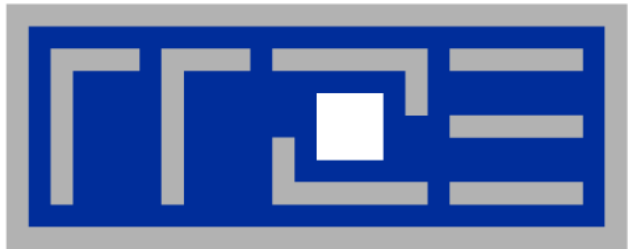


Performance Modeling of Stencil Codes

Holger Stengel
Erlangen Regional Computing Center (RRZE)
University of Erlangen-Nuremberg

ParCo Mini-Symposium: Performance Modeling and Engineering
September 13, 2013
Garching, Germany

Outline

- **Motivation**
- **ECM Performance Model**
- **Stencil Examples**
  - 2D Jacobi
  - 3D 25-point long-range
- **Conclusions**

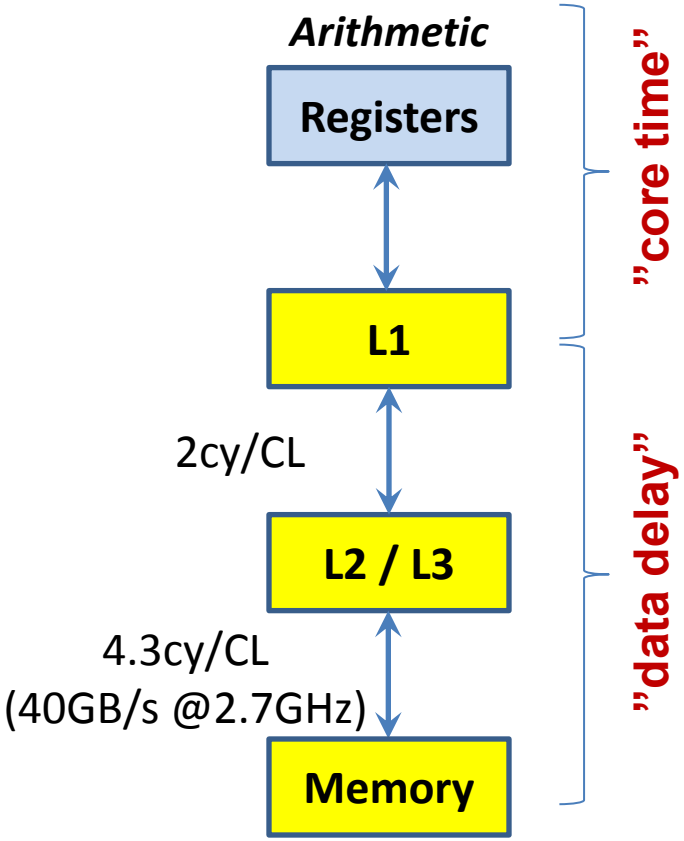
Why performance modeling?

- **Common goal: Improve code performance to save resources**

- **Setting up a performance model helps to**
  - Gain insight into hardware / software interaction
  - Predict performance behavior
  - Unveil performance limitations
  - Perform purposeful optimization

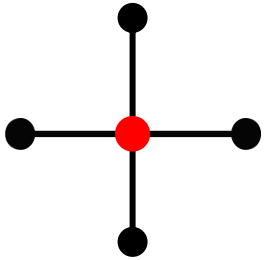
Execution-Cache-Memory (ECM) performance model*

- **Input:**
  - Single-core **execution** time w/ data in L1 (e.g. from Intel IACA tool or manually)
  - **Cache** line transfer between cache levels (processor handbook)
  - Attainable **memory** bandwidth from (multi-) stream measurements
  - Unit of work: cache line size (64B)
- **Assumptions:**
  - Runtime contributions can **overlap** → Upper and lower performance bounds
  - **Scale** single-core performance until first bottleneck is hit
- **Output:**
  - Predicts single-core performance
  - Predicts scaling behavior / saturation point



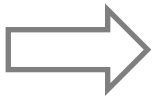
* Treibig & Hager 2010
http://dx.doi.org/10.1007/978-3-642-14390-8_64

