[NNZ];
  - double values[NNZ];

CSR example

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

- (M,N)=(4,5)
- NNZ = 8
- row_start:
  - (0,2,4,6,7)
- col_idx:
  - (0,1,0,2,1,3,2,4)
- values:
  - (1,2,3,4,5,6,7,8)

CSR SpMV

```
for (i = 0; i < M; i++)
  for (j = row_start[i];
       j < row_start[i+1]; j++)
    {
      y[i] += values[j] *
 x[col_idx[j]];
    }
```

CSR advantages

- Sequential accesses
- Precomputed indices
- Index reuse (row_start)
- Destination vector reuse (y(i))
- Popular
  - PETSc, SPARSKIT
  - Variants (AZTEC: MSR)

CSR disadvantages

- Indirect indexing: x[col_idx[j]]
- Locality suffers
- Ignores dense substructures!
  - BLAS 2/3: dense ==> reuse
  - (Reordered) FEM matrices:
    - Small dense blocks
    - 2x2, 3x3, 6x6

Register block optimization: BCSR

- Each nonzero “elem”:
  - Now: block of nonzeros (“Register block”)
  - Contiguous in values[]
  - Indices point to block starts
- “Loop unrolled” SpMV
  - Source vector (x) reuse
  - Exploit specialized FP hardware

BCSR SpMV: 2 x 3

```
for (i = 0; i < M; i++, y += 2)
  {
    y0 = y[0]; y1 = y[1];

    for (j = row_start[i]; j < row_start[i+1];
         j++, val += 6)
      {
        k = col_idx[j];
        x0 = x[k]; x1 = x[k+1]; x2 = x[k+2];

        y0 += val[0]*x0; y1 += val[3]*x0;
        y0 += val[1]*x1; y1 += val[4]*x1;
        y0 += val[2]*x2; y1 += val[5]*x2;
      }
    y[0] = y0; y[1] = y1;
  }
```

BCSR pays off

- At best 31% peak, 4 x speedup
- Nearing (dense) DGEMV
- Large payoff + repeated SpMV ==>
  - overcomes CSR->BCSR conversion costs

SpMV benchmarks: Three strategies

1) "Simulation" with simpler ops

2) Analytic bounds

3) Do SpMV, with:

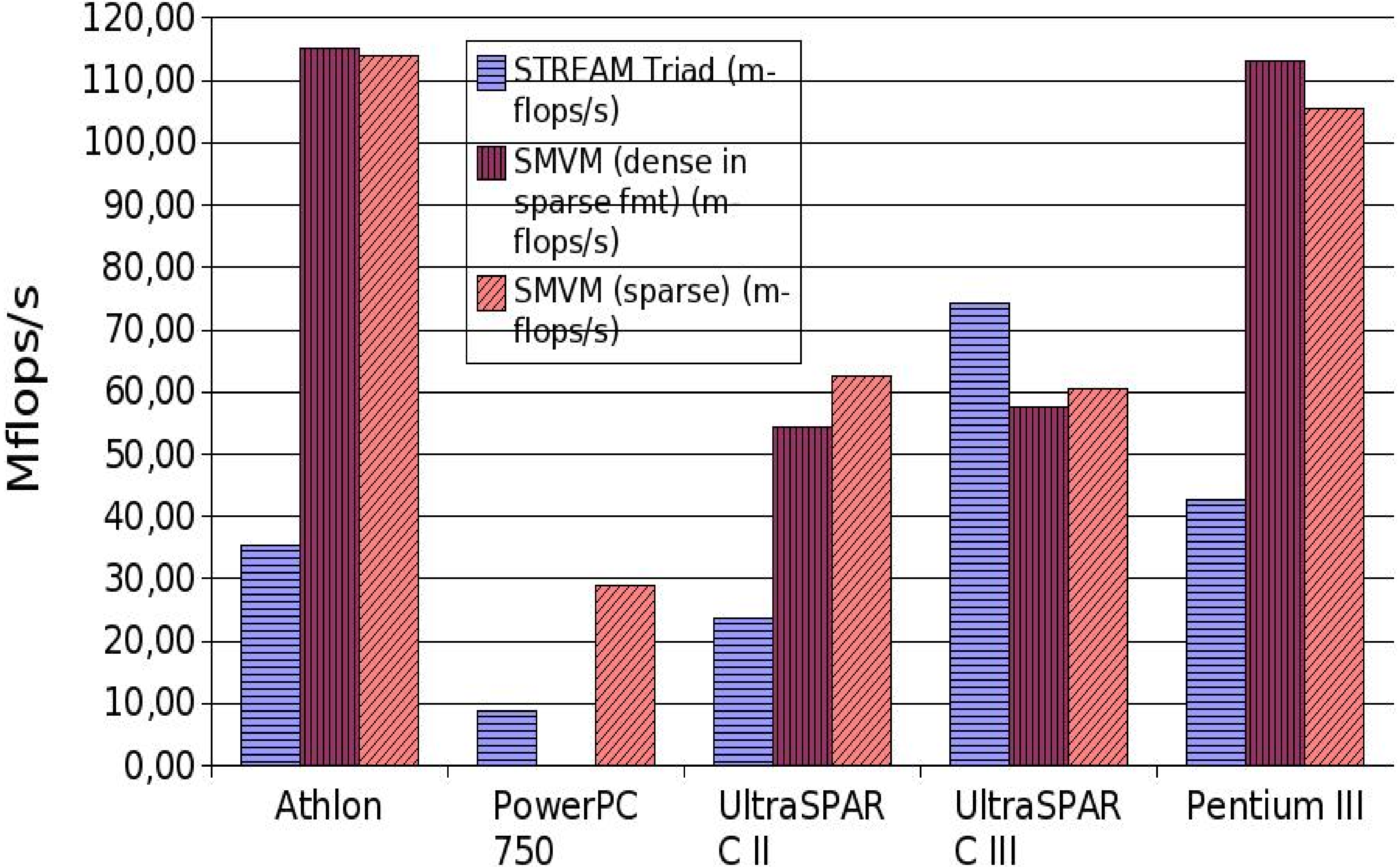
   a) "Real-life" matrices

   b) Generated test problems

1) Simulations of SpMV

- STREAM http://www.streambench.org/
  - Triad: a[i] = b[i] + s*c[i]
  - Dense level-1 BLAS DAXPY
  - No discerning power
    - e.g. Next slide
- Indirect indexed variants
  - x[col_idx[j]] simulation
  - Still not predictive

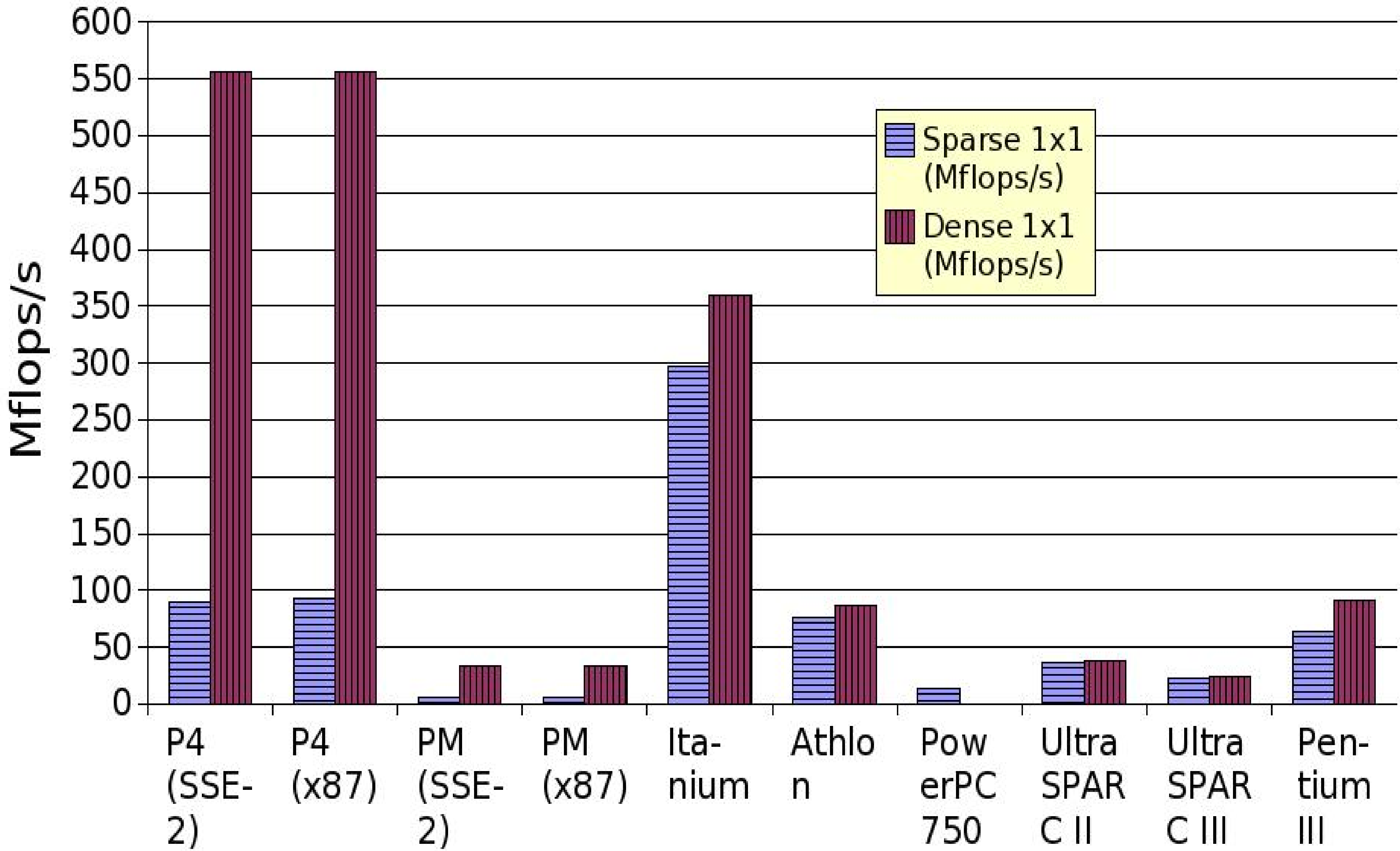
Three SMVM benchmarks: Weaker performers



