Advanced Systems Lab

Spring 2022

Lecture: Memory bound computation, sparse linear algebra, OSKI

Instructor: Markus Püschel, Ce Zhang

TA: Joao Rivera, several more

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Overview

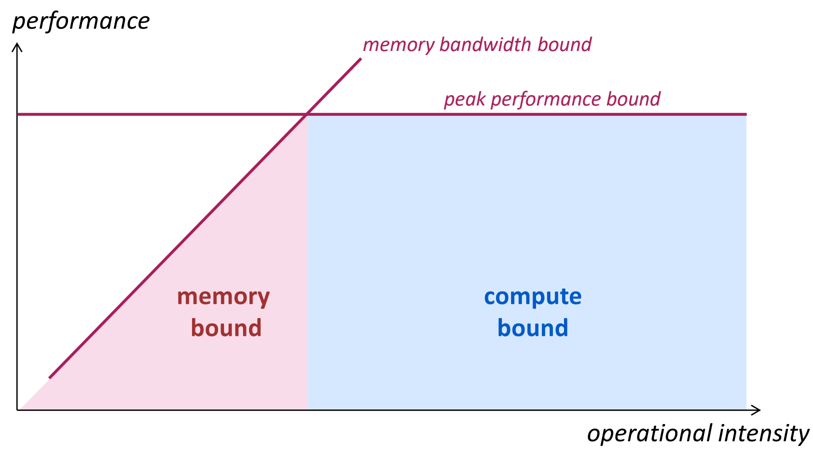
Memory bound computations

Sparse linear algebra, OSKI

Memory Bound Computation

Data movement, not computation, is the bottleneck

Typically: Computations with operational intensity *I(n) = O(1)*



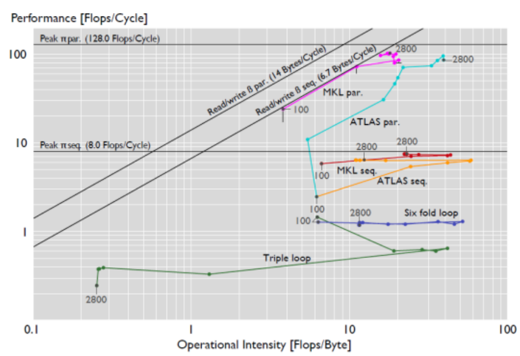
Memory Bound Or Not? Depends On …

The computer
- *Memory bandwidth*
- *Cache size*
- *Peak performance*

The algorithm
- *Dependencies*

How it is implemented
- *Good/bad locality*
- *SIMD or not*

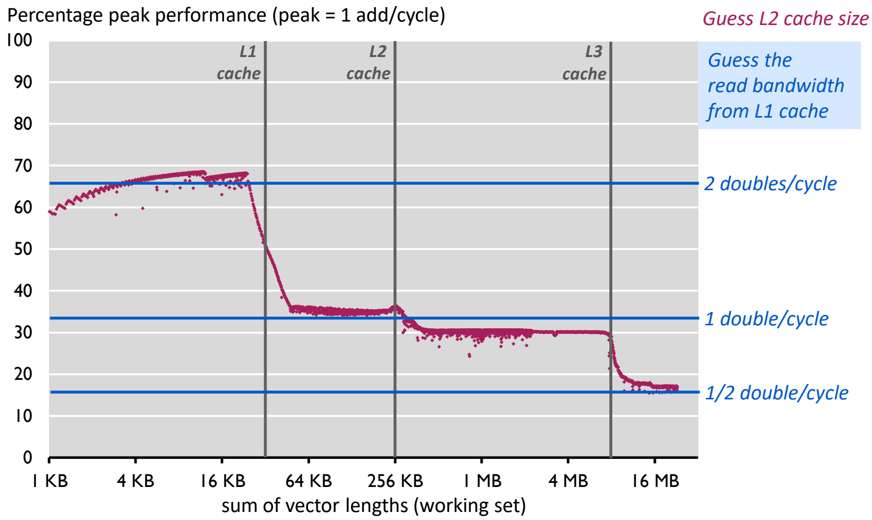
How the measurement is done
- *Cold or warm cache*
- *In which cache data resides*
- *See next slide*



