Performance tools part 2:
Performance analysis with hardware metrics

likwid-perfctr

Probing performance behavior

- How do we find out about the performance properties and requirements of a parallel code? Profiling via advanced tools is often overkill
- A coarse overview is often sufficient: **`likwid-perfctr`**

Simple end-to-end measurement of hardware performance metrics

Operating modes:

- Wrapper
- Marker API
- Stethoscope
- Timeline

Preconfigured and extensible metric groups, list with
`likwid-perfctr -a`

```
BRANCH: Branch prediction miss rate/ratio
ICACHE: Instruction cache miss rate/ratio
CLOCK: Clock frequency of cores
DATA: Load to store ratio
FLOPS_DP: Double Precision MFlops/s
FLOPS_SP: Single Precision MFlops/s
FLOPS_HP: Half Precision MFlops/s
FP_PIPE: Utilization of FP pipelines
L2: L2 cache bandwidth in MBytes/s
MEM: Main memory bandwidth in MBytes/s
TLB: TLB miss rate/ratio
PCI: PCI bandwidth in MBytes/s
```

likwid-perfctr wrapper mode

```
$ likwid-perfctr -g L2 -C S1:0-3 ./a.out
--------------------------------------------------------------------------------
CPU type: Fujitsu A64FX […]
--------------------------------------------------------------------------------
<<<< PROGRAM OUTPUT >>>>
--------------------------------------------------------------------------------
Group 1: L2
+-------------------+---------+---------------+---------------+---------------+---------------+
|       Event       | Counter | HWThread 12   | HWThread 13   | HWThread 14   | HWThread 15   |
+-------------------+---------+---------------+---------------+---------------+---------------+
|   INST_RETIRED    |   PMC0  |  5162200861   |  6006277996   |  6094840108   |  6093169915   |
|    CPU_CYCLES     |   PMC1  |  2948059710   |  3323295597   |  3368188649   |  3368279703   |
| L1D_CACHE_REFILL  |   PMC2  |   126214552   |   149155090   |   151368777   |   151481772   |
|   L1D_CACHE_WB    |   PMC3  |    50509285   |    59805474   |    60617584   |    60656185   |
| L1I_CACHE_REFILL  |   PMC4  |      294489   |       13536   |        1303   |        1271   |
+-------------------+---------+---------------+---------------+---------------+---------------+
[… statistics output omitted …]
+-------------------------------------+---------------+---------------+---------------+---------------+
|               Metric                | HWThread 12   | HWThread 13   | HWThread 14   | HWThread 15   |
+-------------------------------------+---------------+---------------+---------------+---------------+
|        Runtime (RDTSC) [s]          |       0.5754  |       0.5754  |       0.5754  |       0.5754  |
|                CPI                  |       0.5711  |       0.5533  |       0.5526  |       0.5528  |
|  L1D<-L2 load bandwidth [MBytes/s]  |   56153.2645  |   66359.5844  |   67344.4610  |   67394.7328  |
|  L1D<-L2 load data volume [GBytes]  |      32.3109  |      38.1837  |      38.7504  |      38.7793  |
| L1D->L2 evict bandwidth [MBytes/s]  |   22471.7451  |   26607.6498  |   26968.9602  |   26986.1339  |
| L1D->L2 evict data volume [GBytes]  |      12.9304  |      15.3102  |      15.5181  |      15.5280  |
|  L1I<-L2 load bandwidth [MBytes/s]  |     131.0191  |       6.0222  |       0.5797  |       0.5655  |
|  L1I<-L2 load data volume [GBytes]  |       0.0754  |       0.0035  |       0.0003  |       0.0003  |
|     L1<->L2 bandwidth [MBytes/s]    |   78756.0287  |   92973.2564  |   94314.0009  |   94381.4322  |
|     L1<->L2 data volume [GBytes]    |      45.3167  |      53.4974  |      54.2688  |      54.3076  |
+-------------------------------------+---------------+---------------+---------------+---------------+
```

HW threads selected

Raw events (this group)