Dense in sparse (CSR) format

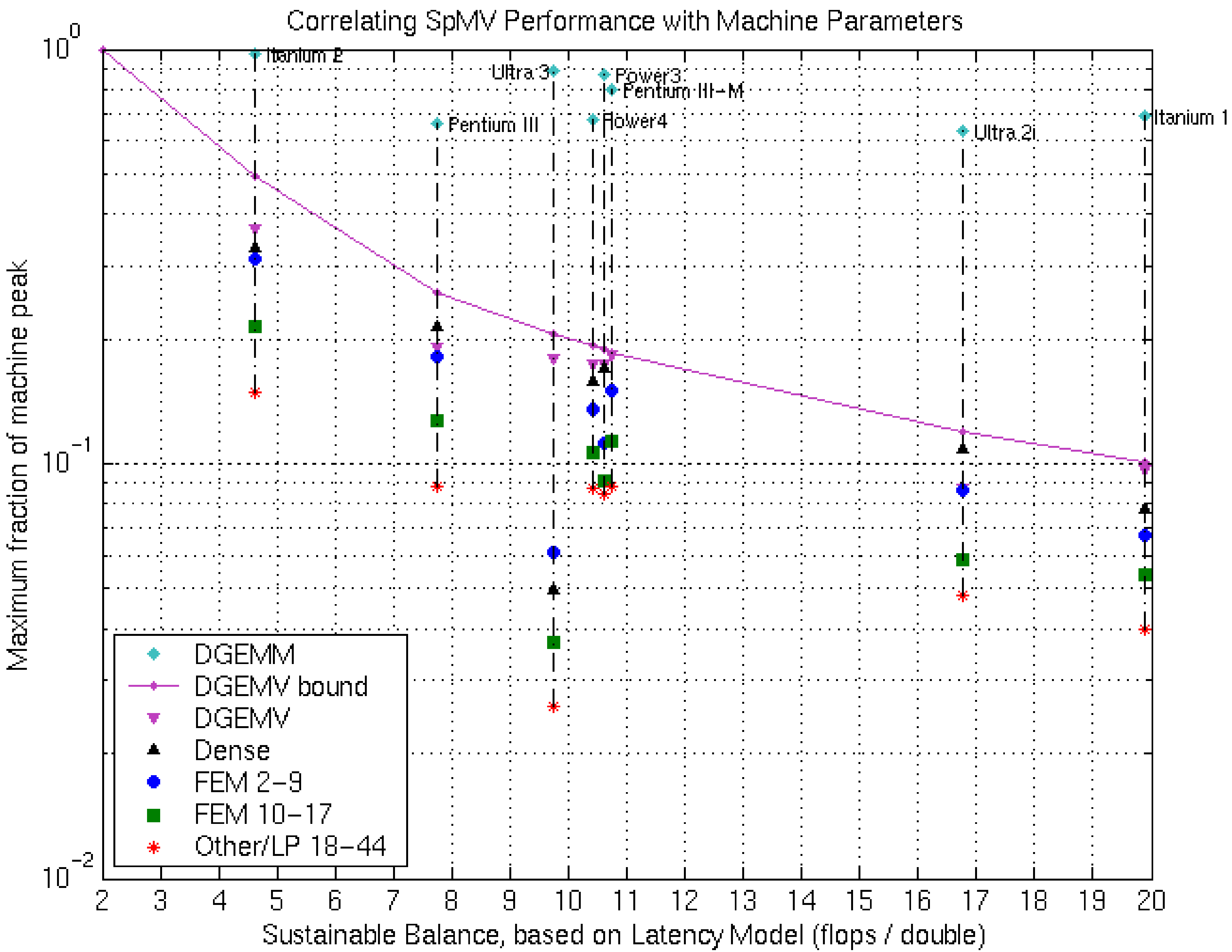
- Heuristic in Sparsity system
- Helpful for larger register blocks
- 1x1: no good

SMVM: 1x1 blocks, sparse matrix and dense matrix in sparse format



2) Analytic bounds and correlates

- "Machine balance"
  - Peak FP / sustainable main memory bandwidth
  - as -> 2: SpMV, DGEMV -> peak
- Actual SpMV: Usually w/in 75% of DGEMV; correlates to balance
  - Exception: UltraSparc 3
    - ==> need actual SpMV benchmark

