KONWIHR activities at RRZE

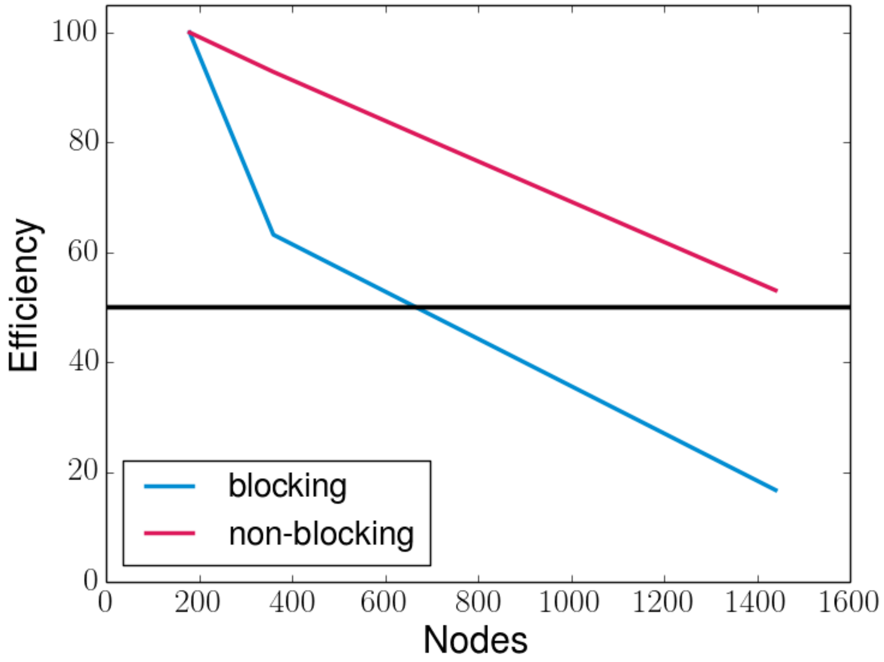
Project “OMI4papps” 2012-2014

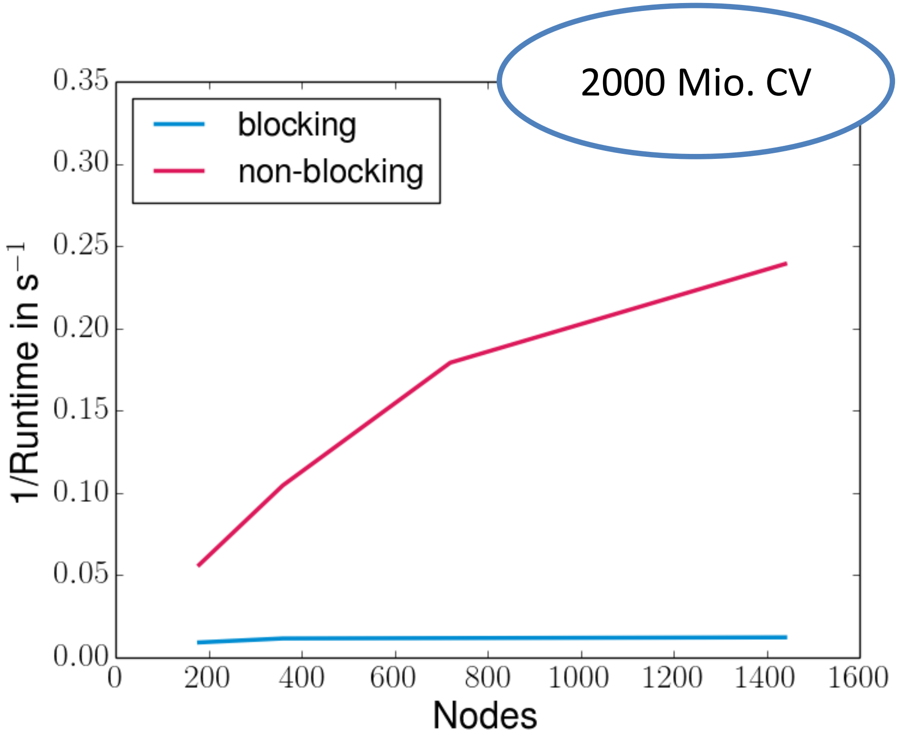
G. Hager, J. Treibig, G. Wellein

KONWIHR software initiative @ RRZE – Projects 2012-14

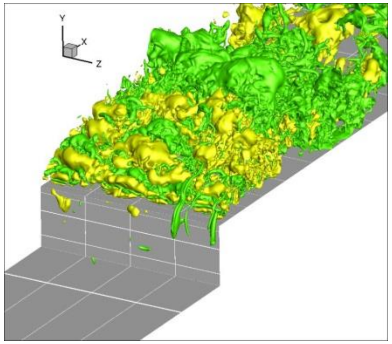
- Treebank: Hochskalierbares Parsen natürlicher Sprache mit High-Performance- und Grid-Computing-Methoden (LS Anglistik, insbesondere Linguistik, FAU)
- Performance Analysis and Parallelization of DFT code in Turbomole (LS Theoretische Chemie, FAU)
- Laufzeitverbesserung von Algorithmen der Sprach- und Sprechererkennung durch Integration von Java und C++ mit Hilfe des Java Native Interfaces (LS Informatik 5, FAU)
- Laufzeitverbesserung der Docking-Vorhersagen im Rahmen des Webservice-Angebots Score-MI (Institut für Biochemie, FAU)
- Entwicklung und Implementierung von hochoptimierten parallelen Differenzensternen auf gestaffelten hierarchischen Gittern (LS Angewandte Mathematik, FAU)