Derived metrics

`likwid-perfctr` marker API

- The marker API can restrict measurements to code regions
- The API only turns counters on/off. The configuration of the counters is still done by `likwid-perfctr`
- Multiple named regions support, accumulation over multiple calls
- Inclusive and overlapping regions allowed

```
#include <likwid-marker.h>
. . .
LIKWID_MARKER_INIT;                    // must be called from serial region

. . .
LIKWID_MARKER_START("Compute");    // call markers for each thread
. . .
LIKWID_MARKER_STOP("Compute");
. . .
LIKWID_MARKER_START("Postprocess");
. . .
LIKWID_MARKER_STOP("Postprocess");
. . .

LIKWID_MARKER_CLOSE;                   // must be called from serial region
```

Before LIKWID 5 use **`likwid.h`**

- Activate macros with **`-DLIKWID_PERFMON`**
- Run **`likwid-perfctr`** with **`-m`** switch to enable marking
- See **https://github.com/RRZE-HPC/likwid/wiki/TutorialMarkerF90** for Fortran example

Compiling, linking, and running with marker API

Compile:

```
cc -I $LIKWID_INCDIR -DLIKWID_PERFMON -c program.c
```

Link:

```
cc -L $LIKWID_LIBDIR program.o -llikwid
```

Run:

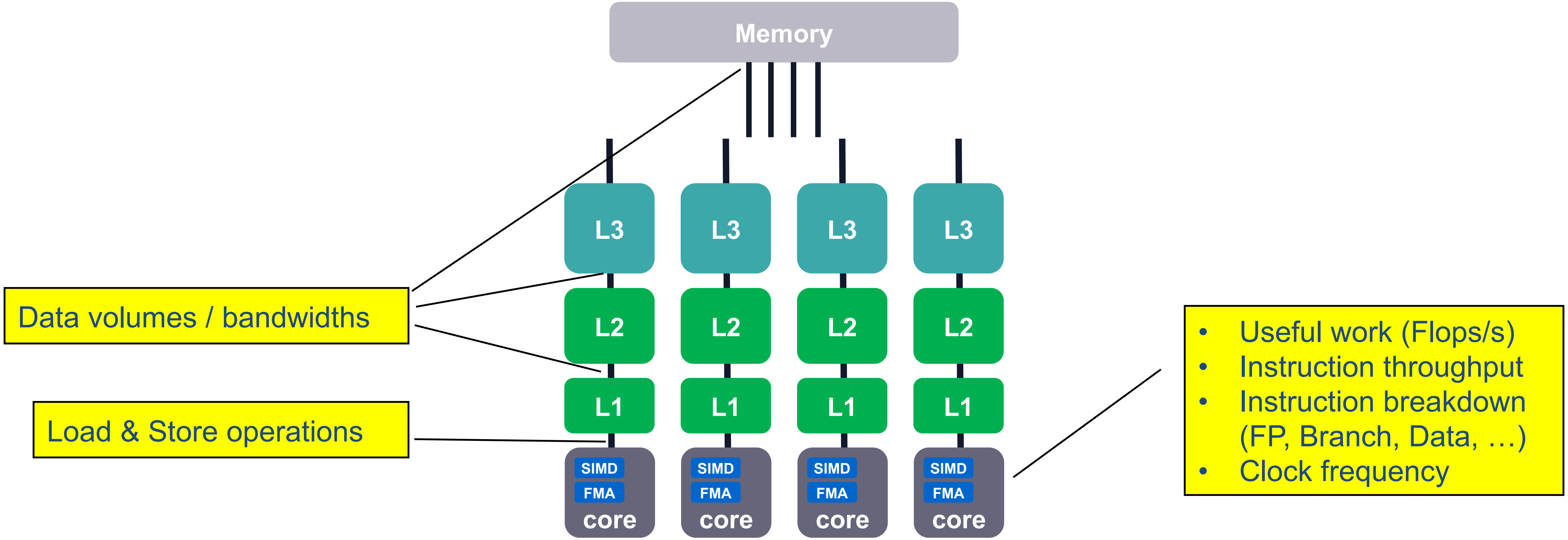
```
likwid-perfctr -C <MASK> -g <GROUP> -m ./a.out
```

→ One separate block of output for every marked region

→ Caveat: Marker API can cause overhead; do not call too frequently!

- Commonly `$LIKWID_INCDIR` and `$LIKWID_LIBDIR` managed by module environment
- OOKAMI: `LIKWID_{INC|LIB}DIR=/lustre/software/likwid/5.1.1/{include|lib}`
  Disable PCP: `/var/lib/pcp/pmdas/perfevent/perfalloc -d`

So... what should I look at first?



