Introduction to Simple Performance Modeling

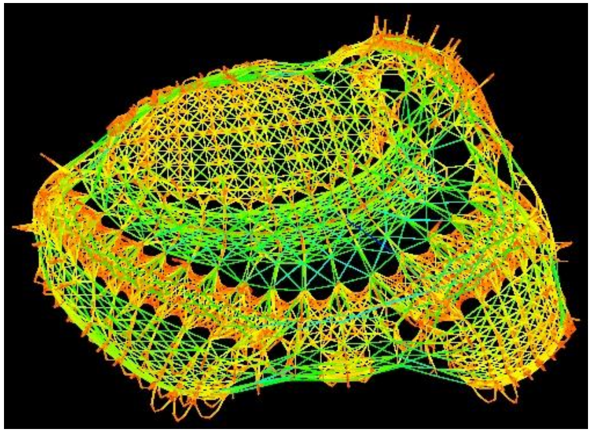
Tim Cramer
RWTH Aachen University
cramer@rz.rwth-aachen.de

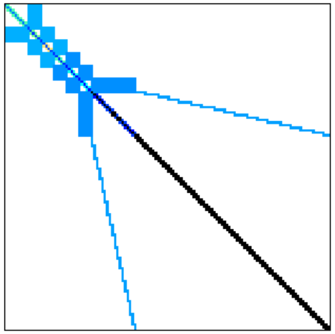
February 10th 2014

Case Study: CG Solver

- **Sparse Linear Algebra**
  - → Sparse Linear Equation Systems occur in many scientific disciplines
  - → Sparse matrix-vector multiplications (SpMxV) are the dominant part in many iterative solvers (like the CG) for such systems
  - → #non-zero elements << n*n

`n`: matrix dimension
`nnz`: #non-zeros





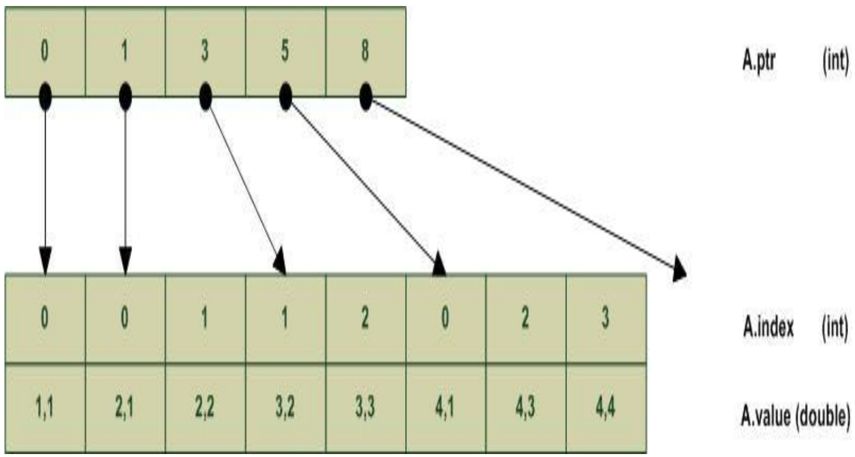
Beijing Botanical Garden

Top right:	Orginal building
Bottom right:	Model
Bottom left:	Matrix

(Source: Beijing Botanical Garden and University of Florida, Sparse Matrix Collection)

Case Study: CG Solver

- **Compressed Row Storage (CRS) format**
  - → only non-zero values are stored (nnz)
  - → *Given* $A\epsilon\ \mathbb{R}^{4x4}$
  - → $A = \begin{pmatrix} 1,1 & 0 & 0 & 0 \\ 2,1 & 2,2 & 0 & 0 \\ 0 & 3,2 & 3,3 & 0 \\ 4,1 & 0 & 4,3 & 4,4 \end{pmatrix}$



- **CG iterative solver**
  - → matvec: $\vec{y} = A \cdot \vec{x}$ (SpMV)
  - → xpay: $\vec{y} = \vec{x} + \alpha \cdot \vec{y}$
  - → axpy: $\vec{y} = \alpha \cdot \vec{x} + \vec{y}$
  - → vectorDot: $c = \vec{a} \cdot \vec{b}$

- **Where to start parallelizing? Where is a performance problem?**
  - → Hotspot analyses & more

