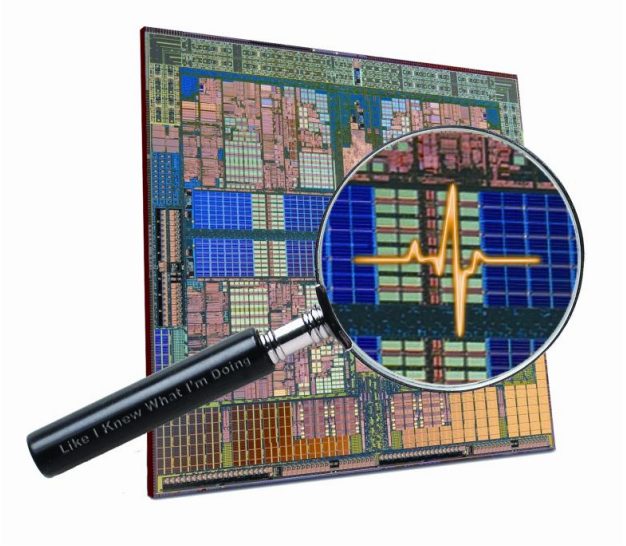


LIKWID – Lightweight Performance Tools

Performance analysis with LIKWID

Thomas Röhl

HPC Services, Erlangen Regional Computing Center (RRZE)

- **What is Likwid?**
  - Topology and Affinity
  - Benchmarking
  - Performance & Power Profiling
- **Example analysis**
- **Novelties in Version 4.0**

LIKWID Lightweight Performance Tools

- **Lightweight command line tools for Linux**
- **Help to face the challenges without getting in the way**
- **Focus on X86 architecture**
- **Philosophy:**
  - Simple
  - Efficient
  - Portable
  - Extensible

Open source project (GPL v3):

http://code.google.com/p/likwid/

Overview of LIKWID tools

- **Topology and Affinity:**
  - likwid-topology
  - likwid-pin
- **Performance Profiling:**
  - likwid-setFrequencies
  - likwid-memsweeper
  - likwid-perfctr
  - likwid-powermeter
- **Benchmarking:**
  - likwid-bench
- **Misc.**
  - likwid-mpirun, likwid-perfscope, likwid-agent (4.0)

Scenario

- **New machine arrived**
- **No in-depth characteristics of the system known**
  - Topology of CPUs, caches and memory domains
  - How fast is the cache and memory hierarchy?
  - Intel only: How much power is consumed?
  - (How many FLOPs can the system achieve?)
- **How does my application behave on the machine?**
  - Single core performance?
  - Does it scale on the socket? Memory domain? Node?

likwid-topology

- Read thread, cache and NUMA topology
- Can measure current CPU clock
- Output detailed cache information
- Print system architecture as ASCII art
- Convert topology information in processable files (XML, CSV, …)

```
--------------------------------------------------------------------------------
CPU name:               Intel(R) Xeon(R) CPU E5-2687W 0 @ 3.10GHz
CPU type:               Intel Core SandyBridge EN/EP processor
CPU stepping:           7
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:                2
Cores per socket:       8
Threads per core:       2
--------------------------------------------------------------------------------
HWThread        Thread        Core        Socket
0               0             0           0
1               0             1           0
[…]
```

Determine system characteristics

- **Knowledge about the performance capabilities of a machine essential for any optimization effort**
  - How many loads and stores per cycle?
  - How many FLOPs per cycle?
  - Cache hierarchy and memory bandwidths?

1. **Write own codes (C, C++, Fortran)**
   - Does it benchmark the right thing?
   - Does the compiler do its optimizations properly?

   Check assembly for each code, compiler, flag
2. **Use benchmark suites e.g. likwid-bench** ☺

Overview – likwid-bench (1)

- **Assembly benchmark kernels**
  - No compiler optimizations
  - Required data volume and FLOPs known
- **Flexible and extensible**
- **Thread management and pinning**
- **NUMA aware data decomposition**
- **Currently 57 ready-to-use stream access kernels**
  - Single and double precision
  - Different vectorization features (NONE, SSE, AVX)
  - Different writeback schemes (Non-temporal stores)
- **Abstract assembly for easy integration of own kernels**

Overview – likwid-bench (2)