Example 1: 2D Jacobi in DP with SSE2 on SNB

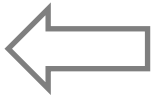


Example 1: 2D Jacobi in DP with SSE2 on SNB

```
1  // Jacobi 2D line update
2  for(int j=start; j<end; j++){
3      t1[i][j]= ( t0[i-1][j]   +
4                  t0[i+1][j]   +
5                  t0[i][j-1]   +
6                  t0[i][j+1] ) * 0.25;
7  }
```



```
1  movups   (%rbp,%r15,8), %xmm1
2  movups   16(%rbp,%r15,8), %xmm3
3  movups   32(%rbp,%r15,8), %xmm5
4  movups   48(%rbp,%r15,8), %xmm7
5  addpd    (%r9,%r15,8), %xmm1
6  addpd    16(%r9,%r15,8), %xmm3
7  addpd    32(%r9,%r15,8), %xmm5
8  addpd    48(%r9,%r15,8), %xmm7
9  addpd    -8(%r10,%r15,8), %xmm1
10 movups   8(%r10,%r15,8), %xmm2
11 movups   24(%r10,%r15,8), %xmm4
12 movups   40(%r10,%r15,8), %xmm6
13 addpd    %xmm2, %xmm3
14 addpd    %xmm4, %xmm5
15 addpd    %xmm6, %xmm7
16 addpd    %xmm2, %xmm1
17 addpd    %xmm4, %xmm3
18 addpd    %xmm6, %xmm5
19 addpd    56(%r10,%r15,8), %xmm7
20 mulpd    %xmm0, %xmm1
21 mulpd    %xmm0, %xmm3
22 mulpd    %xmm0, %xmm5
23 mulpd    %xmm0, %xmm7
24 movups   %xmm1, (%r11,%r15,8)
25 movups   %xmm3, 16(%r11,%r15,8)
26 movups   %xmm5, 32(%r11,%r15,8)
27 movups   %xmm7, 48(%r11,%r15,8)
```



4-way unrolling
→ 8 LUP / iteration

Instruction count

- **13 LOAD**
- **4 STORE**
- **12 ADD**
- **4 MUL**

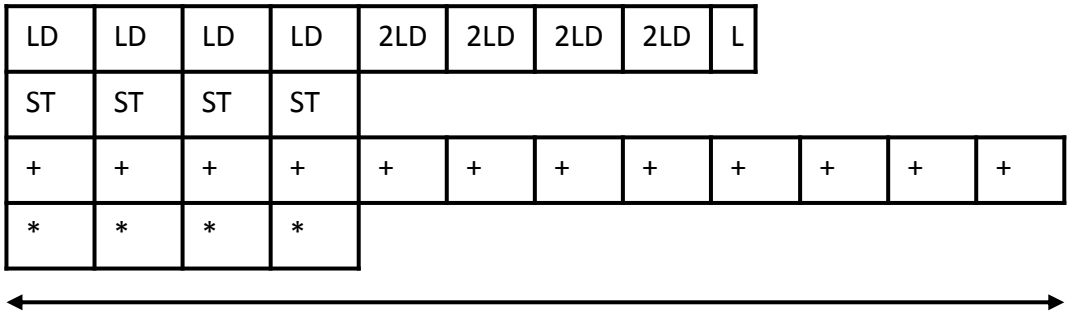
Example 1: 2D Jacobi in DP with SSE2 on SNB

Processor characteristics
(SSE instructions per cycle)

- **2 LOAD || (1 LOAD + 1 STORE)**
- **1 ADD**
- **1 MUL**

Code characteristics
(SSE instructions per iteration)

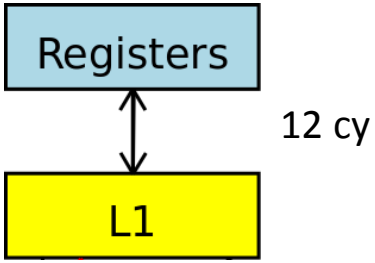
- **13 LOAD**
- **4 STORE**
- **12 ADD**
- **4 MUL**



core execution:
12 cy

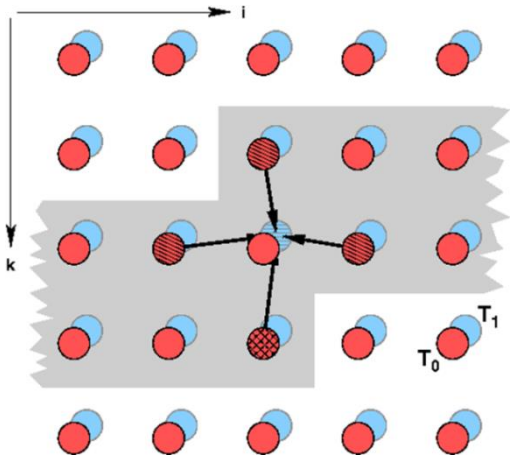
Example 1: 2D Jacobi in DP with SSE2 on SNB