Hotspot

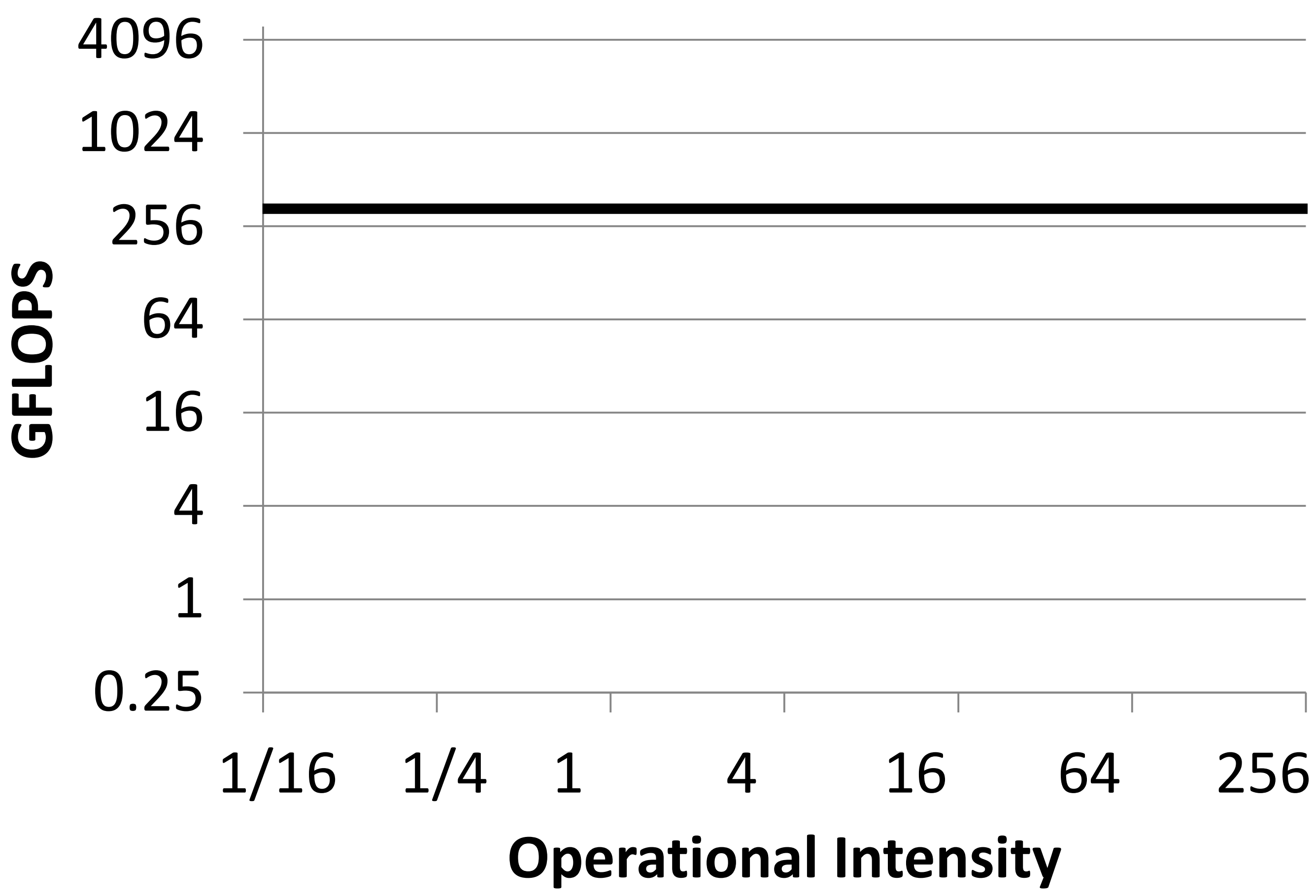
- **Determine the Hotspot**
  - → Use profiling tools (e.g., VI-HPS tools)
  - → Manual measurements of code fragments
- **Testcase**
  - → Fluorem/HV15R
  - → N=2,017,169, nnz=283,073,458
  - → 3.2 GB Memory footprint
  - → I = 1000 Iterations
- **Runtime Shares of the Linear Algebra Kernels**
  - → Used System: Two Intel SandyBridge SNB processors @ 2.6 GHz

System	#Threads	Serial Time [s]	Parallel Time [s]	daxpy / dxpay	dot product	SpMV
2 x SNB	16	340.62	112.12	2.3 %	1.0 %	96.7 %

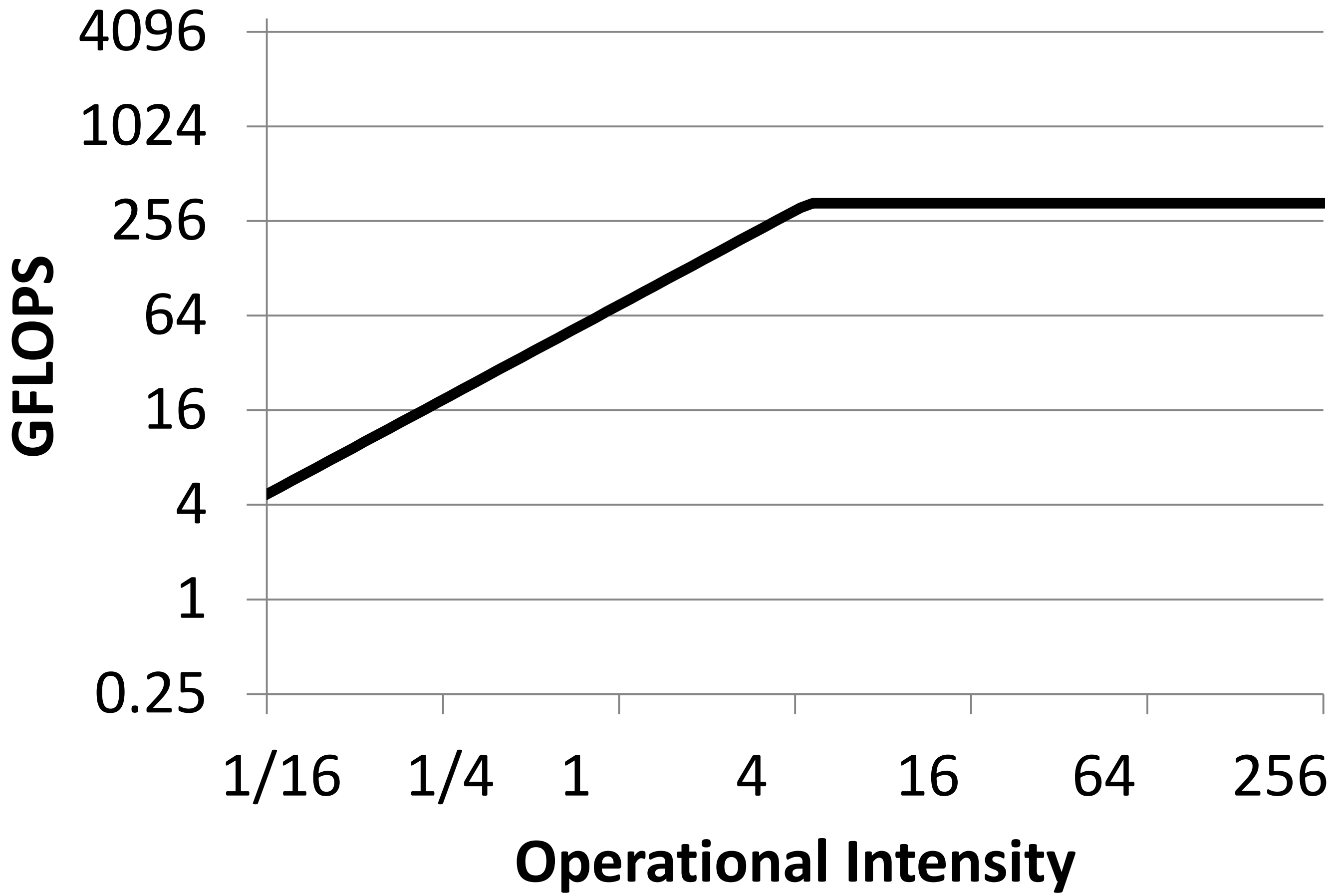
  - → SpMV is the most dominant operation
  - → Delivers the hotspot a reasonable performance?
    - → Considering only the time of 108.5 s (= 0.967 * 112.12s) is not meaningful
    - → Better metric: GFLOPS, here 5.22 GFLOPS for SpMV (= 2 * I * nnz * 10^-9 / 108.5 s)