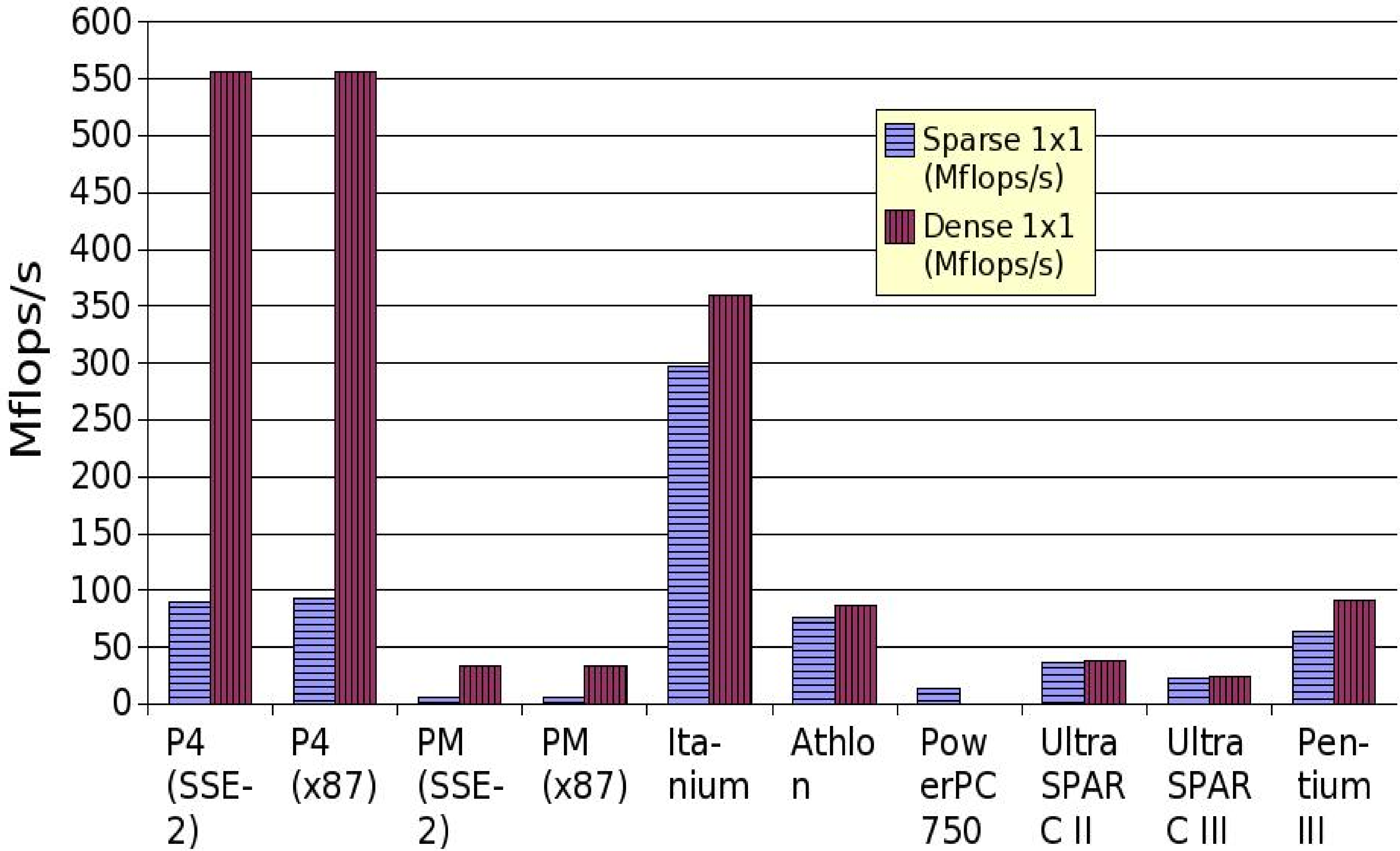


Correlating SpMV Performance with Machine Parameters
Maximum fraction of machine peak
Sustainable Balance, based on Latency Model (flops / double)
10^0
10^{-1}
10^{-2}
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Itanium 2
Ultra 3
Power3
Pentium III−M
Pentium III
Power4
Ultra 2i
Itanium 1
DGEMM
DGEMV bound
DGEMV
Dense
FEM 2−9
FEM 10−17
Other/LP 18−44