- **Situation 1: Data set fits into L1 cache**
  - ECM prediction:
    (8 LUP / 12 cy) * 3.5 GHz = 2.3 GLUP/s
  - Measurement: 2.2 GLUP/s

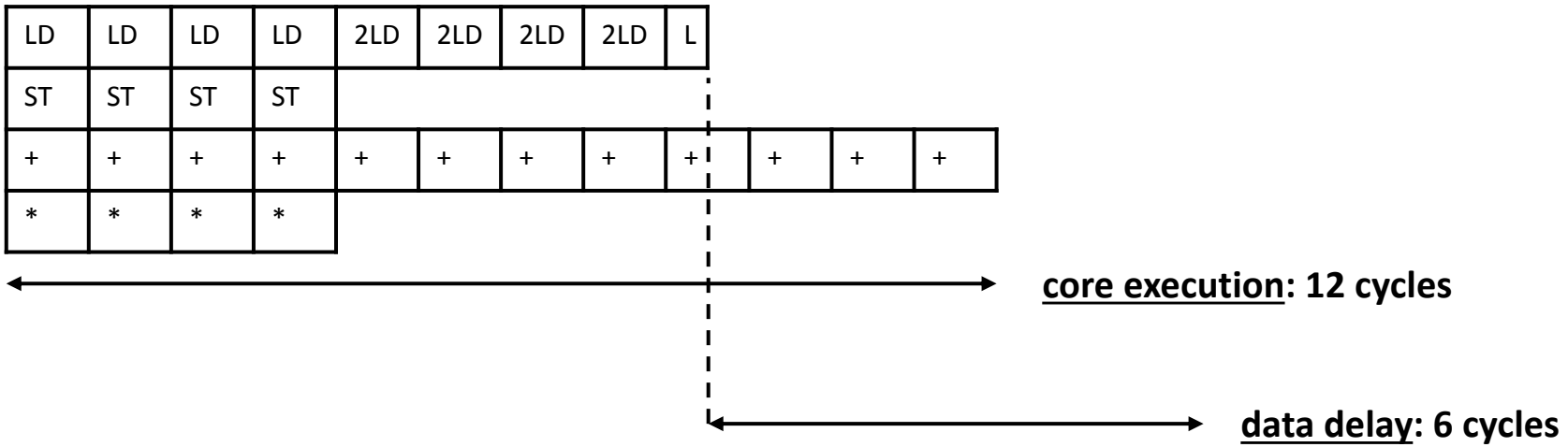


- **Situation 2: Data set fits into L2 cache (not into L1)**
  - 3 additional transfer streams from L2 to L1 (data delay)
  - ECM prediction:
    (8 LUP / (12+6) cy) * 3.5 GHz = 1.5 GLUP/s
  - Measurement: 1.9 GLUP/s

Overlap?

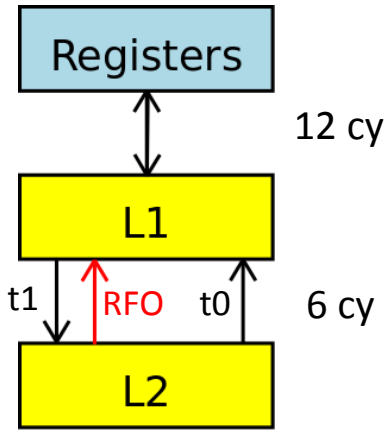


Example 1: 2D Jacobi in DP with SSE2 on SNB



L1 „single ported“
→ no overlap during LD/ST

- ECM prediction w/ overlap:
  (8 LUP / (8.5+6) cy) * 3.5 GHz = 1.9 GLUP/s
- Measurement: 1.9 GLUP/s



“If the model fails, we learn something”