Roofline Model

- **Peak performance of two Intel SandyBridge SNB processors (2.6 GHz) is 333 GFLOPS (2.6 GHz * 8 OPs/cycle * 16 cores)**

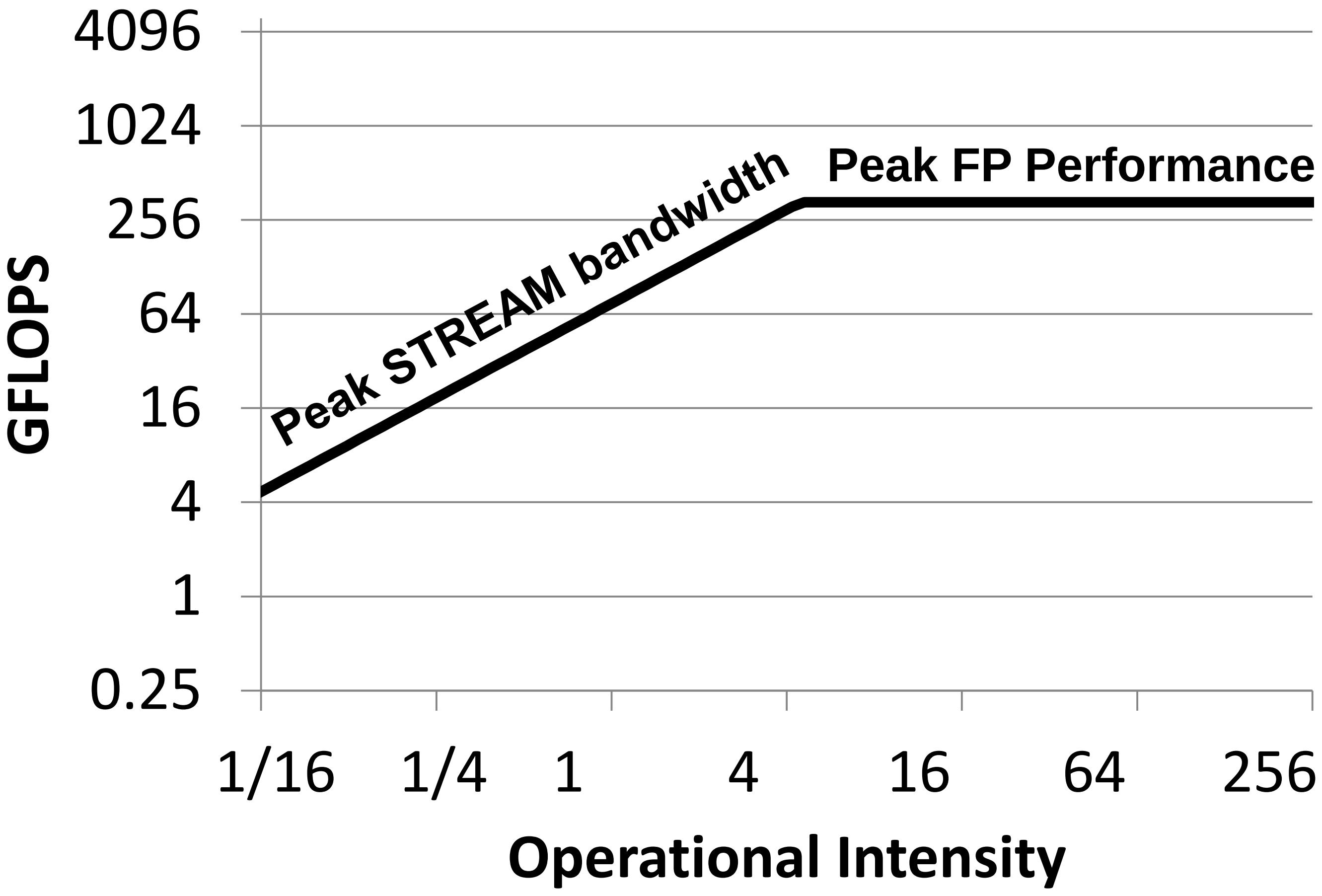


- **Memory bandwidth measured with the STREAM benchmark is about 75 GB/s (Triade: $\vec{a} = \vec{b} + \alpha * \vec{c}$)**