- `likwid-bench -t copy -g 1 -w S0:1GB:4`

```
Allocate: Process running on core 0 - Vector length 67108864 Offset 0
Allocate: Process running on core 0 - Vector length 67108864 Offset 0
-------------------------------------------------------------
LIKWID MICRO BENCHMARK
Test: copy
-------------------------------------------------------------
Using 1 work groups
Using 4 threads
-------------------------------------------------------------
-------------------------------------------------------------
Cycles: 15954769356
Iterations: 100
Size 67108864
Vectorlength: 16777216
Time: 5.159166e+00 sec
Number of Flops: 0
MFlops/s: 0.00
MByte/s: 20812.31
Cycles per update: 2.377446
Cycles per cacheline: 52.832131
-------------------------------------------------------------
```

- **LIKWID 3.1 requires iteration count (-i)**

Overview – likwid-pin (1)

- **Run code using knowledge about system topology**
- **CPU list accepts multiple syntaxes:**
  - Physical numbering of OS (0,4-6 or 7,5,3)
  - Logical numbering inside an affinity domain

    (S0:0-3 or M0:0,2,4)
  - Numbering expression (E:S0:2 or E:N:4:1:2)
  - Scatter policy for memory affine pinning (M:scatter) or interleaved memory policy
  - Logical pinning inside taskset/cgroup
- **Adjusts `OMP_NUM_THREADS` if not set**
- **Shepherd thread of OpenMP implementation not pinned**

Overview – likwid-pin (2)

- **Works with affinity domains:**
  - Node (N)
  - Socket (SX)
  - Shared L3 cache (CX)
  - NUMA node (MX)

```
Domain 0:
        Tag N: 0 28 1 29 2 30 3 31 4 32 5 33 6 34 7 35 8 36 9 37 10 38 [...]
Domain 1:
        Tag S0: 0 28 1 29 2 30 3 31 4 32 5 33 6 34 7 35 8 36 9 37 10 38 [...]
Domain 2:
        Tag S1: 14 42 15 43 16 44 17 45 18 46 19 47 20 48 21 49 22 50 23[...]
Domain 3:
        Tag C0: 0 28 1 29 2 30 3 31 4 32 5 33 6 34
Domain 4:
        Tag C1: 7 35 8 36 9 37 10 38 11 39 12 40 13 41
[…]
Domain 9:
        Tag M2: 14 42 15 43 16 44 17 45 18 46 19 47 20 48
Domain 10:
        Tag M3: 21 49 22 50 23 51 24 52 25 53 26 54 27 55
```

Overview – likwid-powermeter

- **How much energy is consumed by my application?**
- **Only for Intel CPUs >= SandyBridge**
- **Currently only coarse measurement domains**
  - PKG: CPU package
  - PP0: CPUs + internal GPU
  - PP1: Uncore
  - DRAM: Memory system and RAM modules

  Each CPU architecture has own domain subset
- **WARNING: Counters overflow ≈ 60 seconds**

```
likwid-powermeter -c 0,1 ./a.out
```

HPM - Hardware Performance Monitoring

- **Performance counters implemented in hardware**
- **HPM started for X86 with Intel Pentium**
- **Available for other archs like Sparc, POWER and ARM**
- **Variety of low-level data of CPU‘s functional units, cache and memory information**
- **Problems:**
  - HPM registers only accessible at OS level
  - Partly not accurate (“Flopgate” starting with SNB)
- **Each architecture defines:**
  - Own control & counter and management registers
  - Own set of events

likwid-perfctr – Wrapper mode

- Simple end-to-end measurements
- Likwid-perfctr sets up system topology and perfmon
- Start and stop HPM
- Executes user-given application on given cpu set
- Evaluate counter values and derive metrics

```
likwid-perfctr -c S0:0-7 -g FLOPS_DP ./a.out
```

CPUs 0 to 7 on Socket 0
(-C for pinning)