- Numerical solver for the hydrodynamic granular equations (LS Multiscale Simulation of Particulate Systems, FAU)
- Large-Scale Simulations of Small-Scale Physics (LS Rechnerarchitektur, FAU)
- FASTEST-3D (LS Prozessmaschinen und Anlagentechnik, FAU)
  - Making FASTEST-3D ready for modern clusters
  - Optimization of a SIP Solver

FASTEST-3D strong scaling measurements on SuperMUC





- Used non-blocking point-to-point communication instead of MPI_Sendrecv()
- Other optimizations (SP → DP SIP solver, work avoiding)



A process for performance analysis and optimization

We propose a well-defined process to enable a systematic way to performance analysis and optimization

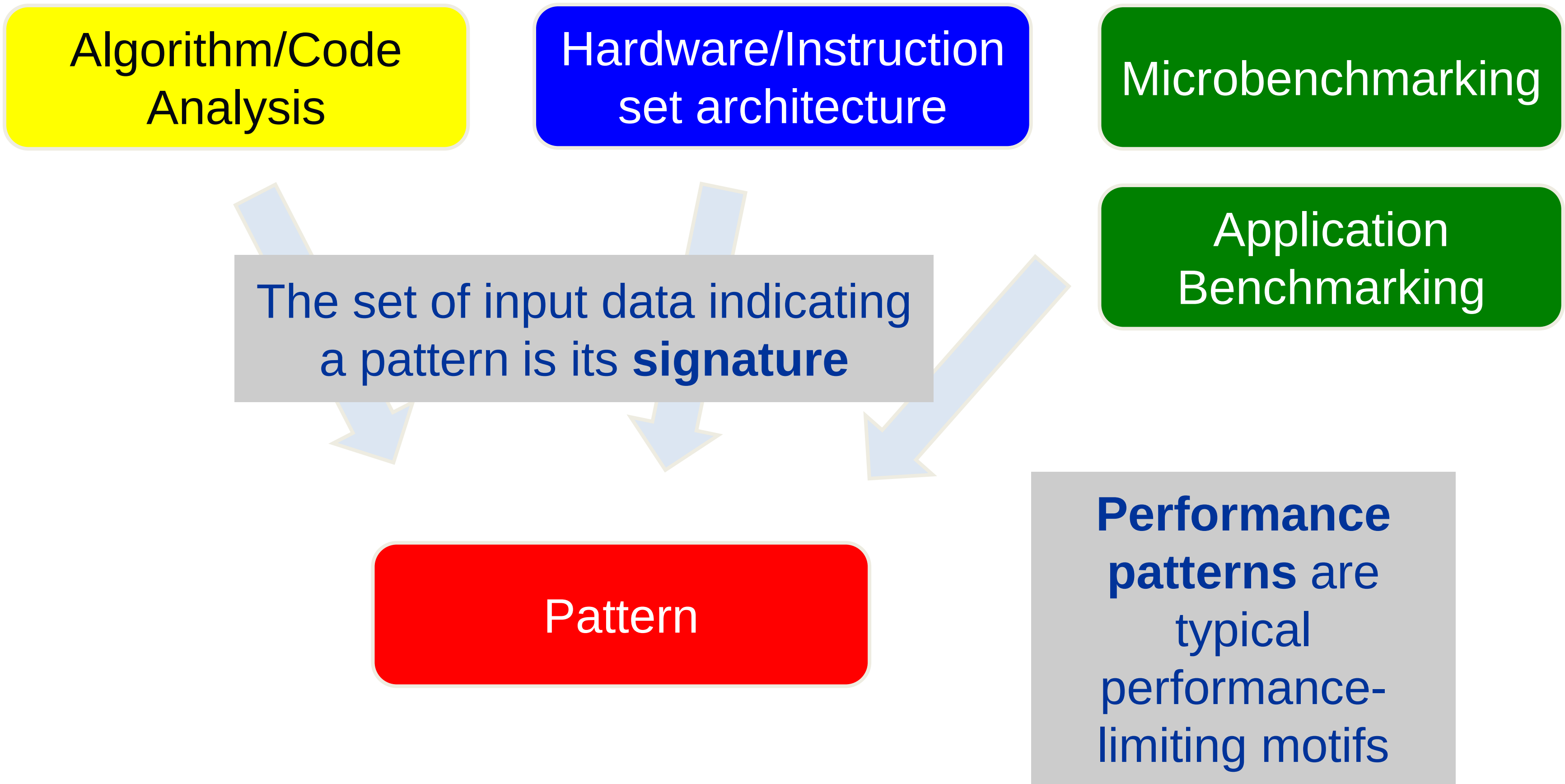
- Executed by humans
- Uses software tools for data acquisition and model validation only