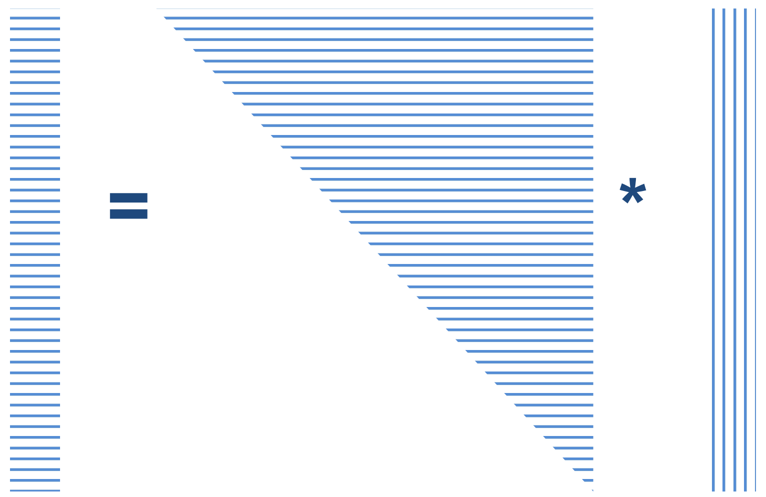
All the above metrics can be acquired using performance groups:

`MEM_DP, MEM_SP, BRANCH, DATA, L2`

Example: triangular matrix-vector multiplication

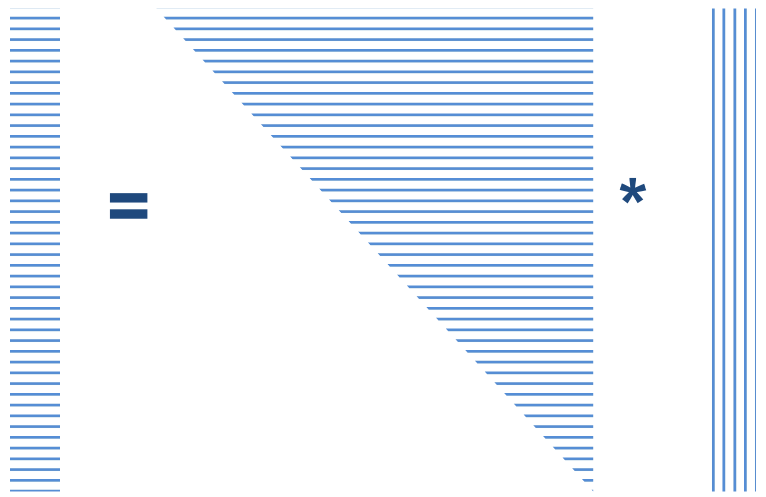
```
#define N 10000 // matrix in memory
#define ROUNDS 10
// Initialization
fillMatrix(mat, N*N, M_PI);
fillMatrix(bvec, N, M_PI);

// Calculation loop
#pragma omp parallel
{
    for (int k = 0; k < ROUNDS; k++) {
        #pragma omp for private(current,j)
        for (int i = 0; i < N; i++) {
            current = 0;
            for (int j = i; j < N; j++)
                current += mat[(i*N)+j] * bvec[j];
            cvec[i] = current;
        }
        while (cvec[N>>1] < 0) {dummy();break;}
    }
}
```



Example: triangular matrix-vector multiplication

```
use likwid
call likwid_markerInit()
!$omp parallel private(iter,c,r)
do iter=1,rep
   call likwid_markerStartRegion(„Compute")
   !$omp do reduction(+:y)
   do c=1,columns
      do r=1,c
         Y(r) = Y(r) + A(r,c) * X(c)
      enddo
   enddo
   !$omp end do
   if(Y(rows/2)<0.d0) print *,Y(rows/2)
   call likwid_markerStopRegion(„Compute")
enddo
!$omp end parallel
call likwid_markerClose()
```