© Markus Püschel
Computer Science

Example: BLAS 1, Warm Data & Code

z = x + y on Core i7 (Nehalem, one core, no SSE), icc 12.0 /O2 /fp:fast /Qipo



Sparse Linear Algebra

Sparse matrix-vector multiplication (MVM)

Sparsity/Bebop/OSKI

References:

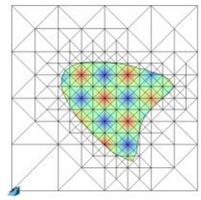
- *Eun-Jin Im, Katherine A. Yelick, Richard Vuduc. SPARSITY: An Optimization Framework for Sparse Matrix Kernels, Int'l Journal of High Performance Comp. App., 18(1), pp. 135-158, 2004*
- *Vuduc, R.; Demmel, J.W.; Yelick, K.A.; Kamil, S.; Nishtala, R.; Lee, B.; Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply, pp. 26, Supercomputing, 2002*
- *Sparsity/Bebop website*

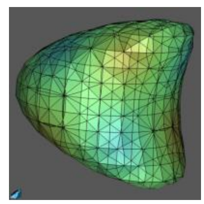
Sparse Linear Algebra

Very different characteristics from dense linear algebra (LAPACK etc.)

Applications:

- *finite element methods*
- *PDE solving*
- *physical/chemical simulation (e.g., fluid dynamics)*
- *linear programming*
- *scheduling*
- *signal processing (e.g., filters)*
- *…*



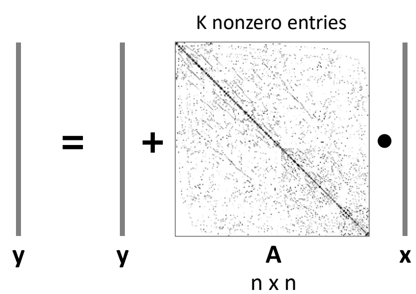


Core building block: Sparse MVM

Graphics: http://aam.mathematik.uni-freiburg.de/IAM/homepages/claus/projects/unfitted-meshes_en.html

Sparse MVM (SMVM)

y = y + Ax, A sparse but known (below A is square)



Typically executed many times for fixed A

What is reused (possible temporal locality)?

Upper bound on operational intensity? $I(n) \le \frac{2K}{8(K+3n)} \le \frac{1}{4}$

© Markus Püschel
Computer Science
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Storage of Sparse Matrices

Standard storage is obviously inefficient: Many zeros are stored

- *Unnecessary operations*
- *Unnecessary data movement*
- *Bad operational intensity*

Several sparse storage formats are available

Popular for performance: Compressed sparse row (CSR) format

CSR

Assumptions:

- *A is m x n*
- *K nonzero entries*

A as matrix

b	c		c
	a		
		b	b
		c	

A in CSR:

`values`	b	c	c	a	b	b	c	*length K*
`col_idx`	0	1	3	1	2	3	2	*length K*

`row_start`	0	3	4	6	7	*length m+1*

Storage:

- *K doubles + (K+m+1) ints = Θ(max(K, m))*
- *Typically: Θ(K)*

Sparse MVM Using CSR

y = y + Ax

```
void smvm(int m, const double* values, const int* col_idx,
          const int* row_start, double* x, double* y)
{
  int i, j;
  double d;

  /* loop over m rows */
  for (i = 0; i < m; i++) {
    d = y[i]; /* scalar replacement since reused */

    /* loop over non-zero elements in row i */
    for (j = row_start[i]; j < row_start[i+1]; j++)
      d += values[j] * x[col_idx[j]];
    y[i] = d;
  }
}
```

CSR + sparse MVM: Advantages?