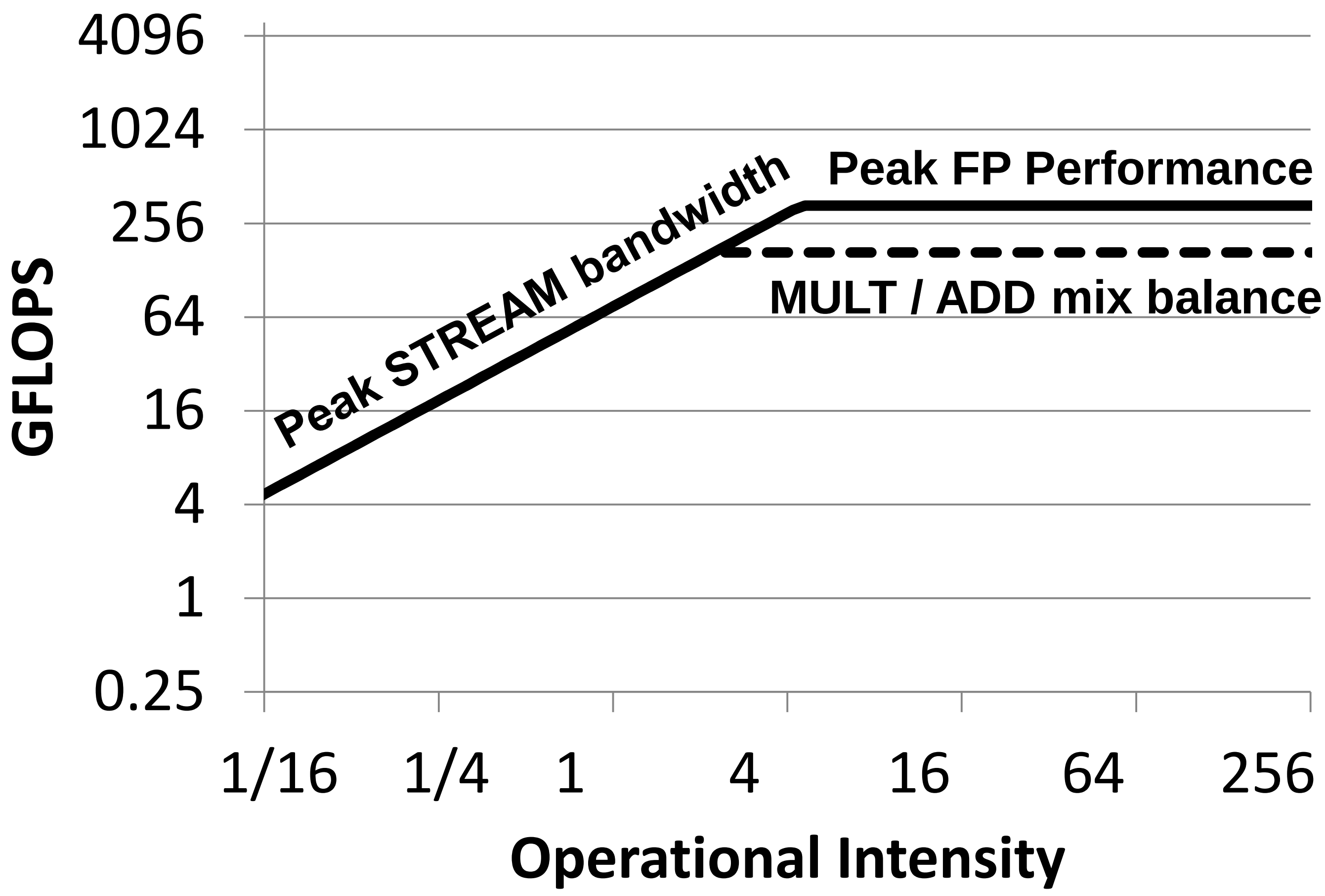


Roofline Model

- The “Roofline” is the peak perfomance depending on the algorithms’s “operational intensity”.



- **To reach the peak performance an even mix of multiply and add operations is need ("fused multiply add")**



SIMD / AVX Arithmetic

Basic Arithmetic on Intel SNB

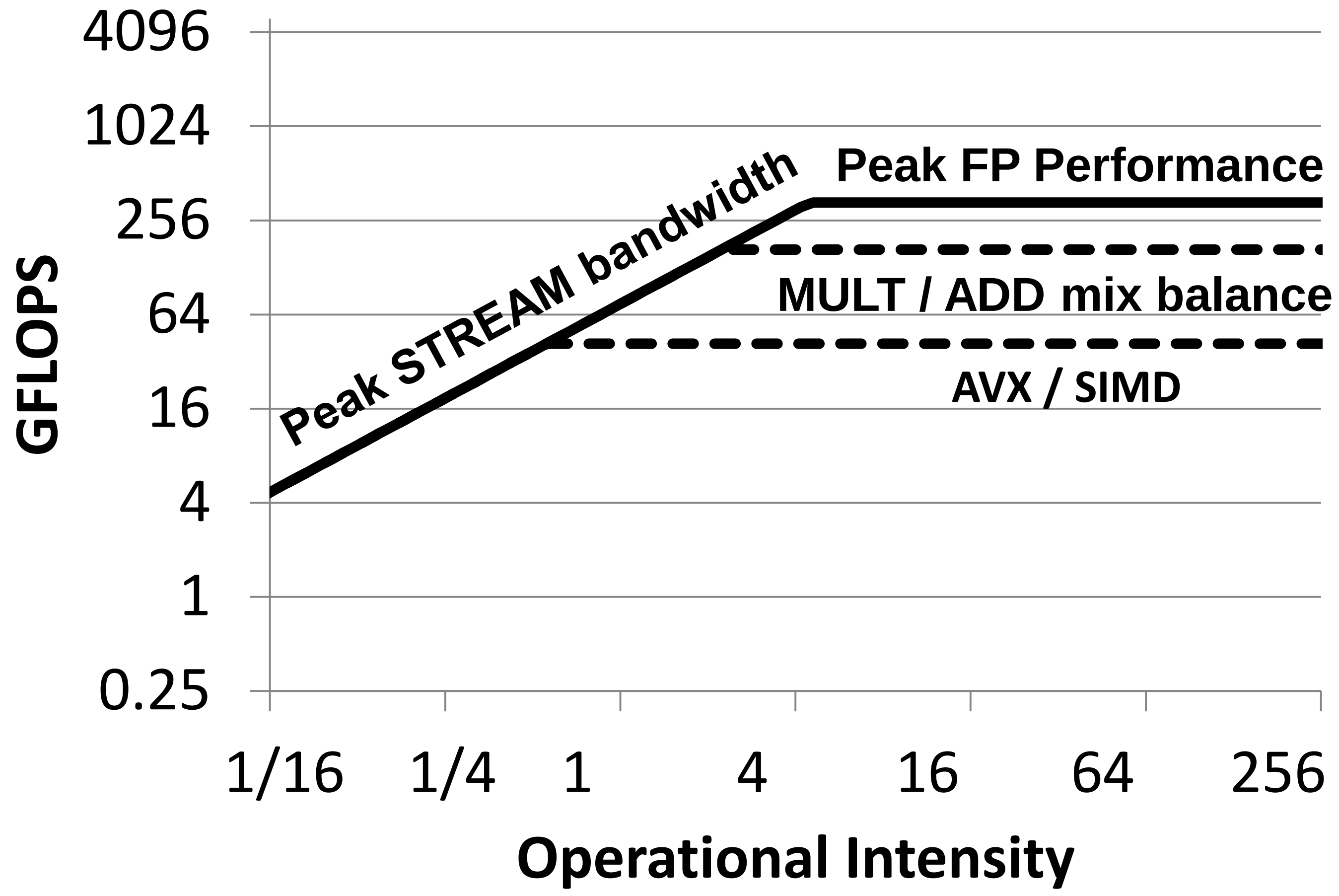
source 1	a3	a2	a1	a0	4 x DP	
		+				
source 2	b3	b2	b1	b0	4 x DP	
		=				
destination	a3+b3	a2+b2	a1+b1	a0+b0	4 x DP	4 results per cycle

