HPC Code Optimisation Workshop 2022

LIKWID Hands On

Thomas Gruber, Software & Tools, NHR@FAU

Preparation

- All files are located in the Github repo
  https://github.com/carlabguillen/hellolikwid

- Folder contains two subfolders:
  - **`dmvm`**: Dense DP Matrix-Vector-Multiplication
  - **`roofline`**: Example gnuplot script to generate Roofline plot

- Although LIKWID can be used for MPI jobs, we focus on single-node in this Hands On

`likwid-perfctr` marker API

- The marker API can restrict measurements to code regions
- The configuration of the counters is 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_REGISTER("Compute"); // optional, call markers for each thread
. . .
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
- APIs for Java, Python, Lua and Julia exist

Compiling, linking, and running with marker API

Compile:

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

Defined by LIKWID module at CooLMUC2

Link:

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

Run:

```
likwid-perfctr -C <CPULIST> -g <GROUP> -m ./program
```

→ One separate block of output for every marked region

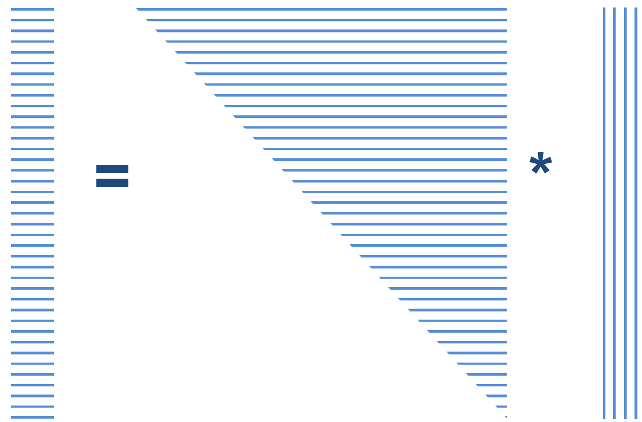
→ Caveat: Marker API can cause overhead; do not call too frequently!

Dense DP Matrix-Vector-Multiplication

For demonstration purposes: Only triangular matrix

Example: triangular matrix-vector multiplication

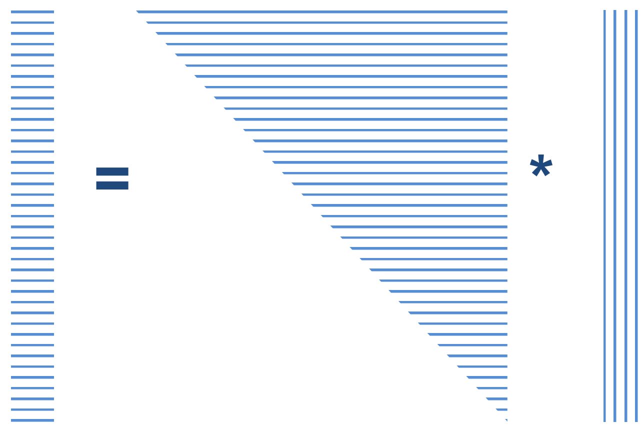
```
#define N 10000 // matrix in memory
#define ROUNDS 10
// Initialization
fillMatrix(mat, N*N, M_PI);
fillMatrix(bvec, N, M_PI);

// 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;}
    }
}
```



Guard with „impossible“ condition and **`dummy()`** function call required if **`cvec`** not used afterwards

Example: triangular matrix-vector multiplication

```
#include <likwid-marker.h>
[…] // defines, fillMatrix, init data
LIKWID_MARKER_INIT;
#pragma omp parallel
{
    for (int k = 0; k < ROUNDS; k++) {
        LIKWID_MARKER_START(“Compute”);
        #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;
        }
        LIKWID_MARKER_STOP(“Compute”);
        while (cvec[N>>1] < 0) {dummy();break;}
    }
}
LIKWID_MARKER_CLOSE;
```



