Assignment 3 – Task 1: Dense MVM

(scalar product style)

- Locality of access (`a[][]` in memory):

```
for(i=0; i<N; ++i)
  for(j=0; j<N; ++j)
    c[i] += a[i][j] * b[j];
```

`a[][]` purely spatial locality (each element accessed exactly once in linear order)

`c[]` spatial and temporal locality: all elements accessed in linear order, each element reused N-1 times

`b[]` spatial locality: all elements accessed in linear order
temporal locality: N-1 times reuse if array b fits into some cache

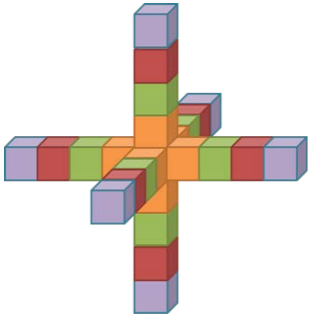
Assignment 3 – Task 1: Dense MVM

```
for(i=0; i<N; ++i)
  for(j=0; j<N; ++j)
    c[i] += a[i][j] * b[j];
```

- dMVM Code balance: Assume c[i] is in a register, a[ ][ ] is in memory

B_C in / if **b[ ]** fits into	**L1**	**L2**	**L3**	**Memory**
L1	8 B/F	4 B/F	4 B/F	4 B/F
L2	8 B/F	8 B/F	4 B/F	4 B/F
L3	8 B/F	8 B/F	8 B/F	4 B/F
Memory	8 B/F	8 B/F	8 B/F	8 B/F

Assignment 3 – Task 2: Long-range stencil P_{max}



```
for(int k=4; k < N-4; k++) {
 for(int j=4; j < N-4; j++) {
  for(int i=4; i < N-4; i++) {
   U[k][j][i] = 2*V[k][j][i] - U[k][j][i] + C[k][j][i] *
   ( c0* V[k][j][i] +
     c1*(V[ k ][ j ][i+1]+V[ k ][ j ][i-1] +
         V[ k ][j+1][ i ]+V[ k ][j-1][ i ] +
         V[k+1][ j ][ i ]+V[k-1][ j ][ i ])+
     c2*(V[ k ][ j ][i+2]+V[ k ][ j ][i-2] +
         V[ k ][j+2][ i ]+V[ k ][j-2][ i ] +
         V[k+2][ j ][ i ]+V[k-2][ j ][ i ])+
     c3*(V[ k ][ j ][i+3]+V[ k ][ j ][i-3] +
         V[ k ][j+3][ i ]+V[ k ][j-3][ i ] +
         V[k+3][ j ][ i ]+V[k-3][ j ][ i ])+
     c4*(V[ k ][ j ][i+4]+V[ k ][ j ][i-4] +
         V[ k ][j+4][ i ]+V[ k ][j-4][ i ] +
         V[k+4][ j ][ i ]+V[k-4][ j ][ i ])
   );
  }
 }
}
```

LOAD:	25+1+1
STORE:	1
ADD/SUB:	26
MULT:	7

The hardware

Microarchitecture	SandyBridge-EP	IvyBridge-EP	Haswell-EP
Shorthand	SNB	IVB	HSW
Xeon Model	E5-2680	E5-2690 v2	E5-2695 v3
Year	03/2012	09/2013	09/2014
Clock speed (fixed)	2.7 GHz	2.2 GHz	2.3 GHz
Cores/Threads	8/16	10/20	14/28
Load/Store throughput per cycle			
AVX(2)	1 LD & 1/2 ST	1 LD & 1/2 ST	2 LD & 1 ST
SSE/scalar	2 LD ‖ 1 LD & 1 ST	2 LD ‖ 1 LD & 1 ST	2 LD & 1 ST
L1 port width	2×16+1×16 B	2×16+1×16 B	2×32+1×32 B
ADD throughput	1 / cy	1 / cy	1 / cy
MUL throughput	1 / cy	1 / cy	2 / cy
FMA throughput	n/a	n/a	2 / cy
L2-L1 data bus	32 B	32 B	64 B
L3-L2 data bus	32 B	32 B	32 B
LLC size	20 MiB	25 MiB	35 MiB
Main memory	4×DDR3-1600	4×DDR3-1866	4×DDR4-2133
Peak memory BW	51.2 GB/s	51.2 GB/s	68.3 GB/s
Load-only BW	43.6 GB/s (85%)	46.1 GB/s (90%)	60.6 GB/s (89%)
T_{L3Mem} per CL	3.96 cy	3.05 cy	2.43 cy

Assignment 3 – Task 2: Long-range stencil P_{max}

LOAD:	25+1+1
STORE:	1
ADD/SUB:	26
MULT:	7

Cycles per scalar iteration (33 Flops):