Fused Multiply Add on Intel SNB

source 1	a3	a2	a1	a0	4 x DP	
		*				
source 2	b3	b2	b1	b0	4 x DP	
		+				
source 3	c3	c2	c1	c0	4 x DP	
		=				
destination	a3*b3+c3	a2*b2+c2	a1*b1+c1	a0*b0+c0	4 x DP	8 results per cycle

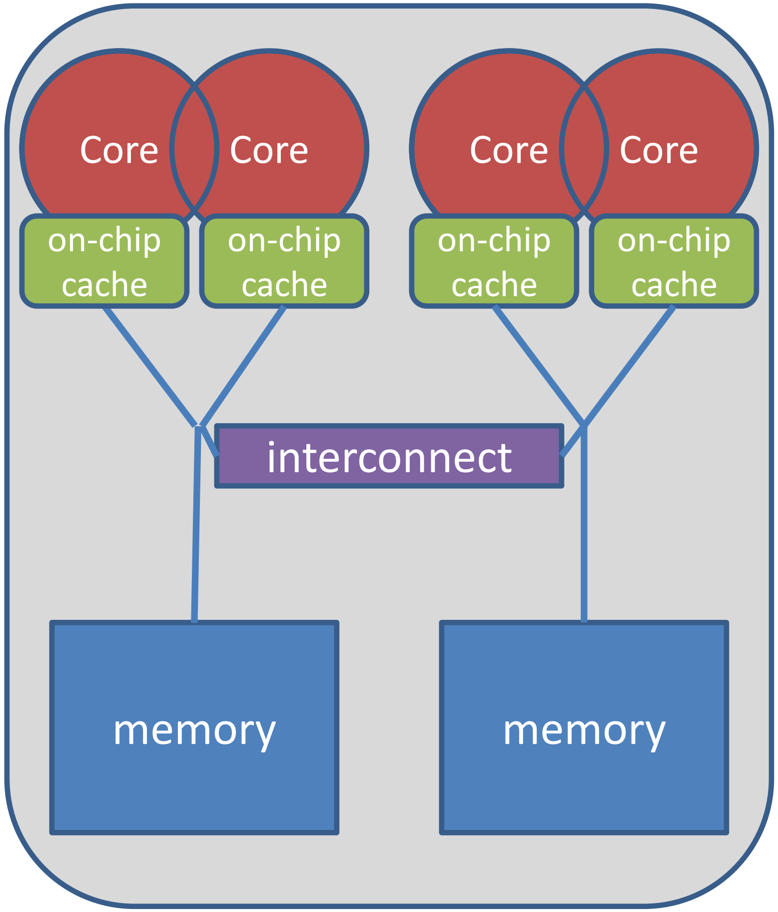
256 bit

- **Without AVX / SIMD vectorization only 1/8 of the peak performance is achievable**