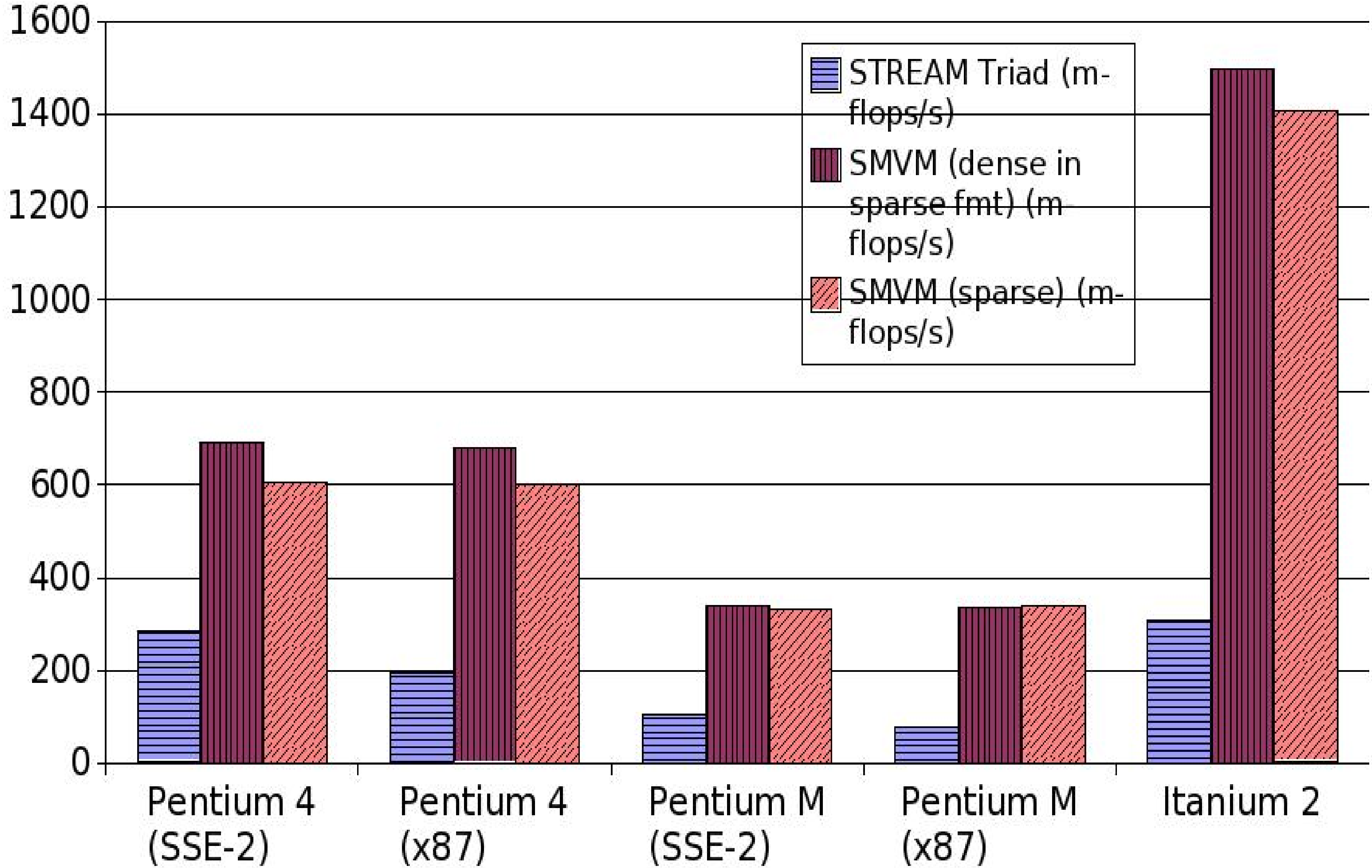
3) CSR SpMV as benchmark

- Regular CSR “unfair”
  - FEM frequent case
  - Analogy: tuned BLAS ==> Matlab v. 6 dropped “flop/s”: So drop CSR!
- BCSR makes processors competitive
  - Pentium M vs. Pentium 4 vs. Athlon:
    - 1x1: P4 >> Athlon > PM
    - Optimal blocks: P4 = 2 x PM = 4 x Athlon

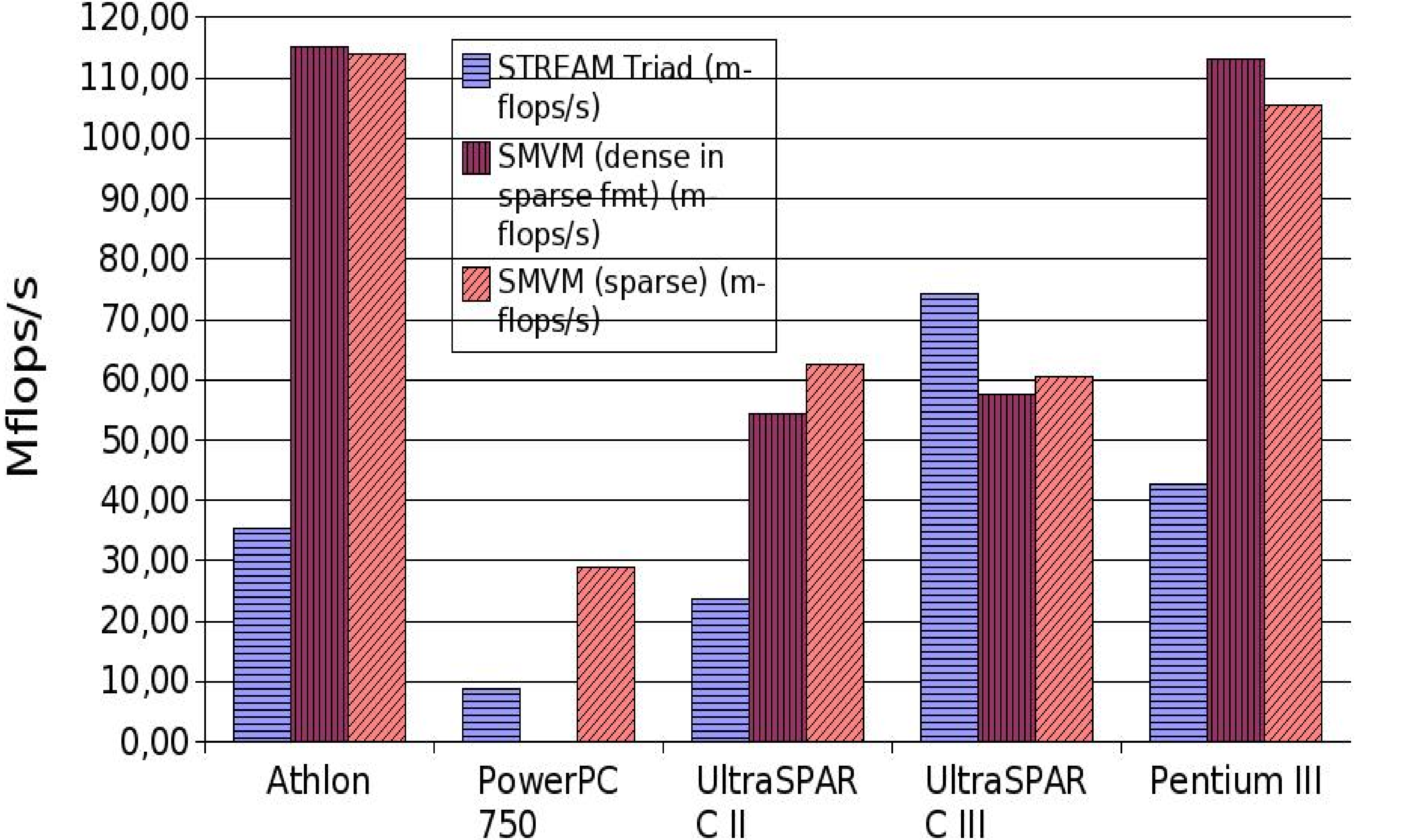
SMVM: 1x1 blocks, sparse matrix and dense matrix in sparse format