Double precision FLOPs perf. group

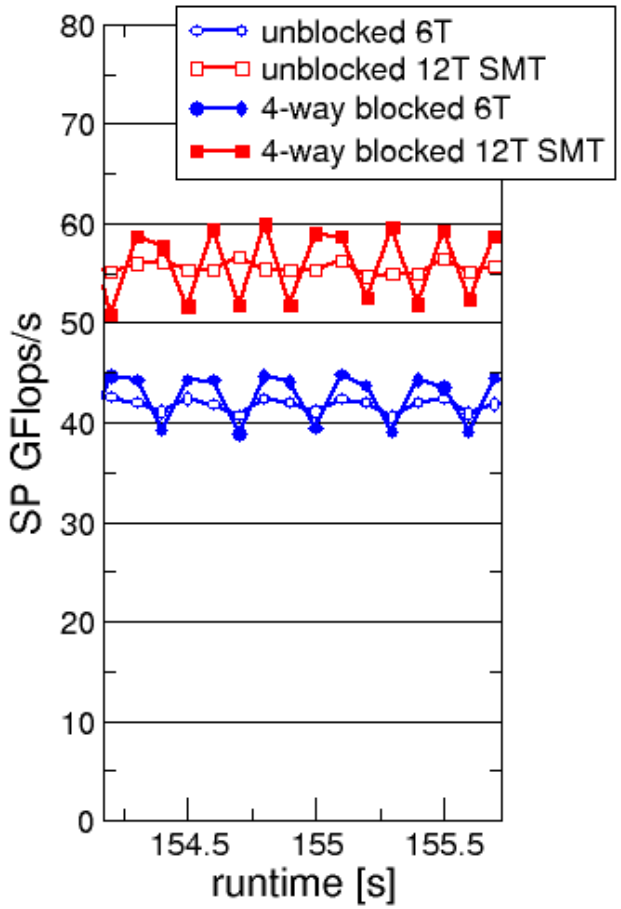
`likwid-perfctr -a`
for all available groups

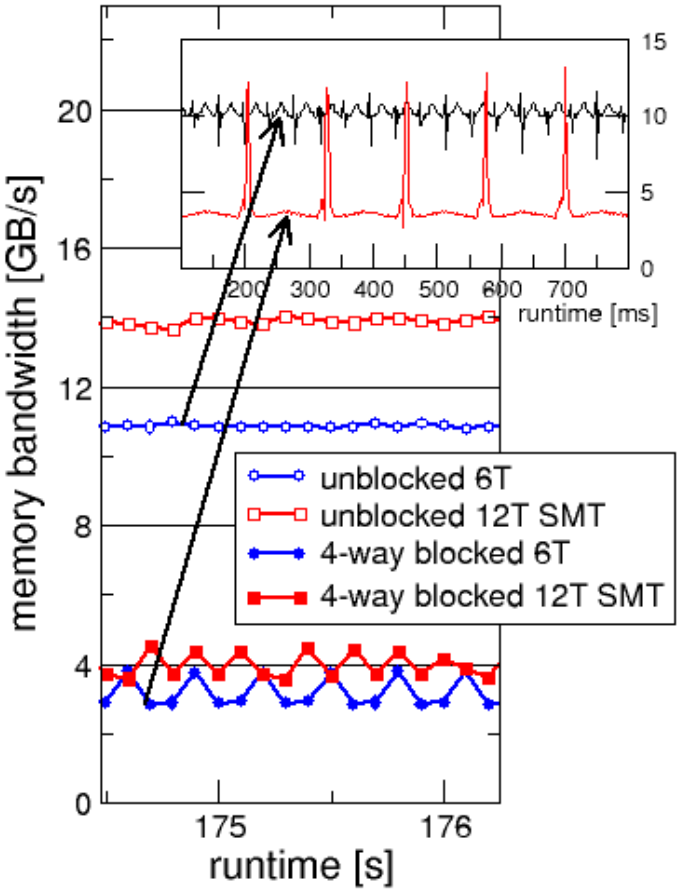
`likwid-perfctr -e`
for all available events and counters

likwid-perfctr – Timeline mode

- likwid-perfctr supports time resolved measurements:

```
likwid-perfctr -c 0 -g MEM -t 50ms
```





likwid-perfctr – Marker API mode

- To measure only parts of an application a marker API is available.
- The configuration of the counters is still done by `likwid-perfctr` application.
- Multiple named regions can be measured
- Results on multiple calls are accumulated

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

LIKWID_MARKER_START("Compute");
<code>
LIKWID_MARKER_STOP("Compute");

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

```
likwid-perfctr -C 0 -g FLOPS_DP -m ./a.out
```

How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**

```
#define N 10000 // large enough to use memory
#define ROUNDS 10
void fillMatrix(double* matrix, int size, double value) {
    for (int i = 0; i < size; i++)
        matrix[i] = M_PI;
}
// 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;}
    }
}
```