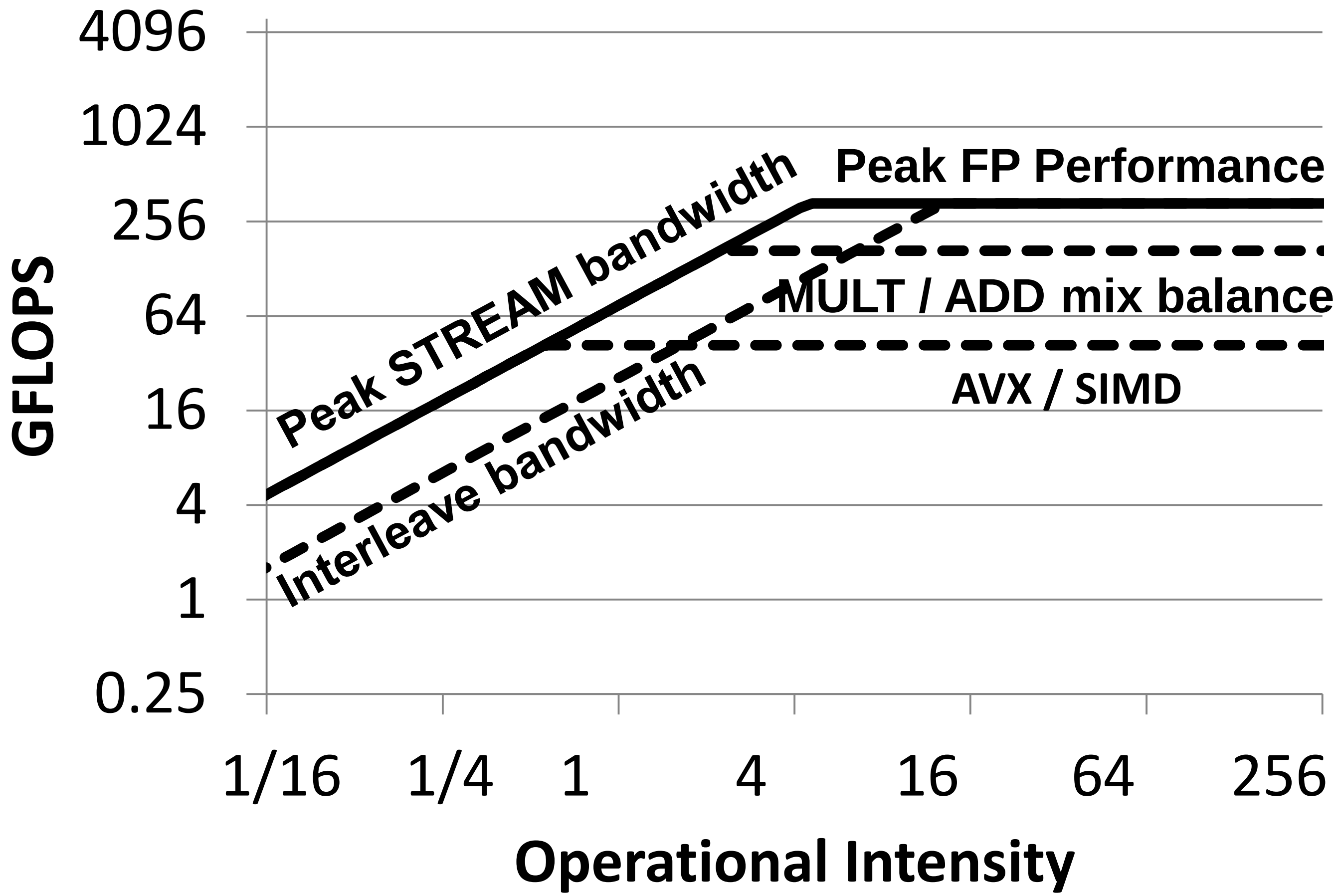


Non Uniform Memory Access

- → Most machines are multiple socket machines
- → Latency and memory bandwidth depend on which core accesses the memory
- → Linux uses a first touch policy for the memory placement



- **Memory controller can only be saturated if the memory placement is correct (ccNUMA, first touch)**



Implementation of the Hotspot (SpMV)

- **Sparse Matrix Vector Multiplication (SpMV)**

```
# pragma omp parallel private(i,j,is,ie,j0,y0, thread, bs, be)
thread = omp_get_thread_num();
bs = A->blockptr[thread];
be = A->blockptr[thread + 1];
for (i = bs; i < be; i++) {
    y0 = 0;
    is = A->ptr[i];
    ie = A->ptr[i + 1];
    for (j = is; j < ie; j++) {
        j0 = index[j];
        y0 += value[j] * x[j0];
    }
    y[i] = y0
}
```