Uses one of the most powerful tools available:

The scientist is an investigator, trying to make sense of what's going on.

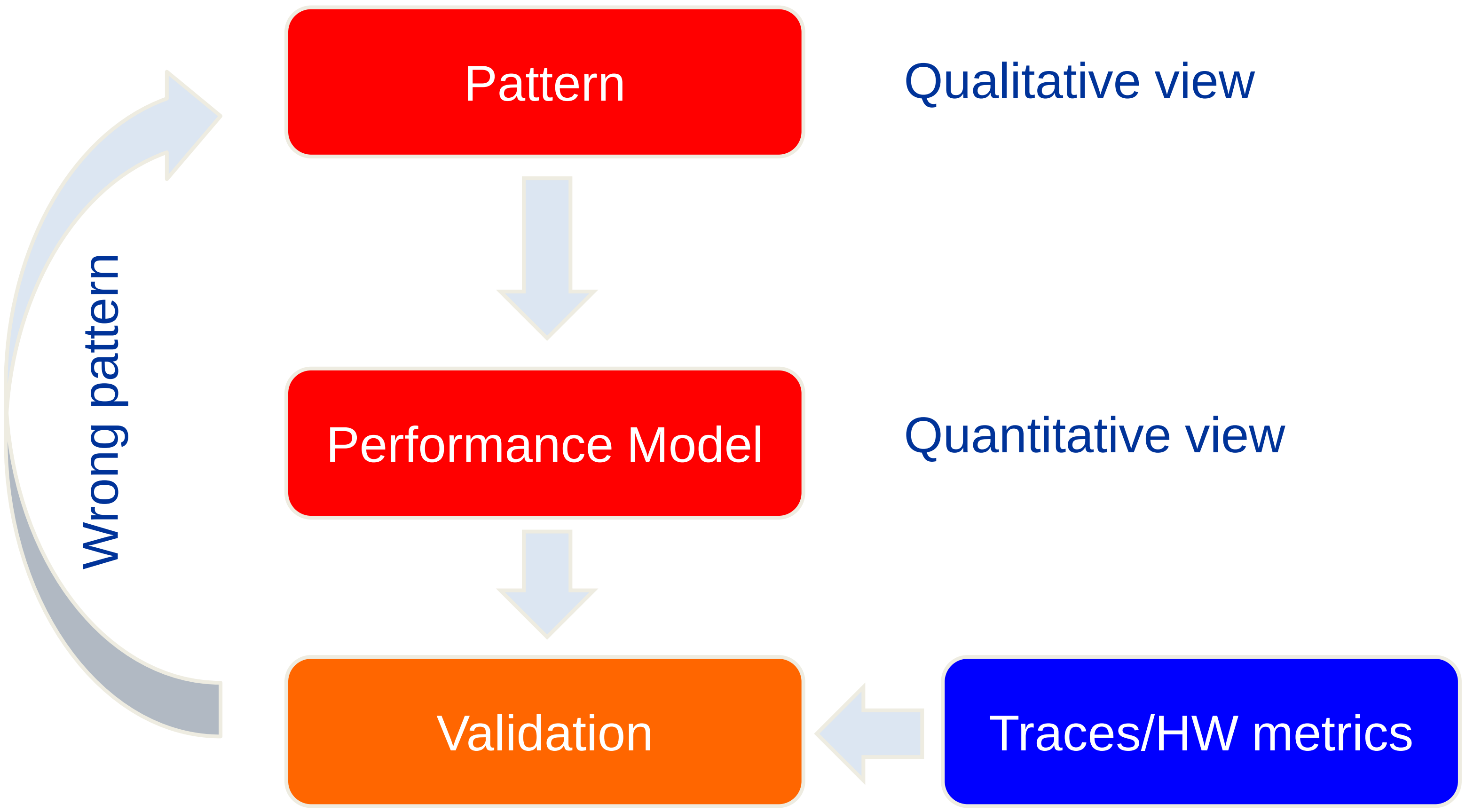
This pertains to domain science as well as to performance aspects of computation