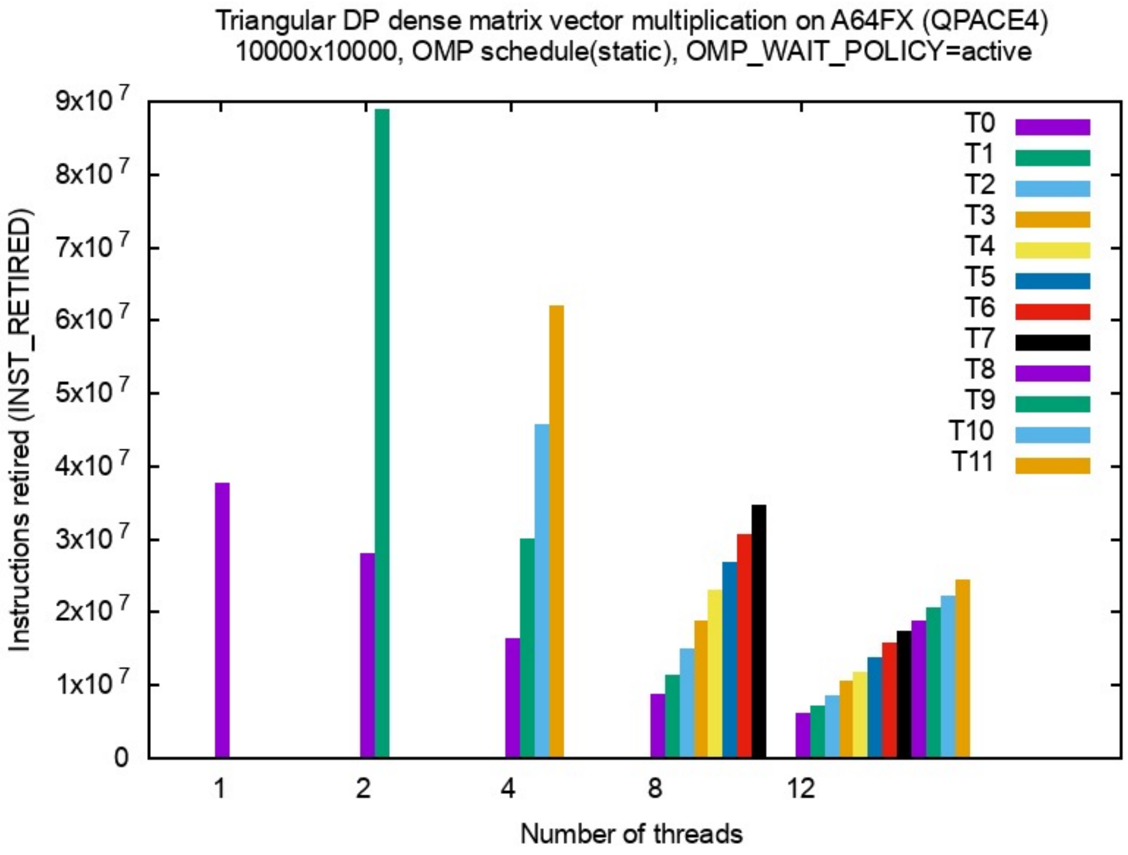
Example: triangular matrix-vector multiplication

```
$ OMP_WAIT_POLICY=active likwid-perfctr -C 0,1,2 -g L2 -m ./a.out
--------------------------------------------------------------------------------
CPU type: Fujitsu A64FX
--------------------------------------------------------------------------------
Region Compute, Group 1: L2
+-------------------+------------+------------+------------+
|    Region Info    | HWThread 0 | HWThread 1 | HWThread 2 |
+-------------------+------------+------------+------------+
| RDTSC Runtime [s] |   0.596584 |   0.596547 |   0.596601 |
|     call count    |          1 |          1 |          1 |
+-------------------+------------+------------+------------+

+------------------+---------+------------+------------+------------+
|       Event      | Counter | HWThread 0 | HWThread 1 | HWThread 2 |
+------------------+---------+------------+------------+------------+
|   INST_RETIRED   |   PMC0  | 3365588000 | 4929463000 | 6959266000 |???
|    CPU_CYCLES    |   PMC1  | 3148230000 | 3169624000 | 3219023000 |
| L1D_CACHE_REFILL |   PMC2  |  246731800 |  145789800 |   28658610 |
|   L1D_CACHE_WB   |   PMC3  |  129227900 |   73909810 |    3699794 |
| L1I_CACHE_REFILL |   PMC4  |       6485 |       1968 |       2033 |
+------------------+---------+------------+------------+------------+
```

Example: triangular matrix-vector multiplication

Retired instructions might be misleading!
Waiting in implicit OpenMP barrier executes many instructions