length nnz, too big for cache

length n, fits into cache

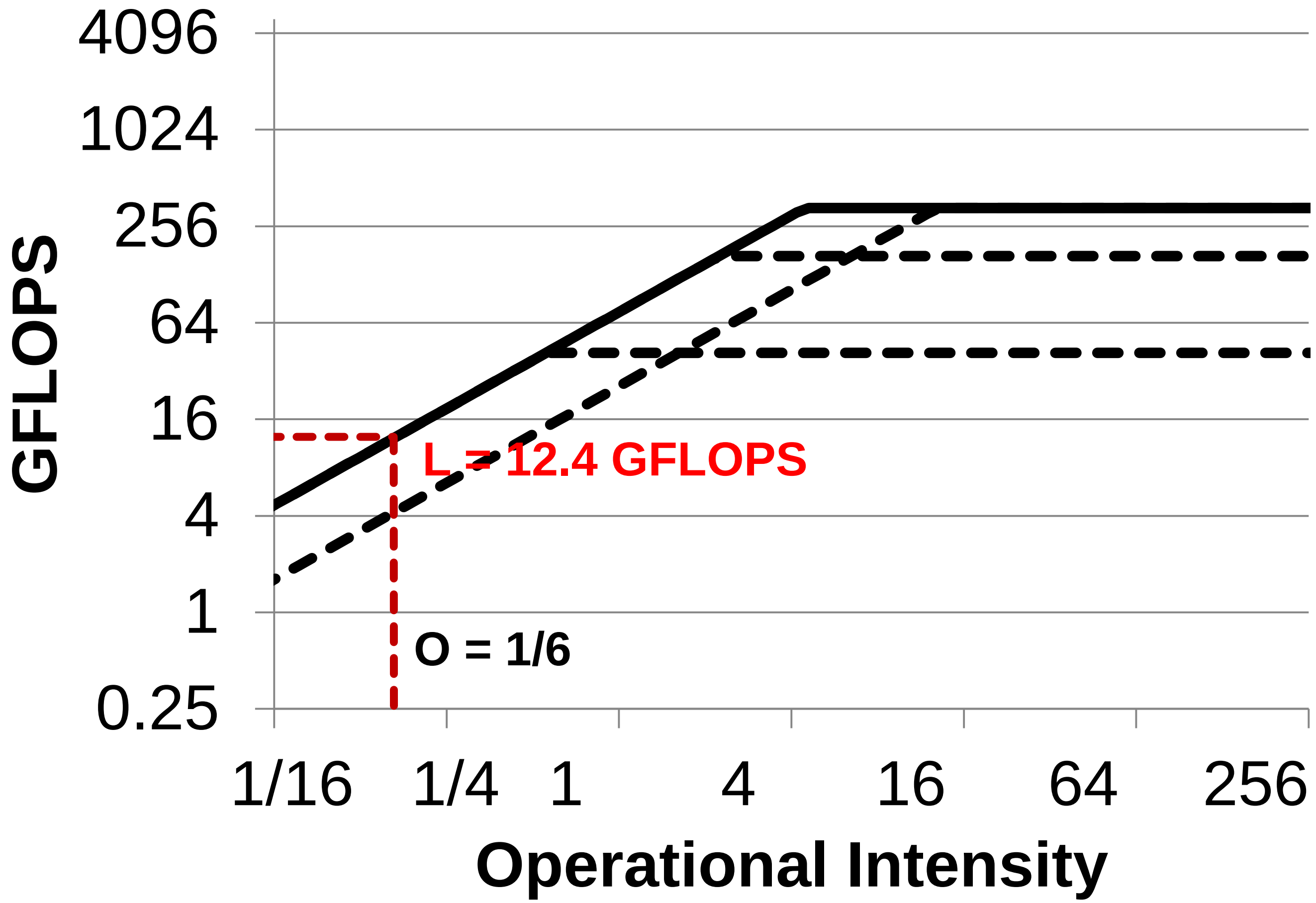
2 Floating Point Ops

Roofline Model: SpMV

- **Roofline Model**
  - → Using **memory bandwidth** BW and **theoretical peak** performance P
- **Model for SMXV** $\vec{y} = A * \vec{x}$
  - → Assumptions
    - → $\vec{x}, \vec{y}$ can be kept in the cache (~ 15 MB)
    - → A too big for caches (~ 3200 MB)
    - → $n \ll nnz$
    - → Compressed Row Storage (CRS) Format: One value (`double`) and one index (`int`) element have to be loaded (dimension nnz) → 12 Bytes
  - → **Operational intensity** $\mathbf{O} = \frac{2\ FLOPS}{12\ Byte} = \frac{1}{6} \frac{FLOPS}{Byte}$ (→ memory-bound)
  - → **Performance Limit:** $L = \min\{P, O * BW\}$