Performance Engineering Process: Analysis



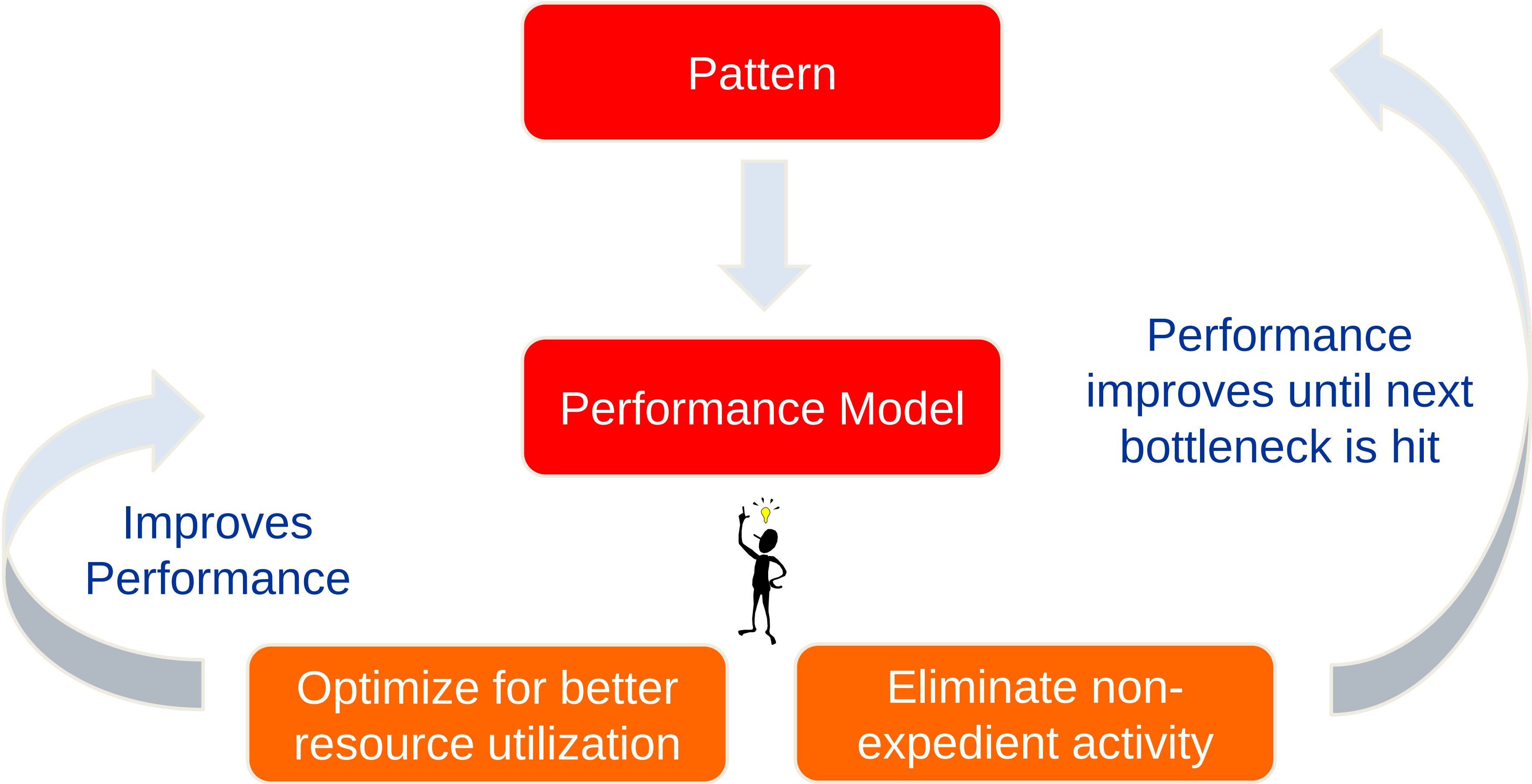
Step 1 **Analysis:** Understanding observed performance