Three SMVM benchmarks: Stronger performers



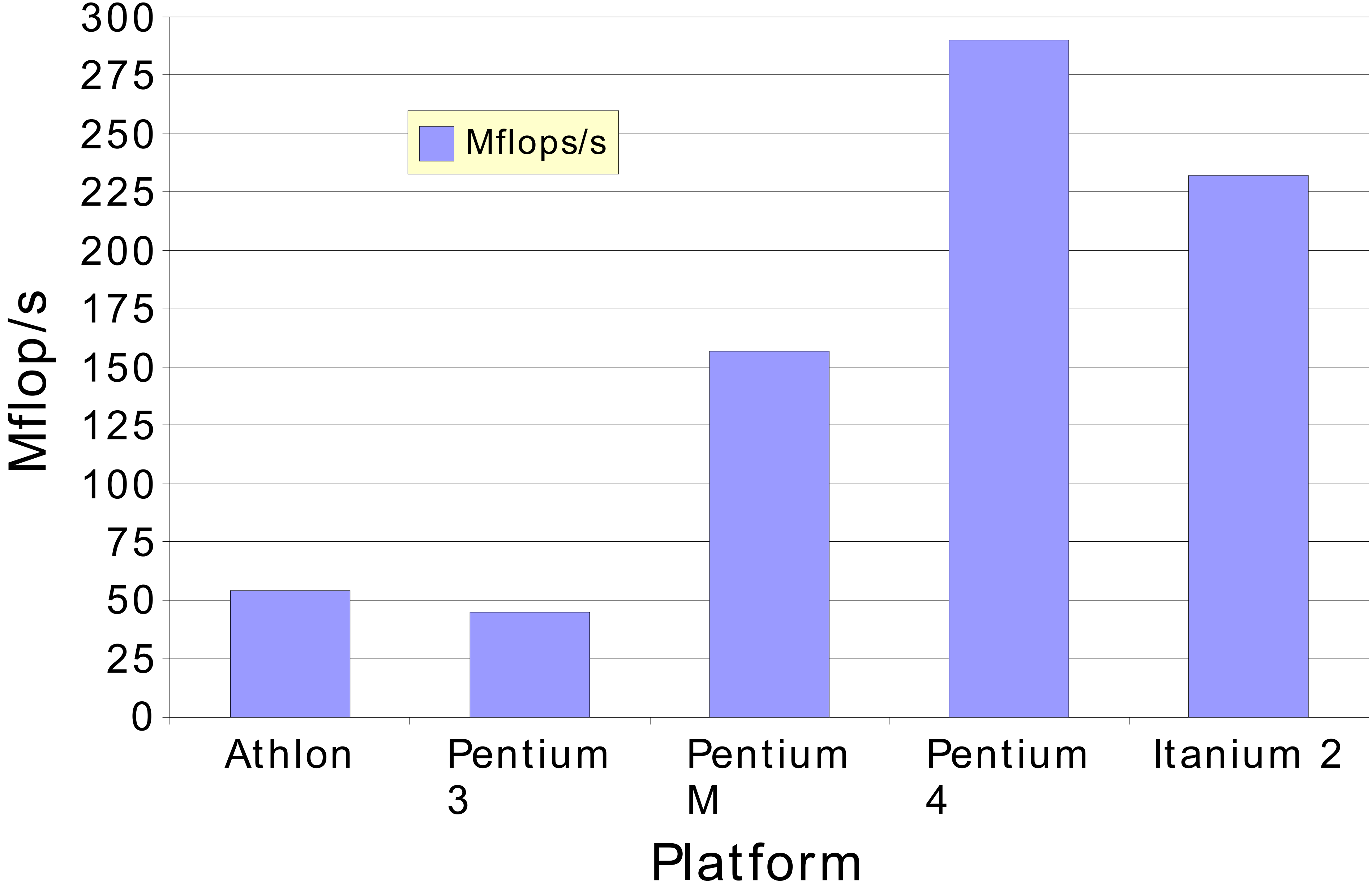
Three SMVM benchmarks: Weaker performers



Standard CSR SpMV benchmarks

- Examples:
  - NAS-CG
  - SparseBench
  - SciMark 2.0 (see next slide – reflects 1x1)
- No BCSR
- CG/GMRES: heterogeneous ops
- Fixed problem sizes, structure