		Scalar [cy/it]	SSE [cy/it]	AVX [cy/it]
Ivy Bridge	LD/ST ADD MULT Overall bottleneck	13+1 26 7 **ADD (26)**	(13+1)/2 26/2 7/2 **ADD (13)**	27/4 26/4 7/4 **LD (6.75)**
Haswell (no FMA)	LD/ST ADD MULT Overall bottleneck	13.5 26 3.5 **ADD (26)**	13.5/2 26/2 3.5/2 **ADD (13)**	13.5/4 26/4 3.5/4 **ADD (6.5)**
Haswell (MULT/ADD as FMA)	LD/ST ADD or MULT Overall bottleneck	13.5 33/2 **FMA (16.5)**	13.5/2 33/4 **FMA (8.25)**	13.5/4 33/8 **FMA (4.125)**

Assignment 3 – Task 3
Code balance gymnastics

Loop over **i** implied:

a) `s = s + a[i];` (DP) sum reduction

$$B_c = 8\ \mathrm{B/F}$$

b) `s = s + a[i]*b[i];` (DP) scalar product

$$B_c = 8\ \mathrm{B/F}$$

c) `a[i] = b[i] + s*c[i];` (SP) STREAM triad

$$B_c = 8\ \mathrm{B/F}$$

d) `a[i] = a[i] + s*c[i];` (SP) SAXPY

$$B_c = 6\ \mathrm{B/F}$$

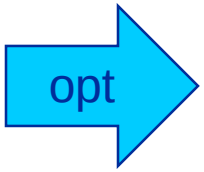
Assignment 3 – Task 3
Code balance gymnastics

e) **`y[i] = y[i] + x[i] * v[index[i]];`**
(DP for **`x[]`**,**`y[]`**, and **`v[]`**, and int32 for **`index[]`**)

- Best case:
  `index[i] = k` → $B_c = \frac{24+4}{2}\,\mathrm{B/F} = 14\ \mathrm{B/F}$
- Linear, stride-1 index:
  `index[i] = k+i` → $B_c = \frac{24+8+4}{2}\,\mathrm{B/F} = 18\ \mathrm{B/F}$
- Worst case: random, no CL reuse (CL length = L_c bytes):
  `index[i] = rand()` → $B_c = \frac{24+L_c+4}{2}\frac{B}{F} = \left(14 + \frac{L_c}{2}\right)\frac{B}{F}$
- General formula: $B_c = (14 + 4\alpha)\frac{B}{F}$, with $0 \le \alpha \le \frac{L_c}{8}$

Assignment 3 – Task 4
Loop optimization and modeling

```
for(i=0; i<N; ++i) {
  a[i] = b[i] * c[i];
}
for(i=0; i<N-1; ++i) {
  b[i] = a[i+1] * 0.5;
}
```




```
for(i=0; i<N-1; ++i) {
  a[i] = b[i] * c[i];
  b[i] = b[i+1] * c[i+1] * 0.5;
}
a[N-1] = b[N-1] * c[N-1];
```

$$B_c = \frac{24 + 24 + 8}{2}\frac{B}{F} = 28\ \mathrm{B/F}$$

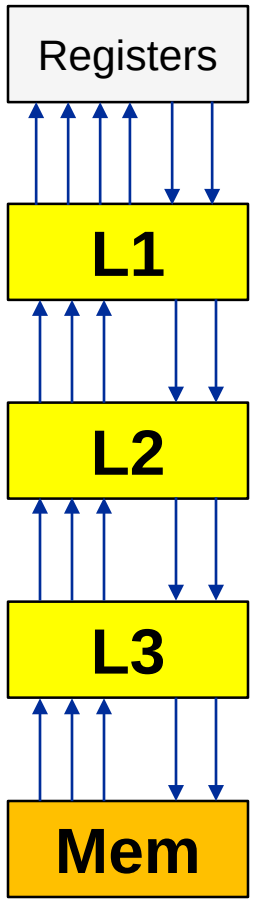
$$B_c = \frac{16 + 16 + 8}{3}\frac{B}{F} \approx 13.3\ \mathrm{B/F}$$

- Is the code SIMD vectorizable?
  - Original: yes (trivially)
  - Optimized: yes, because the forward reference to `b[i+1]` is harmless
- This is another example where we clearly see that a flop might not be a good metric for work
  - 56 B/it vs. 40 B/it would be a more fair comparison

Assignment 3 – Task 4
Loop optimization and modeling

b) Execution & data transfer modeling for optimized code on HSW per CL (AVX)

```
for(i=0; i<N-1; ++i) {
  a[i] = b[i] * c[i];
  b[i] = b[i+1] * c[i+1] * 0.5;
}
```



- In-core instructions per 8 scalar iterations: **8 LD**, **4 ST**, 6 MULT → **4 cy**
- L2: 5 CLs → 5 cy
- L3: 5 CLs → 10 cy
- Memory: 5 CLs * 4.6 cy/CL = 23 cy

Level	Transfer time to higher level [cy]	Total transfer time [cy]
L1	4	**4**
L2	5 cy	**9 cy**
L3	10 cy	**19 cy**
Memory	23 cy	**42 cy**

- Performance: $P = \frac{3 \cdot 8\ flops}{42\ cy} \cdot 2.3 \frac{Gcy}{s} = 1.3 \frac{Gflops}{s}$
- Memory bandwidth: $b = P \cdot B_c = 17.3\ \mathrm{GB/s}$