Performance Engineering Process: Modelling



Step 2 **Formulate Model:** Validate pattern and get quantitative insight.

Performance Engineering Process: Optimization



Step 3 **Optimization:** Improve utilization of offered resources.

Optimizing a loop nest from an FEM code – a study in compiler psychology

G. Hager (RRZE)

N. Zander, S. Kollmannsberger
(LS Computation in Engineering, TUM)

The original code

```
counter = 1
DO i=1, sizeA1
   DO j=1, i
      sum = 0.0D0
      DO k=1, 6
         sum = sum + A(k,i) * B(j,k)
      END DO
      C(counter) = C(counter) + sum
      counter = counter + 1
   END DO
END DO
```

P_{ser} = 4.5 Gflop/s

Compiler unrolls k loop completely but fails to vectorize j loop (dependency!)

- sizeA1 = 600
- double precision arrays
- Loop nest executed many times
- Platform: Intel SNB @ 2.7 GHz (fixed)
- Intel compiler 13.1, **`-Ofast -xAVX`**

Manual unrolling, precompute counter, alignment (assumption)

```
DO i=1, sizeA1
  counter = i*(i-1)/2
!DEC$ VECTOR ALIGNED
  DO j=1, i
    C(counter+j) = C(counter+j) + A(1,i) * B(j,1)
    C(counter+j) = C(counter+j) + A(2,i) * B(j,2)
    C(counter+j) = C(counter+j) + A(3,i) * B(j,3)
    C(counter+j) = C(counter+j) + A(4,i) * B(j,4)
    C(counter+j) = C(counter+j) + A(5,i) * B(j,5)
    C(counter+j) = C(counter+j) + A(6,i) * B(j,6)
  END DO
END DO
```

P_{ser} = 9.4 Gflop/s

- Nothing is actually aligned, but the code works anyway (only with AVX!)

AVX assembly code

This is “perfect code:”

```
..B1.15:                        # Preds ..B1.15 ..B1.14
        vmulpd    (%rsi,%r9,8), %ymm1, %ymm6                     #56.58
        vmulpd    (%r13,%r9,8), %ymm0, %ymm8                     #57.58
        vmulpd    (%r12,%r9,8), %ymm5, %ymm10                    #58.58
        vmulpd    (%r11,%r9,8), %ymm4, %ymm12                    #59.58
        vmulpd    (%r15,%r9,8), %ymm3, %ymm14                    #60.58
        vaddpd    (%r10,%r9,8), %ymm6, %ymm7                     #56.21
        vmulpd    (%rdi,%r9,8), %ymm2, %ymm6                     #61.58
        vaddpd    %ymm8, %ymm7, %ymm9                            #57.21
        vaddpd    %ymm10, %ymm9, %ymm11                          #58.21
        vaddpd    %ymm12, %ymm11, %ymm13                         #59.21
        vaddpd    %ymm14, %ymm13, %ymm15                         #60.21
        vaddpd    %ymm6, %ymm15, %ymm7                           #61.21
        vmovupd   %ymm7, (%r10,%r9,8)                            #61.21
        addq      $4, %r9                                        #52.15
        cmpq      %rax, %r9                                      #52.15
        jb        ..B1.15       # Prob 82%                       #52.15
```