SciMark 2.0 (C) SpMV performance in Mflop/s



3a) “Real-life” matrices

- Storage cost
  - Spark98:
    - 10 matrices
    - 28 MB compressed, 70 MB uncompressed
  - Mark Adams' FEM
    - 1 problem: 60+MB compressed!
  - big ==> fewer examples possible
- Too specific
  - “Average matrix”???
  - Need different block sizes

3a) “Real-life” matrices (con't)

- Fixed benchmark size
  - Caches grow!
  - Sparsity tuning
    - “matrix out of cache”
- Why not “stretch”?
  - Changes x[col_idx[j]] stride
  - “Shrinking” harder
    - Future: less memory per node (BlueGene)

3b) Our strategy: Dynamic generation

- Randomly scattered blocks
  - bandwidth option
- All block dimensions in $\{1,2,3,4,6,8,12\}^2$
- Guaranteed fill
  - runtime parameter

Benchmark output

- Mflop/s for: 1x1, worst, median, best, common block sizes
- Application-specific performance estimates
- 1x1 frequent: e.g. linear programming

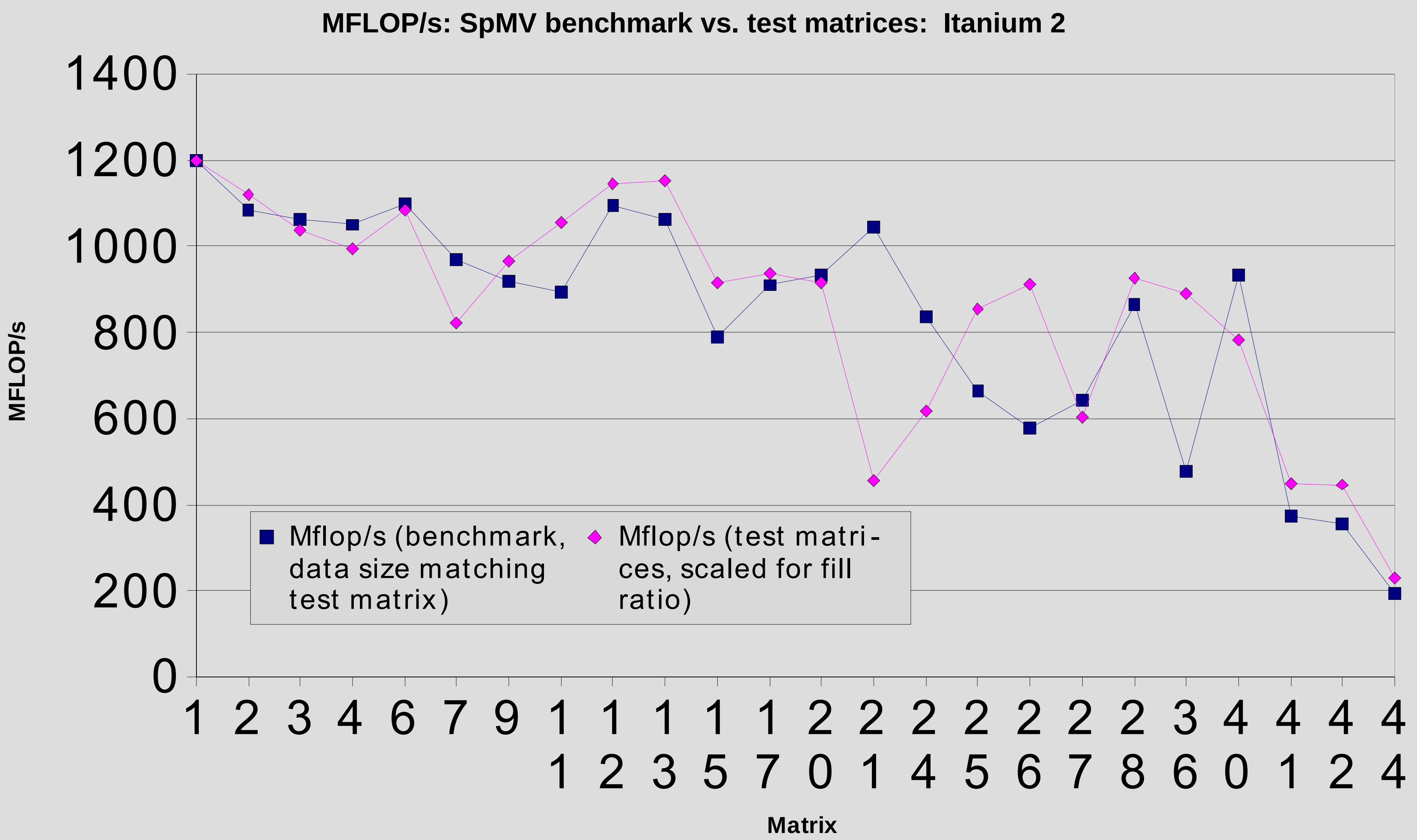
Data set size selection

- Sparsity: vectors in cache, matrix out
  - ==> Choose M, NNZ
- Pluses:
  - Problems grow with processors
  - “Fair”
- Minuses:
  - “Cache”: off-chip? huge?
  - Fill (NNZ / M^2) non-constant
    - x locality
  - No cache? (vector machines)
- Now: memory bound only