CSR

Advantages:

- *Only nonzero values are stored*
- *All three arrays for A (*`values`*,* `col_idx`*,* `row_start`*) accessed consecutively in MVM (good spatial locality)*
- *Good temporal locality with respect to y*

Disadvantages:

- *Insertion into A is costly*
- *Poor temporal locality with respect to x*

Impact of Matrix Sparsity on Performance

Adressing overhead (dense MVM vs. dense MVM in CSR):

- *~ 2x slower (example only)*

Fundamental difference between MVM and sparse MVM (SMVM):

- *Sparse MVM is input **dependent** (sparsity pattern of A)*
- *Changing the order of computation (e.g., when blocking) requires changing the data structure (CSR)*

Bebop/Sparsity: SMVM Optimizations

Idea: Blocking for registers

Reason: Reuse x to reduce memory traffic

Execution: Block SMVM y = y + Ax into micro MVMs

- *Block size r x c becomes a parameter*
- *Consequence: Change A from CSR to r x c block-CSR (BCSR)*

BCSR: Next slide

© Markus Püschel
Computer Science
ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

BCSR (Blocks of Size r x c)

Assumptions:

- *A is m x n*
- *Block size r x c*
- *$K_{r,c}$ nonzero blocks*

A as matrix (r = c = 2)

b	c		c
	a		
		b	b
		c	

A in BCSR (r = c = 2):

`b_values`	b c 0 a 0 c 0 0 b b c 0	*length $rcK_{r,c}$*
`b_col_idx`	0 1 1	*length $K_{r,c}$*
`b_row_start`	0 2 3	*length m/r+1*

Storage:

- *$rcK_{r,c}$ doubles + ($K_{r,c}$+m/r+1) ints = $\Theta(rcK_{r,c})$*
- *$rcK_{r,c} \geq K$*

Sparse MVM Using 2 x 2 BCSR

```
void smvm_2x2(int bm, const int *b_row_start, const int *b_col_idx,
              const double *b_values, double *x, double *y)
{
  int i, j;
  double d0, d1, c0, c1;

  /* loop over bm block rows */
  for (i = 0; i < bm; i++) {
    d0 = y[2*i];    /* scalar replacement since reused */
    d1 = y[2*i+1];

    /* dense micro MVM */
    for (j = b_row_start[i]; j < b_row_start[i+1]; j++, b_values += 2*2) {
      c0 = x[2*b_col_idx[j]+0]; /* scalar replacement since reused */
      c1 = x[2*b_col_idx[j]+1];
      d0 += b_values[0] * c0;
      d1 += b_values[2] * c0;
      d0 += b_values[1] * c1;
      d1 += b_values[3] * c1;
    }
    y[2*i]   = d0;
    y[2*i+1] = d1;
  }
}
```


© Markus Püschel
Computer Science
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

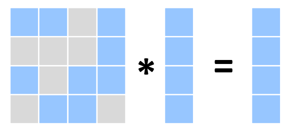
BCSR

Advantages:

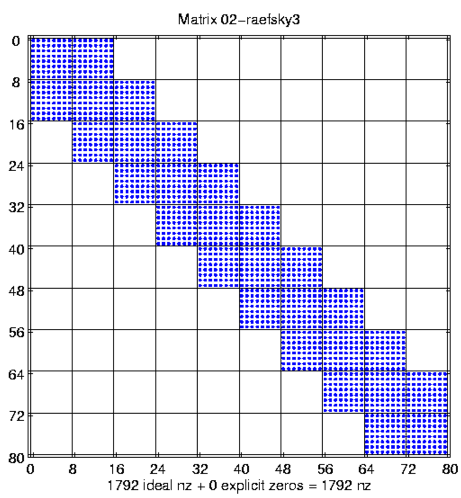
- *Temporal locality with respect to x and y*
- *Reduced storage for indexes*

Disadvantages:

- *Storage for values of A increased (zeros added)*
- *Computational overhead (also due to zeros)*



Which Block Size (r x c) is Optimal?