Example 2: 3D long-range stencil in SP with AVX on SNB

Example 2: 3D long-range stencil in SP with AVX on SNB

- **4 neighbors per direction**
- **Operations per update**
  - 27 LOAD (25 V, 1 ROC, 1 U)
  - 1 STORE (U)
  - 26 ADD
  - 15 MUL
- **Core time**
  → IACA

```
1  // 3D long-range line update (single precision)
2  for(i=4; i<nnx-4; i++) {
3     lap = coef0   *   V(i,j,k)
4         + coef[1] * ( V(i+1,j  ,k  ) + V(i-1,j  ,k  ) )
5         + coef[1] * ( V(i  ,j+1,k  ) + V(i  ,j-1,k  ) )
6         + coef[1] * ( V(i  ,j  ,k+1) + V(i  ,j  ,k-1) )
7         + coef[2] * ( V(i+2,j  ,k  ) + V(i-2,j  ,k  ) )
8         + coef[2] * ( V(i  ,j+2,k  ) + V(i  ,j-2,k  ) )
9         + coef[2] * ( V(i  ,j  ,k+2) + V(i  ,j  ,k-2) )
10        + coef[3] * ( V(i+3,j  ,k  ) + V(i-3,j  ,k  ) )
11        + coef[3] * ( V(i  ,j+3,k  ) + V(i  ,j-3,k  ) )
12        + coef[3] * ( V(i  ,j  ,k+3) + V(i  ,j  ,k-3) )
13        + coef[4] * ( V(i+4,j  ,k  ) + V(i-4,j  ,k  ) )
14        + coef[4] * ( V(i  ,j+4,k  ) + V(i  ,j-4,k  ) )
15        + coef[4] * ( V(i  ,j  ,k+4) + V(i  ,j  ,k-4) );
16    U(i,j,k) = 2.f * V(i,j,k) - U(i,j,k) + ROC2(i,j,k) * lap;
17 }
```

Interlude: Intel Architecture Code Analyzer (IACA)

- **Performs architecture-specific code analysis**
- **Prerequisite: Mark start and end of dominant work loop**
  - In high-level code (documented)
  - In assembly code (see iacaMarks.h)
    - Does not influence code optimization (e.g. vectorization)
    - Assembly loop might perform multiple updates per iteration (unrolling, SIMD)
- **Important reports (throughput mode):**
  - Block throughput: runtime of one loop iteration (→ core-time)
  - Throughput bottleneck: limiting resource for code execution
  - Port pressure: dominant pipeline port

IACA example output

```
[..]
Throughput Analysis Report
--------------------------
Block Throughput: 34.25 Cycles       Throughput Bottleneck: FrontEnd

Port Binding In Cycles Per Iteration:
---------------------------------------------------------------------------
|  Port  |  0   -  DV  |  1   |  2   -  D   |  3   -  D   |  4   |  5   |
---------------------------------------------------------------------------
| Cycles | 19.0   0.0  | 26.0 | 31.5   30.5 | 31.5   30.5 | 2.0  | 24.0 |
---------------------------------------------------------------------------
[..]

| Num Of |              Ports pressure in cycles               |     |     |
|  Uops  |  0  - DV  |  1   |  2   -  D   |  3   -  D   |  4   |  5  |     |
---------------------------------------------------------------------------
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | vmovups xmm1, xmmword ptr [r12+r14*4+0x10]
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | vmovups xmm0, xmmword ptr [rsi+r14*4+0x10]
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | vmovups xmm13, xmmword ptr [r12+r14*4+0x14]
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | vmovups xmm14, xmmword ptr [r12+r14*4+0xc]
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | vmovups xmm12, xmmword ptr [rdi+r14*4+0x10]
|   1    |           |      | 0.5    0.5  | 0.5    0.5  |      |     |     | mov r15, qword ptr [rsp+0x338]
|   2    |           |      | 0.5    0.5  | 0.5    0.5  |      | 1.0 |     | vinsertf128 ymm10, ymm1, xmmword ptr [r12+r
|   1    | 1.0       |      |             |             |      |     |     | vmulps ymm1, ymm7, ymm10
|   1    | 1.0       |      |             |             |      |     |     | vmulps ymm15, ymm8, ymm10
|   2    |           |      | 0.5    0.5  | 0.5    0.5  |      | 1.0 |     | vinsertf128 ymm11, ymm0, xmmword ptr [rsi+r
|   1    |           | 1.0  |             |             |      |     |     | vsubps ymm1, ymm1, ymm11
[..]
```