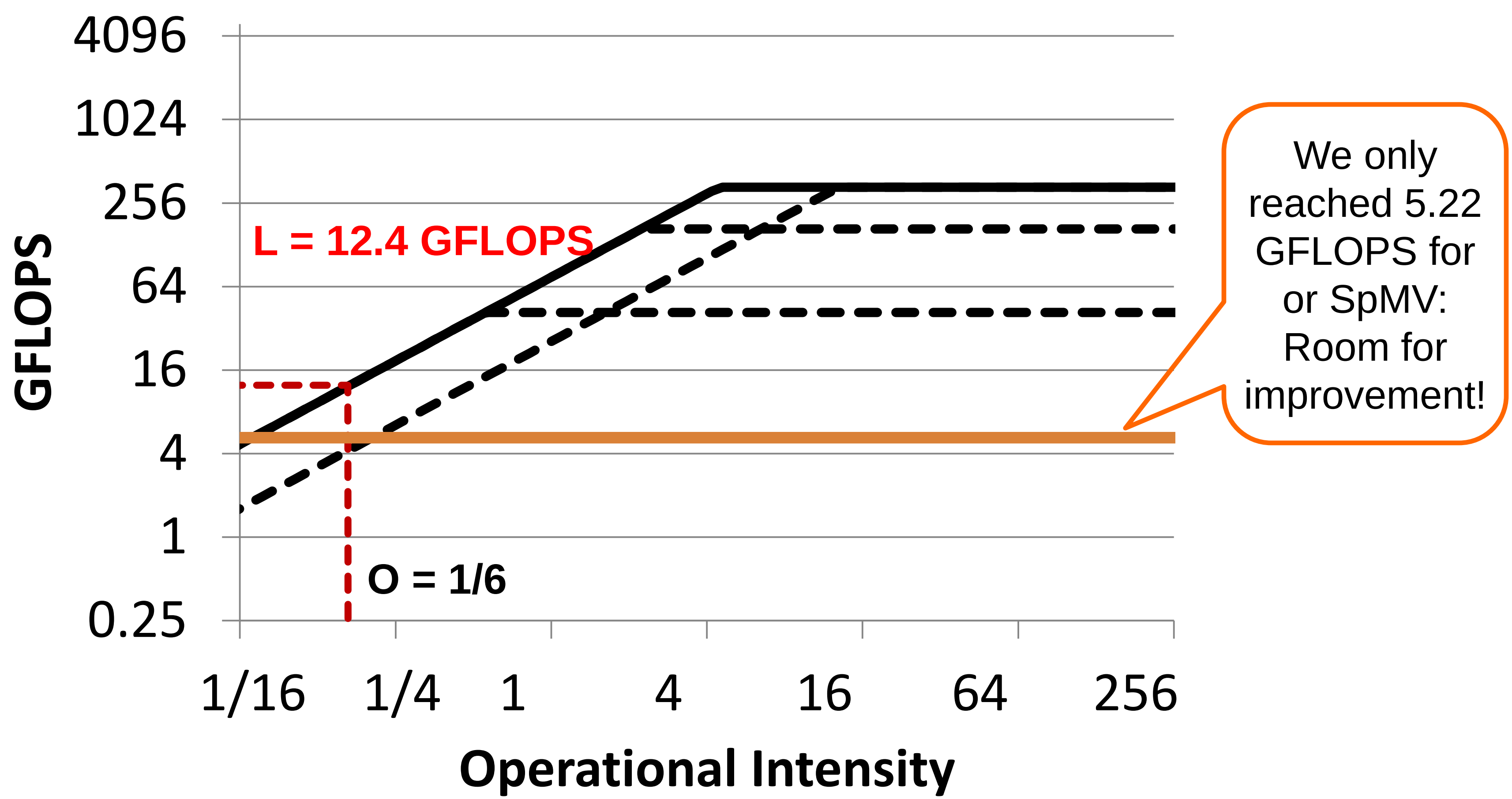
Roofline Model for SNB

- **Roofline Model 2 x SNB (2.6 GHz, STREAM 74.2 GB/s, Peak 332.8 GFLOPS)**



Reached Performance

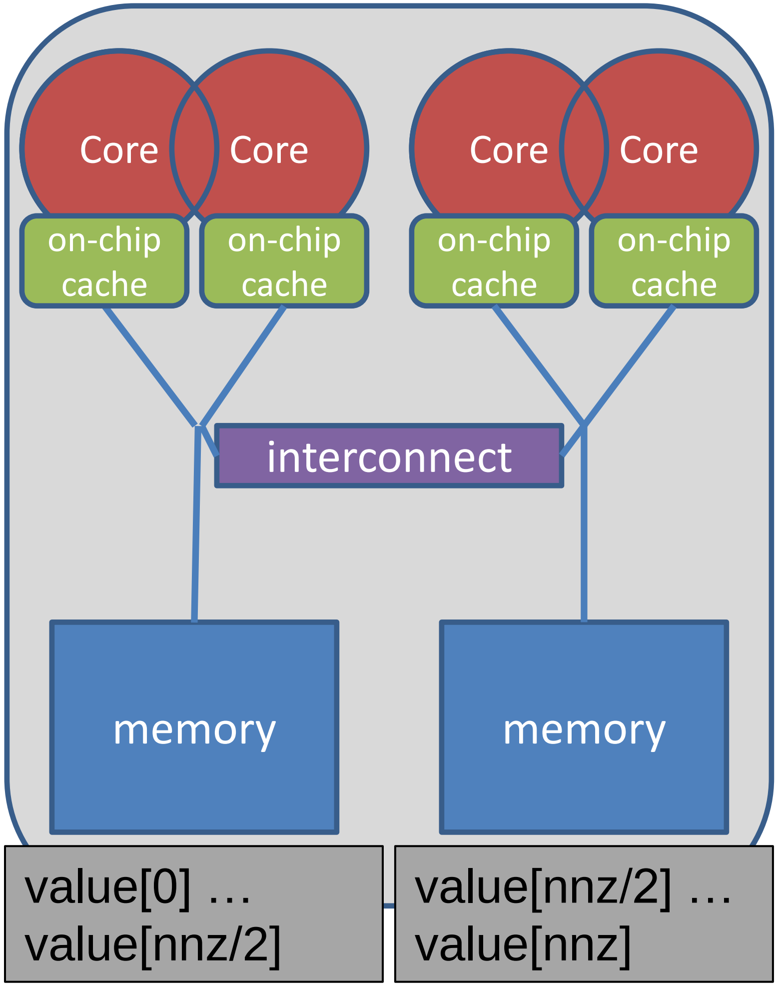
- **Roofline Model 2 x SNB (2.6 GHz, STREAM 74.2 GB/s, Peak 332.8 GFLOPS)**



First Touch Memory Placement

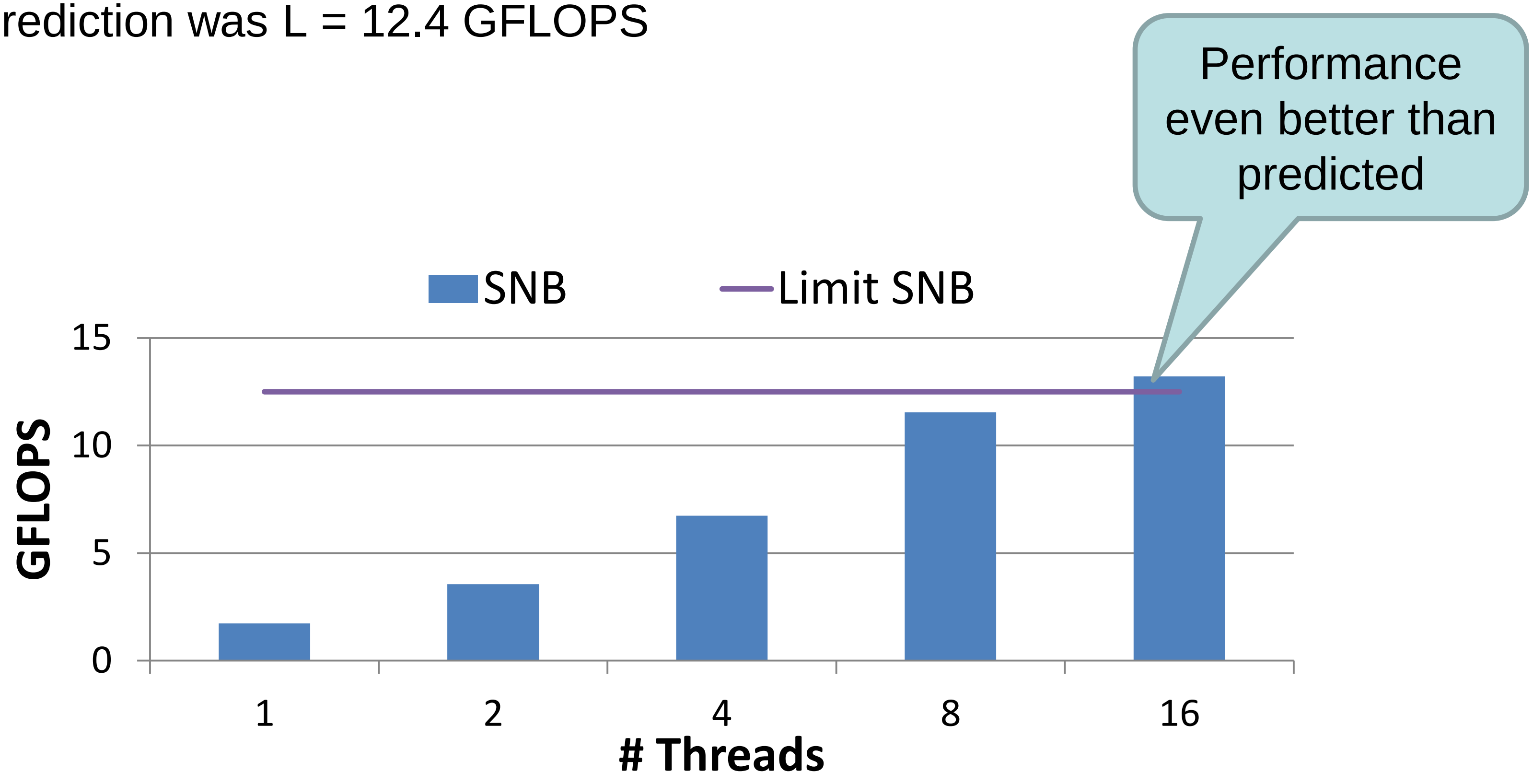
- **First Touch w/ parallel OpenMP code**
  - → All array elements are allocated in the memory of the NUMA node containing the core executing the thread initializing the respective partition

```
double* value;
value = (double*)
   malloc(nnz * sizeof(double));

omp_set_num_threads(16);

#pragma omp parallel for
for (int i = 0; i < nnz; i++) {
   value[i] = 0.0;
}
```



SpMV Results

- **After fixing the performance issue we reach 13.1 GFLOPS**
  - → Prediction was L = 12.4 GFLOPS



- **Model not perfect, but you can get an idea of the order of magnitude**

Thank you for your attention.