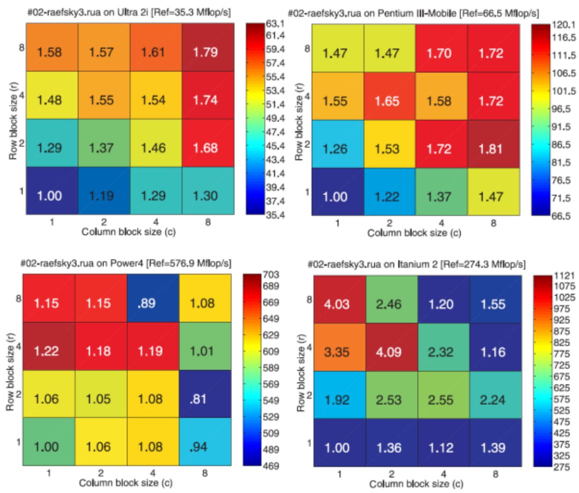


Example:

- 20,000 x 20,000 matrix (only part shown)
- Perfect 8 x 8 block structure
- No overhead when blocked r x c, with r, c divides 8

source: R. Vuduc, Georgia Tech

Speed-up Through r x c Blocking



- *machine dependent*
- *hard to predict*

Source: *Eun-Jin Im, Katherine A. Yelick, Richard Vuduc. SPARSITY: An Optimization Framework for Sparse Matrix Kernels, Int'l Journal of High Performance Comp. App., 18(1), pp. 135-158, 2004*

How to Find the Best Blocking for given A?

Best block size is hard to predict (see previous slide)

Solution 1: Searching over all r x c within a range, e.g., 1 ≤ r,c ≤ 12

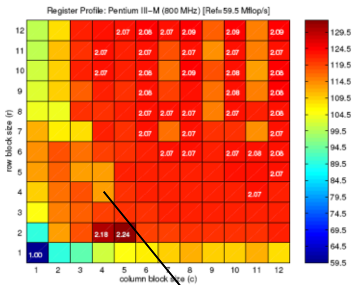
- *Conversion of A in CSR to BCSR roughly as expensive as 10 SMVMs*
- *So total cost = 1440 SMVMs*
- *Too expensive*

Solution 2: Model

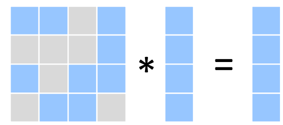
- *Estimate the gain through blocking*
- *Estimate the loss through blocking*
- *Pick best ratio*

Model: Example

Gain by blocking (dense MVM)



Overhead (average) by blocking



16/9 = 1.77

1.4

1.4/1.77 = 0.79 (no gain)

Model: Doing that for all r and c and picking best

Model

Goal: find best r x c for y = y + Ax

Gain through r x c blocking (estimation):

$$G_{r,c} = \frac{\textit{dense MVM performance in r x c BCSR}}{\textit{dense MVM performance in CSR}}$$

dependent on machine, independent of sparse matrix

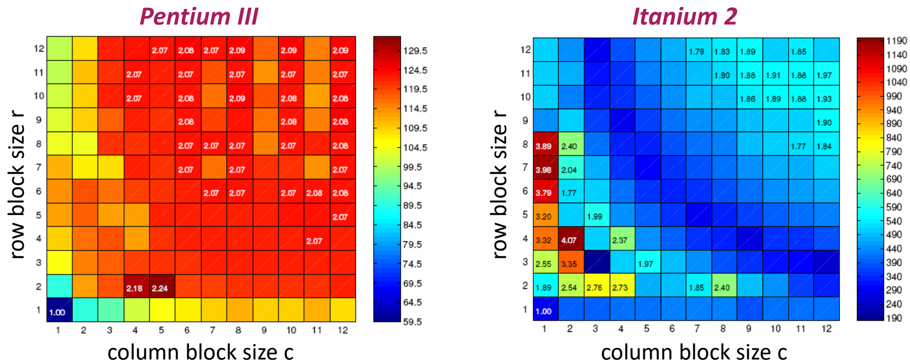
Overhead through r x c blocking (estimation)
scan part of matrix A