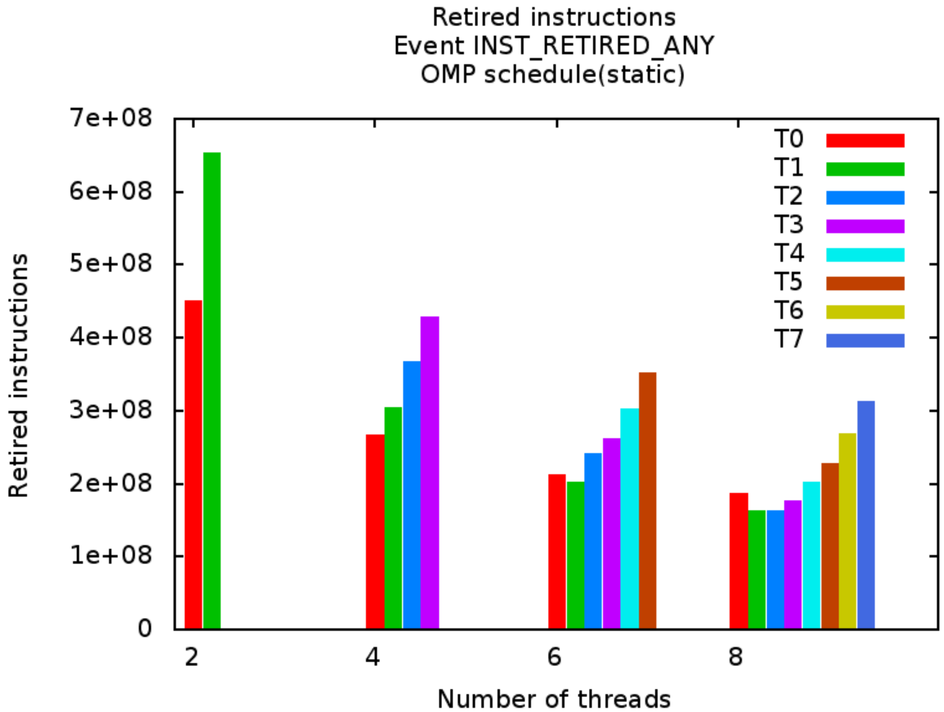
Example: triangular matrix-vector multiplication

```
$ likwid-perfctr -C 0,1,2 -g L2 -m ./a.out
--------------------------------------------------------------------------------
CPU type: Intel Xeon Haswell EN/EP/EX processor
CPU clock:          2.30 GHz
--------------------------------------------------------------------------------
Region Compute, Group 1: L2
+-------------------+------------+------------+------------+
|    Region Info    | HWThread 0 | HWThread 1 | HWThread 2 |
+-------------------+------------+------------+------------+
| RDTSC Runtime [s] |  27.953160 |  27.949260 |  27.920180 |
|     call count    |        100 |        100 |        100 |
+-------------------+------------+------------+------------+
+-----------------------+---------+-------------+-------------+-------------+
|         Event         | Counter |  HWThread 0 |  HWThread 1 |  HWThread 2 |
+-----------------------+---------+-------------+-------------+-------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 50077730000 | 30286830000 | 10286100000 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  | 75722540000 | 44191610000 | 13774590000 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  | 59173590000 | 35044110000 | 12890660000 |
|    L1D_REPLACEMENT    |   PMC0  |  6265837000 |  3760637000 |  1227229000 |
|    L2_TRANS_L1D_WB    |   PMC1  |     7262759 |     5481429 |     2614783 |
|     ICACHE_MISSES     |   PMC2  |      139899 |      113342 |       72966 |
+-----------------------+---------+-------------+-------------+-------------+
```

Instruction increase with ThreadID???

Example: triangular matrix-vector multiplication

Retired instructions are misleading!
Waiting in implicit OpenMP barrier executes many instructions



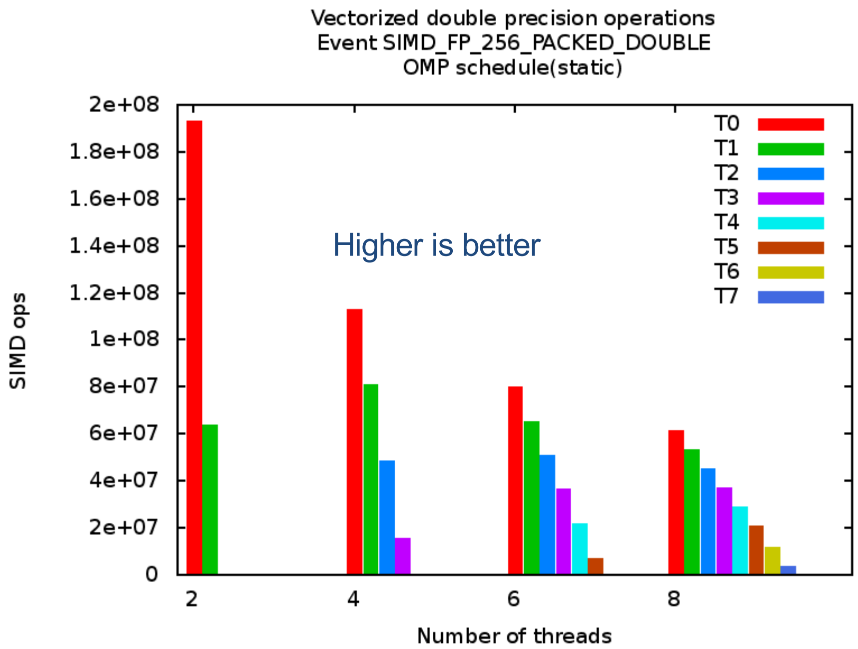
We need to measure actual work (or use a tool that can separate user from runtime lib instructions)