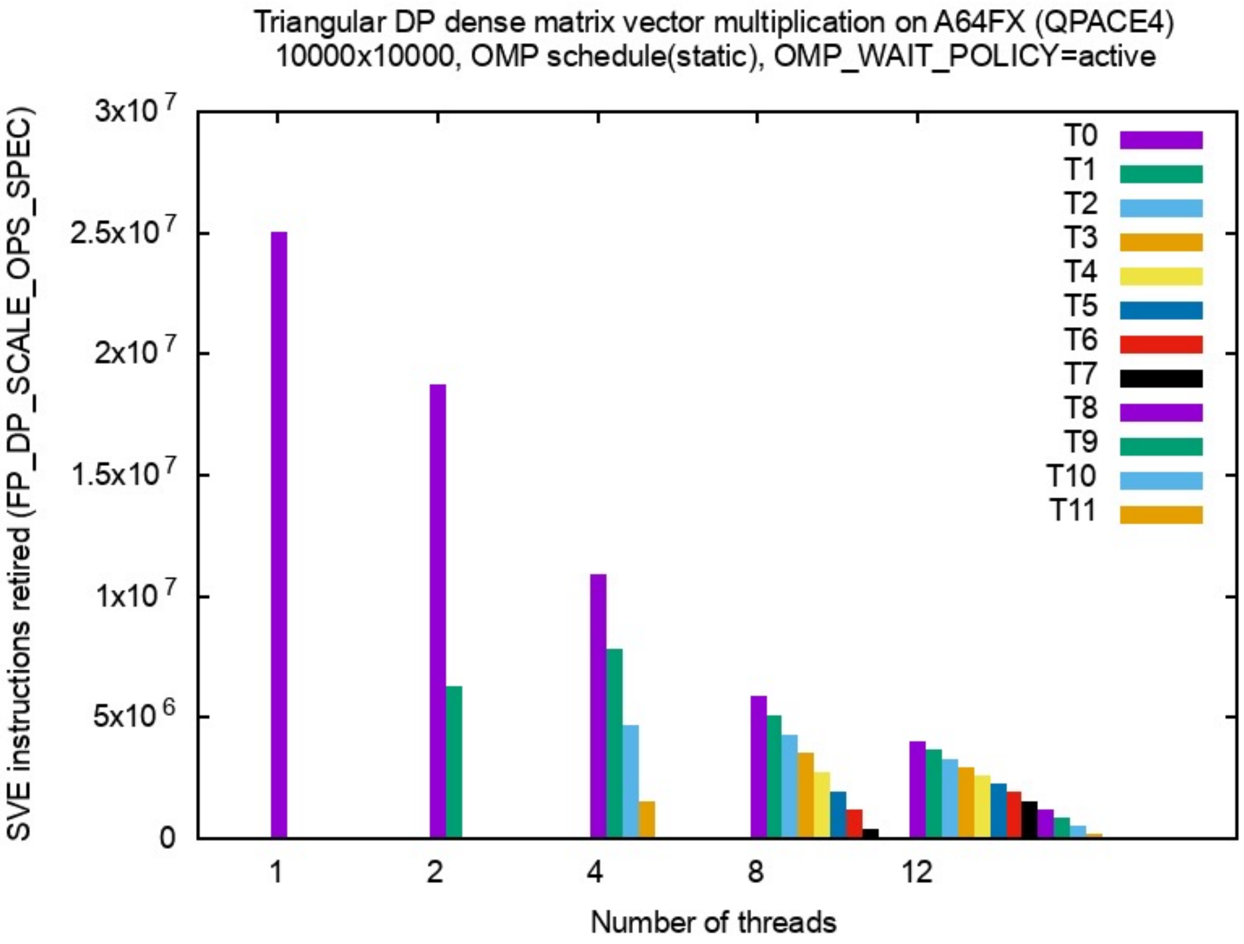
We need to measure actual work (or use a tool that can separate user from runtime lib instructions)