$$O_{r,c} = \frac{\textit{number of matrix values in r x c BCSR}}{\textit{number of matrix values in CSR}}$$

independent of machine, dependent on sparse matrix

Expected gain: $G_{r,c}/O_{r,c}$

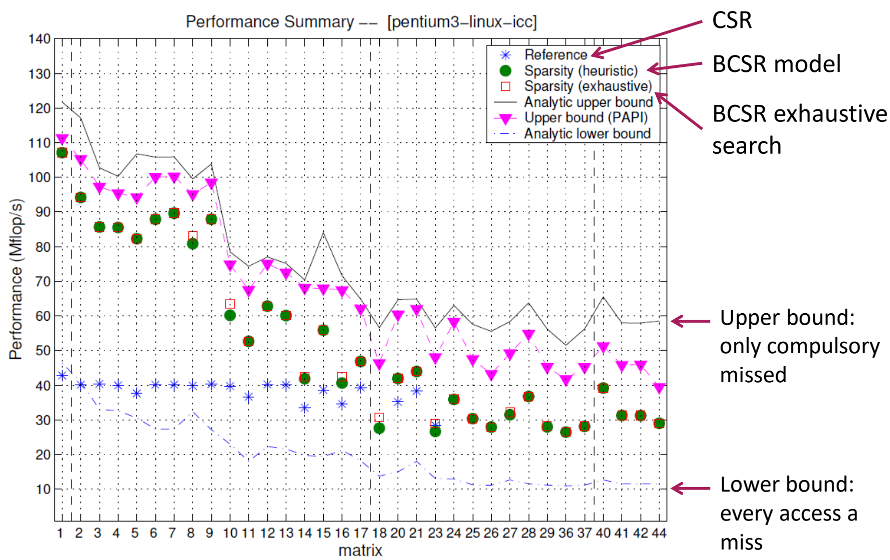
Gain from Blocking (Dense Matrix in BCSR)



- *machine dependent*
- *hard to predict*

Source: Eun-Jin Im, Katherine A. Yelick, Richard Vuduc. *SPARSITY: An Optimization Framework for Sparse Matrix Kernels*, *Int'l Journal of High Performance Comp. App.*, 18(1), pp. 135-158, 2004

Typical Result (assumes cold cache)



Source: Eun-Jin Im, Katherine A. Yelick, Richard Vuduc. *SPARSITY: An Optimization Framework for Sparse Matrix Kernels*, *Int'l Journal of High Performance Comp. App.*, 18(1), pp. 135-158, 2004

Principles in Bebop/Sparsity Optimization

Optimization for memory hierarchy = increasing locality

- *Blocking for registers (micro-MVMs)*
- *Requires change of data structure for A*
- *Optimizations are input dependent (on sparse structure of A)*

Fast basic blocks for small sizes (micro-MVM):

- *Unrolling + scalar replacement*

Search for the fastest over a relevant set of algorithm/implementation alternatives (parameters r, c)

- *Use of performance model (versus measuring runtime) to evaluate expected gain*

Different from ATLAS

SMVM: Other Ideas

Value compression

Index compression

Pattern-based compression

Multiple inputs

© Markus Püschel
Computer Science
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Value Compression

Situation: Matrix A contains many duplicate values

Idea: Store only unique ones plus index information

b	c		c
	a		
		b	b
		c	

A in CSR:

`values`	b	c	c	a	b	b	c
`col_idx`	0	1	3	1	2	3	2
`row_start`	0	3	4	6	7		

A in CSR-VI:

	a	b	c

`values`	1	2	2	0	1	1	2
`col_idx`	0	1	3	1	2	3	2
`row_start`	0	3	4	6	7		

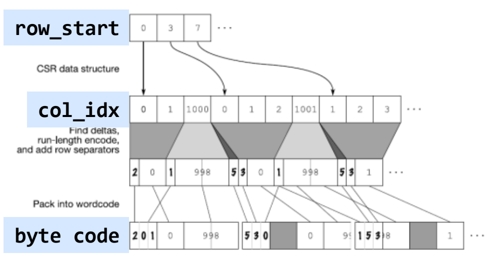
Kourtis, Goumas, and Koziris, Improving the Performance of Multithreaded Sparse Matrix-Vector Multiplication using Index and Value Compression, pp. 511-519, ICPP 2008

Index Compression

Situation: Matrix A contains sequences of nonzero entries

Idea: Use special byte code to jointly compress `col_idx` and `row_start`

Coding



Decoding

```
0: acc = acc * 256 + arg;
1: col = col + acc * 256 + arg; acc = 0;
   emit_element(row, col); col = col + 1;
2: col = col + acc * 256 + arg; acc = 0;
   emit_element(row, col);
   emit_element(row, col + 1); col = col + 2;
3: col = col + acc * 256 + arg; acc = 0;
   emit_element(row, col);
   emit_element(row, col + 1);
   emit_element(row, col + 2); col = col + 3;
4: col = col + acc * 256 + arg; acc = 0;
   emit_element(row, col);
   emit_element(row, col + 1);
   emit_element(row, col + 2);
   emit_element(row, col + 3); col = col + 4;
5: row = row + 1; col = 0;
```

Source: *Willcock and Lumsdaine, Accelerating Sparse Matrix Computations via Data Compression, pp. 307-316, ICS 2006*

© Markus Püschel
Computer Science
ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Pattern-Based Compression

Situation: After blocking A, many blocks have the same nonzero pattern

Idea: Use special BCSR format to avoid storing zeros;
needs specialized micro-MVM kernel for each pattern

A as matrix

b	c		c
	a		
		b	b
		c	

Values in 2 x 2 BCSR

b	c	0	a	0	c	0	0	b	b	c	0

Values in 2 x 2 PBR

b	c	a	c	b	b	c

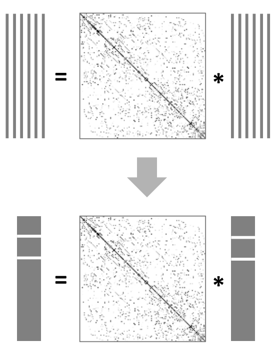
+ bit string: 1101 0100 1110

Source: *Belgin, Back, and Ribbens, Pattern-based Sparse Matrix Representation for Memory-Efficient SMVM Kernels, pp. 100-109, ICS 2009*

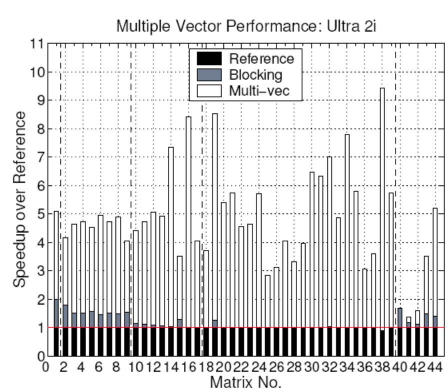
Multiple Inputs

Situation: Compute SMVM y = y + Ax for several independent x

Experiments: up to 9x speedup for 9 vectors



enables blocking across MVMs like MMM



Source: Eun-Jin Im, Katherine A. Yelick, Richard Vuduc. *SPARSITY: An Optimization Framework for Sparse Matrix Kernels*, *Int'l Journal of High Performance Comp. App.*, 18(1), pp. 135-158, 2004

© Markus Püschel
Computer Science