Example: triangular matrix-vector multiplication

Floating-point instructions reliable ↔ useful work metric

Caveats

- FP instr. counters from SandyBridge to Haswell are only qualitatively correct
- Masked SIMD lanes (AVX512) cannot be counted directly on x86

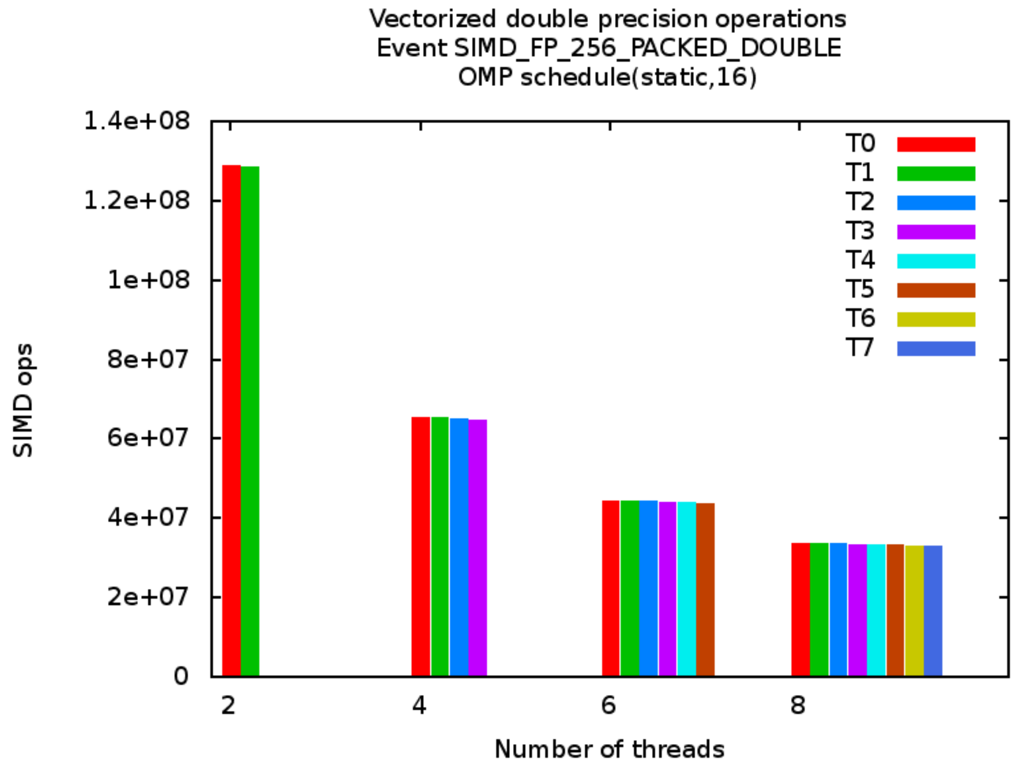


Haswell provides only the **`AVX_INST`** event. Only AVX instruction but more than only calculations

Example: triangular matrix-vector multiplication

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

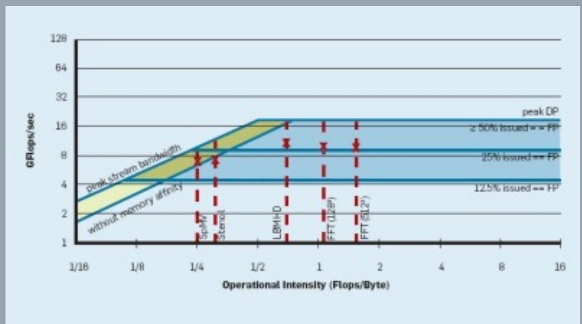
No imbalance anymore! Less waiting time in barrier.



NHR FAU

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Empirical Roofline model with LIKWID



High Performance Computing

Recap: The Roofline Model

Apply the naive Roofline model in practice

- Machine parameter #1: Peak performance: $P_{peak} \left[\frac{F}{s}\right]$
- Machine parameter #2: Memory bandwidth: $b_S \left[\frac{B}{s}\right]$