A little more with inner loop unrolling

```
DO i=1, sizeA1
  counter = i*(i-1)/2
!DEC$ VECTOR ALIGNED
!DEC$ unroll(4)
  DO j=1, i
    C(counter+j) = C(counter+j) + A(1,i) * B(j,1)
    C(counter+j) = C(counter+j) + A(2,i) * B(j,2)
    C(counter+j) = C(counter+j) + A(3,i) * B(j,3)
    C(counter+j) = C(counter+j) + A(4,i) * B(j,4)
    C(counter+j) = C(counter+j) + A(5,i) * B(j,5)
    C(counter+j) = C(counter+j) + A(6,i) * B(j,6)
  END DO
END DO
```

P_{ser} = 9.6 Gflop/s

- Is this good or bad? (P_{peak} = 20.7 Gflop/s)

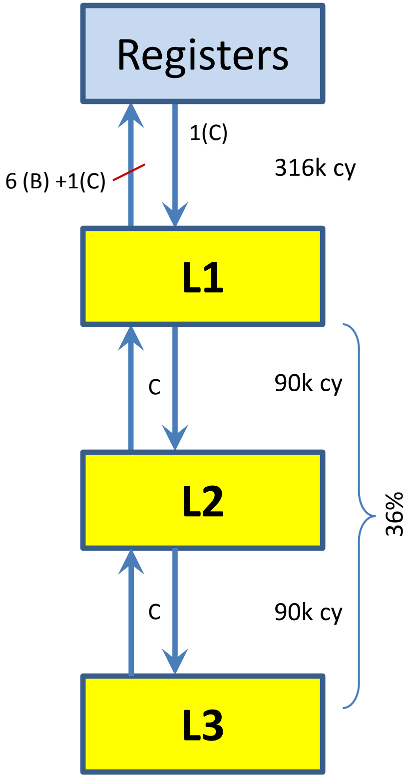
Making sense of the performance – the pattern

- **Assumed pattern: Peak pipeline throughput**
  - ADD, MULT, or LOAD? These are pretty much the only instructions in the code
  - Data is in cache

- **Observed performance is just about half of Peak**
  - We need a performance model!
  - Roofline is not sufficient due to single core code and in-cache data
  - **→ ECM Model!**

Making sense of the performance – ECM Model

- Assumption: C comes from L3 (streaming), A and B are in L1 or registers




```
DO i=1, sizeA1
   counter = i*(i-1)/2
   DO j=1, i
      C(counter+j) = C(counter+j) + A(1,i) * B(j,1)
      C(counter+j) = C(counter+j) + A(2,i) * B(j,2)
      C(counter+j) = C(counter+j) + A(3,i) * B(j,3)
      C(counter+j) = C(counter+j) + A(4,i) * B(j,4)
      C(counter+j) = C(counter+j) + A(5,i) * B(j,5)
      C(counter+j) = C(counter+j) + A(6,i) * B(j,6)
   END DO
END DO
```

- L1: 7 LOADs, 1 STORE, 6 MULT, 6 ADD
  → LOAD limited → 48 Flops in 7 cycles
  → 18.5 Gflop/s
  → All work from L1: 600*601/2*7/4 cy = 316000 cy
- L2/L3: 600*601/2*8*2/32 cy = 90150 cy
- → P_{ECM} = 11.8 Gflop/s

ECM Model: Discrepancies and validation

- Model: 11.8 Gflop/s, Measurement: 9.6 Gflop/s → 82%
- Possible explanations
  - Traffic for B from L2 not negligible
  - Non-streaming access to B in L2 (latency-bound?)
- Validation: Measure L2 cache traffic with likwid-perfctr (10000 kernel invocations)

```
$ likwid-perfctr -C N:1 -g L2 ./matrixMatrixProduct

|   L2 Load [MBytes/s]     |  8371.3  |
|   L2 Evict [MBytes/s]    | 6407.81  |
| L2 bandwidth [MBytes/s] | 14779.1  |
| L2 data volume [GBytes] | 33.4015  |
```

Expected for C:
Load == Evict

Expected for C:
28.8 GB

- Conclusion: Extra traffic must be caused by reloads of B from L2 cache (500 kB ≈ 20x reload)
- Explains ≈30% of the deviation

THANK YOU.