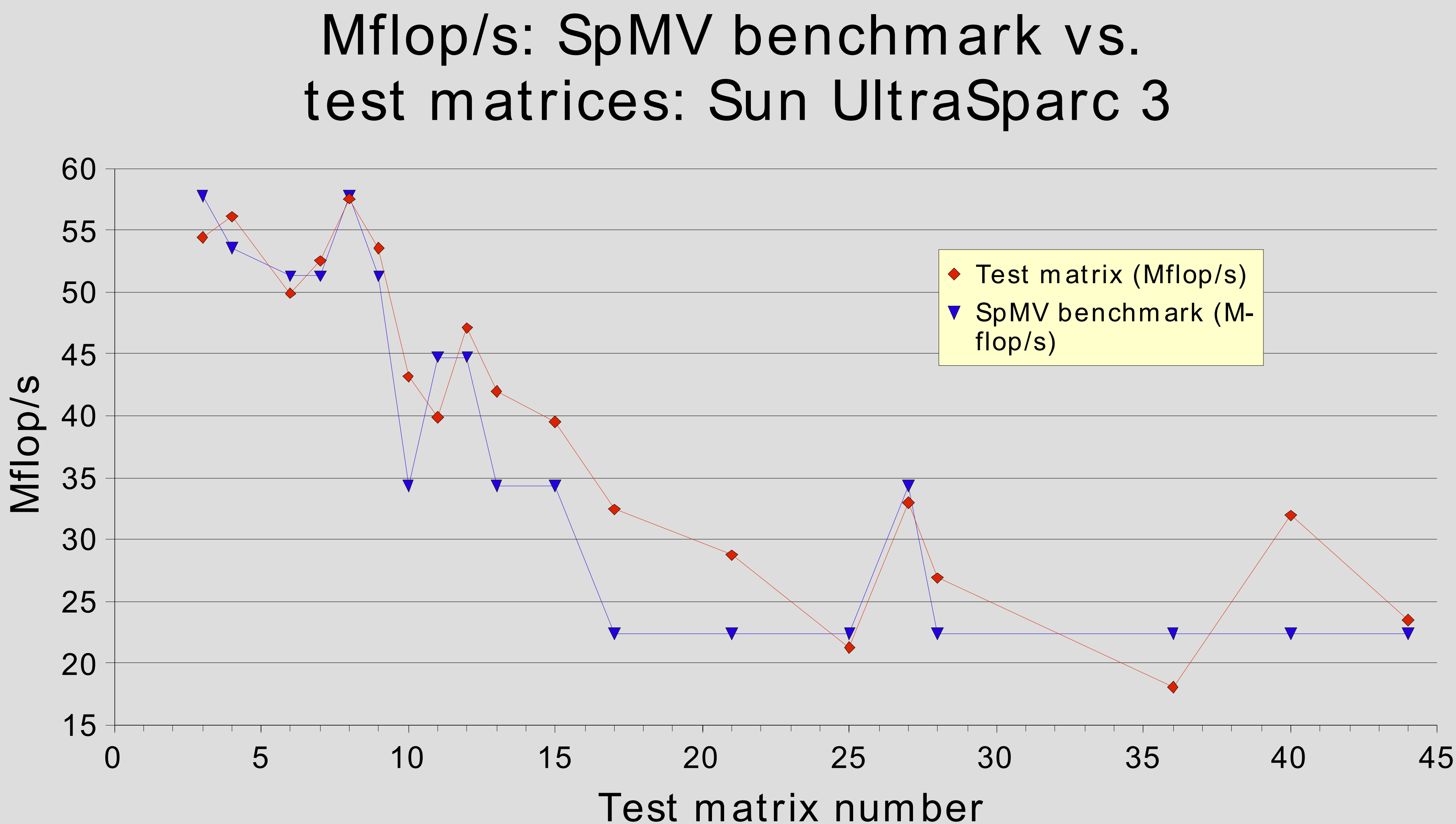
Evaluation: Test matrix suite

- 44 matrices from R. Vuduc's thesis
- 1: Dense in sparse format
- 2-17: FEM
  - 2-9: Regular blocks (fills .044-.39%)
  - 10-17: Irregular (fills .087-.40%)
- 18-44: “non-FEM”
  - 18-39: Assorted
  - 40-44: Linear programming (.021-.12%)
- Scale by test matrix “fill ratio”
  - (block storage efficiency)

Itanium 2 prediction



UltraSparc 3 prediction



Benchmark tuning options

- Diagonal bandwidth
- Fill (NNZ / M^2)
- Nonzero distribution: random seed
- Adjust for target application
- Sensitivity underexplored

Plans

- As component of shared-, distributed-memory parallel SpMV benchmarks
- Incorporation into High-Performance Computing Challenge benchmark suite

Thanks! (1 of 2)

- BeBOP leaders:
  - Profs. James Demmel and Katherine Yelick
- Sparsity SpMV code, assistance:
  - Eun-Jin Im, Rich Vuduc
- Other code contributions:
  - Rajesh Nishtala

Thanks! (2 of 2)

- Computing resources:
  - Argonne National Lab
  - Intel
  - National Energy Research Scientific Computing Center
  - Tyler Berry (tyler@arete.cc)
  - Felipe Gasper (fgasper@fgmusic.org)

Resources

- BeBOP:
  - http://bebop.cs.berkeley.edu/
- HPCC benchmarks:
  - http://icl.cs.utk.edu/hpcc/index.html
- Me:
  - http://mhoemmen.arete.cc/