Example: triangular matrix-vector multiplication

Floating-point instructions reliable ↔ useful work metric

Caveats

- Width/scale of SVE instructions not included in hardware performance events

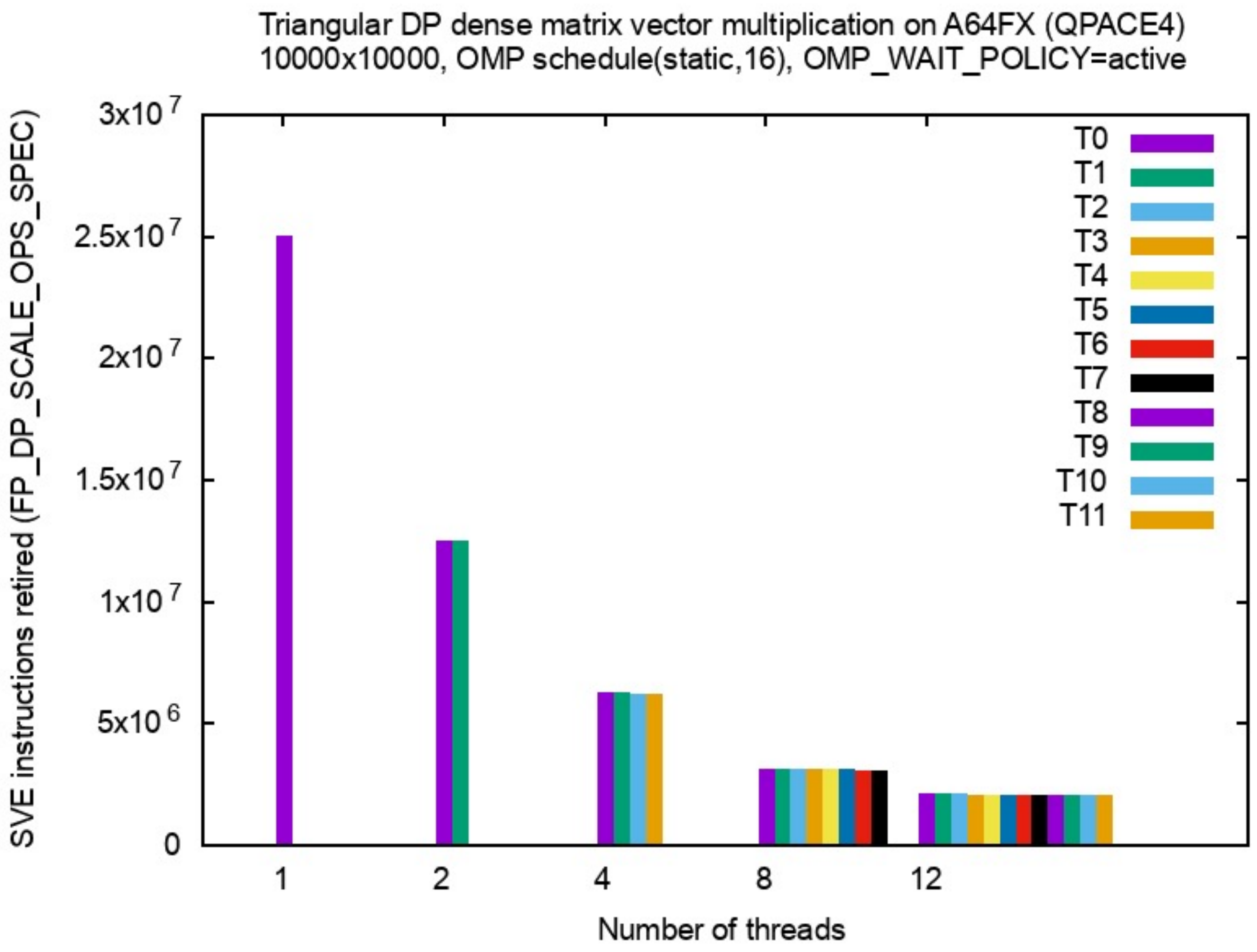


Example: triangular matrix-vector multiplication

Changing OMP schedule to static with chunk size 16 ↔ smaller work packages per thread

No imbalance anymore!

Is it also faster?



Summary of hardware performance monitoring

- Useful only if you know what you are looking for
  - Hardware event counting bears the potential of acquiring massive amounts of data for nothing!

- Resource-based metrics are most useful
  - Cache lines transferred, work executed, loads/stores, cycles
  - Instructions, CPI, cache misses may be misleading

- Caveat: Processor work != user work
  - Waiting time in libraries (OpenMP, MPI) may cause lots of instructions
  - → distorted application characteristic

- Another very useful application of PM: validating performance models!
  - Roofline is data centric → measure data volume through memory hierarchy