} Machine model

- Code characteristic: Computational intensity: $I \left[\frac{F}{B}\right]$

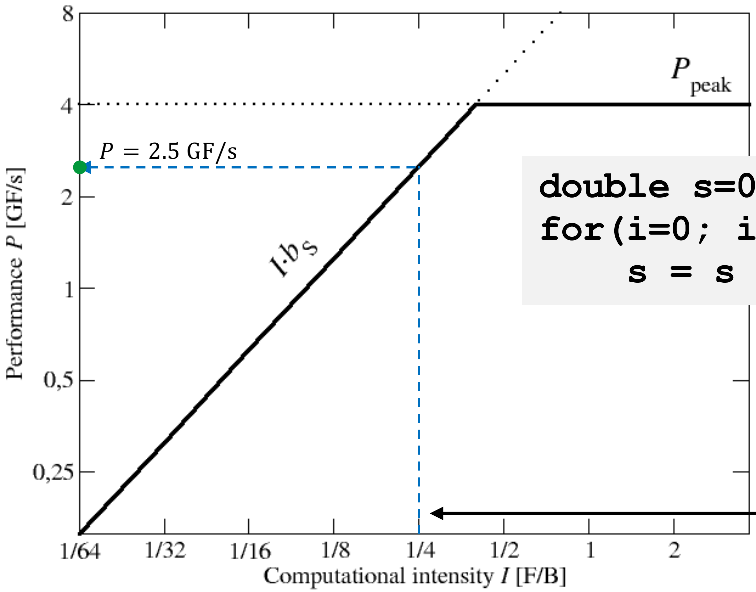
} Application model

Machine properties:

$$\boldsymbol{P_{peak}} = 4 \frac{\text{GF}}{\text{s}}$$

$$\boldsymbol{b_S} = 10 \frac{\text{GB}}{\text{s}}$$

Application property: I




```
double s=0, a[];
for(i=0; i<N; ++i) {
    s = s + a[i] * a[i];}
```

$$I = \frac{2\ F}{8\ B} = 0.25\ {}^{F}/_{B}$$

Recap: Refined Roofline Model

1. P_{max} = Applicable peak performance of a loop, assuming that data comes from the level 1 cache (this is not necessarily P_{peak})
   → e.g., P_{max} = 176 GFlop/s

2. I = Computational intensity ("work" per byte transferred) over the slowest data path utilized (code balance $B_C = I^{-1}$)
   → e.g., I = 0.167 Flop/Byte → B_C = 6 Byte/Flop

3. b_S = Applicable (saturated) peak bandwidth of the slowest data path utilized
   → e.g., b_S = 56 GByte/s

"Flop" is not the only useful unit of work!

Performance limit:

$$P = \min(P_{\max}, I \cdot b_S) = \min\left(P_{\max}, \frac{b_S}{B_C}\right)$$

[Byte/s] (b_S)
[Byte/Flop] (B_C)

R.W. Hockney and I.J. Curington: $f_{1/2}$: A parameter to characterize memory and communication bottlenecks. Parallel Computing 10, 277-286 (1989). DOI: 10.1016/0167-8191(89)90100-2

W. Schönauer: Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers. Self-edition (2000)

S. Williams: Auto-tuning Performance on Multicore Computers. UCB Technical Report No. UCB/EECS-2008-164. PhD thesis (2008)

`likwid-bench`

- **`likwid-bench`** provides a set of assembly kernels
- Different implementations dependent on the architecture:
  DP/SP, scalar, SSE, AVX, AVX512, NEON, SVE, FMAs, NT-Stores
- Examples (more with **`likwid-bench -a`**):
  - **`load`** (load only)
  - **`store`** (store only)
  - **`stream`** (**`A[i] = B[i] + s* C[i]`**), **`stream_sp`**, **`stream_sse`**, …

Thread-local initialization, use **`-W`**

- **`likwid-bench -t stream_mem_avx -w S0:200kB:8:1:2`**

Location for data and threads:
N: node
Sx: socket x
My: NUMA domain y

Size for all data

For stream:
each array 200kB/3

Number of threads

`<threads>` or
`<threads>:<chunk>:<stride>`

Get input data for Roofline

- Use a benchmark „similar“ to your application kernel!
- Maximum performance:
  - All data in L1 (**L1_SIZE/2 * NUM_THREADS**) (**2** is a safety factor)
- Maximum bandwidth:
  - Use big enough data sizes (**>= 2GB**)
  - Use thread-local initialization (**-W**)

- Mark region in code with MarkerAPI
- Run with **likwid-perfctr -g MEM_DP/SP -m**
  - Sum of ‚Operational intensity STAT‘
  - Sum of Flops in statistics table

Gnuplot script

- Add application dot:

```
set object circle at first op_ins,app_perf radius char 0.5 fc rgb 'red' fs solid
```

- Generate roofline:

```
roof(op_ins) = maxperf > (op_ins * maxband)
                          ? (op_ins * maxband)
                          : maxperf
plot roof(x) notitle
```

- See **`likwid-roofline.gnuplot`** for a more extended script
  - Title
  - Labels
  - Preparation for additional rooflines
    (be aware that the operational intensity might change depending on cache level)