→ AVX vectorization, no unrolling: One iteration updates 8 SP (float) elements

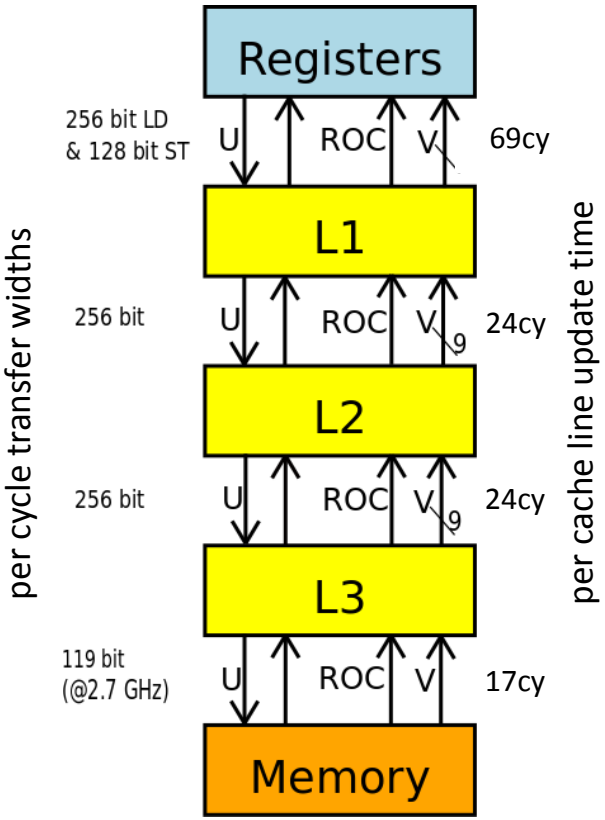
Example 2: 3D long-range stencil in SP with AVX on SNB

- **4 neighbors per direction**
- **Operations per update**
  - 27 LOAD (25 V, 1 ROC, 1 U)
  - 1 STORE (U)
  - 26 ADD
  - 15 MUL
- **Core time (IACA)**
  - 34.25 cy / 8 LUP (SP)
    → 69 cy / CL
  - LOAD dominated

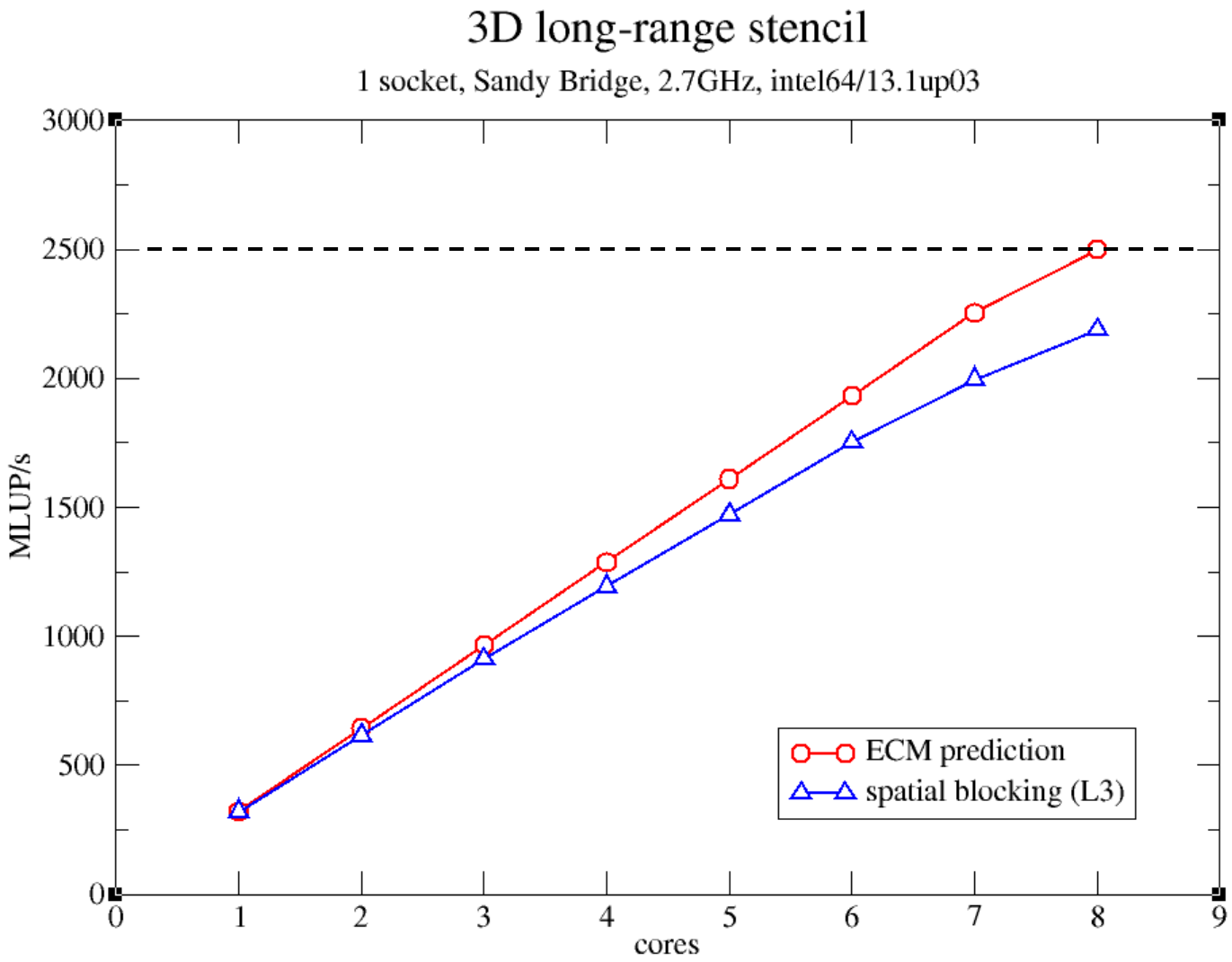
```
1  // 3D long-range line update (single precision)
2  for(i=4; i<nnx-4; i++) {
3     lap = coef0   *   V(i,j,k)
4         + coef[1] * ( V(i+1,j  ,k  ) + V(i-1,j  ,k  ) )
5         + coef[1] * ( V(i  ,j+1,k  ) + V(i  ,j-1,k  ) )
6         + coef[1] * ( V(i  ,j  ,k+1) + V(i  ,j  ,k-1) )
7         + coef[2] * ( V(i+2,j  ,k  ) + V(i-2,j  ,k  ) )
8         + coef[2] * ( V(i  ,j+2,k  ) + V(i  ,j-2,k  ) )
9         + coef[2] * ( V(i  ,j  ,k+2) + V(i  ,j  ,k-2) )
10        + coef[3] * ( V(i+3,j  ,k  ) + V(i-3,j  ,k  ) )
11        + coef[3] * ( V(i  ,j+3,k  ) + V(i  ,j-3,k  ) )
12        + coef[3] * ( V(i  ,j  ,k+3) + V(i  ,j  ,k-3) )
13        + coef[4] * ( V(i+4,j  ,k  ) + V(i-4,j  ,k  ) )
14        + coef[4] * ( V(i  ,j+4,k  ) + V(i  ,j-4,k  ) )
15        + coef[4] * ( V(i  ,j  ,k+4) + V(i  ,j  ,k-4) );
16    U(i,j,k) = 2.f * V(i,j,k) - U(i,j,k) + ROC2(i,j,k) * lap;
17 }
```

Example 2: 3D long-range stencil in SP with AVX on SNB

- **Data delay**
  - Spatial blocking for minimal traffic between L3 cache and memory
  - 8 additional streams for smaller caches
- **Single-core performance**
  - 2.7GHz / (134cy / 16LUP) = 322MLUP/s
  - Measurement: 320MLUP/s
  - LOAD dominated → no overlap expected
- **Socket scaling (8 cores)**
  - 8*322MLUP/s = 2576MLUP/s
  - Limit: 40GB/s / 16B/LUP = 2500MLUP/s
    - Saturation at 8 cores
  - Measurement: ~2200MLUP/s (88% max.)
- **Optimization possibilities**
  - Data transfer well optimized (blocking)
  - Reduce core time (LD); limit: ADD (2*26cy)
    - Possible speedup: 69cy-52cy=17cy (~15%)



Socket scaling



memory bandwidth limit

Thank you.

“There is nothing as practical as a good theory."

Kurt Lewin (1890-1947)