How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**

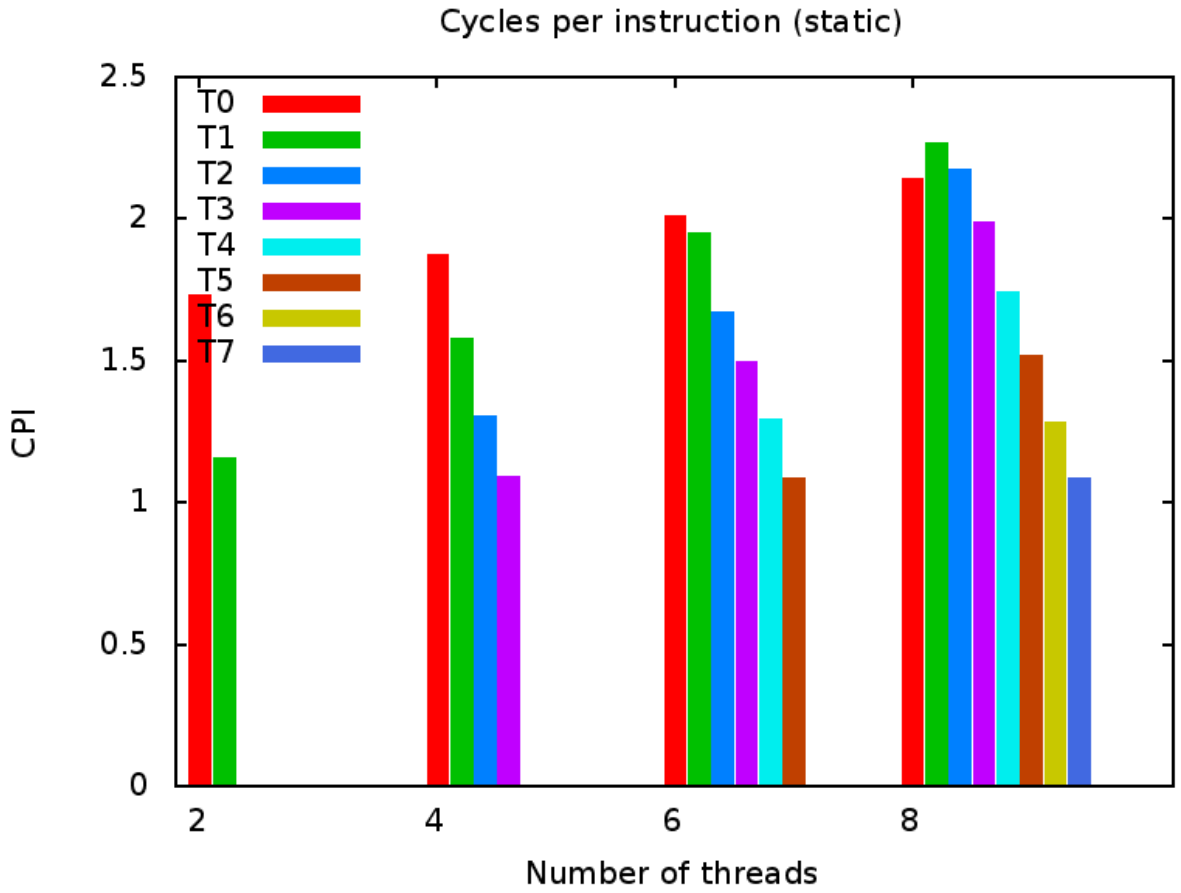
```
#include <likwid.h>
[…] // defines, fillMatrix and init matrix & vectors
LIKWID_MARKER_INIT;
#pragma omp parallel
{
    LIKWID_MARKER_THREADINIT;
}
#pragma omp parallel
{
    for (int k = 0; k < ROUNDS; k++) {
        LIKWID_MARKER_START("Compute");
        #pragma omp for private(current,j)
        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;
```

How do things work with LIKWID

```
$ likwid-perfctr -C 0,1,2 -g L2 -m ./a.out
--------------------------------------------------------------------------------
CPU type:       Intel Core SandyBridge EN/EP processor
CPU clock:      3.09 GHz
--------------------------------------------------------------------------------
======================
Group 1: Region Compute
======================
+-------------------+---------+---------+---------+
|    Region Info    |  Core 0 |  Core 1 |  Core 2 |
+-------------------+---------+---------+---------+
| RDTSC Runtime [s] | 0.161382 | 0.161365 | 0.161365 |
|     call count    |    10    |    10    |    10    |
+-------------------+---------+---------+---------+
+----------------------+--------+-------------+-------------+-------------+
|         Event        | Counter |    Core 0    |    Core 1    |    Core 2    |
+----------------------+--------+-------------+-------------+-------------+
|   INSTR_RETIRED_ANY   |  FIXC0  | 2.626800e+08 | 3.187585e+08 | 3.780255e+08 |
| CPU_CLK_UNHALTED_CORE |  FIXC1  | 4.972802e+08 | 4.961411e+08 | 4.933711e+08 |
|  CPU_CLK_UNHALTED_REF |  FIXC2  | 4.972801e+08 | 4.961404e+08 | 4.933714e+08 |
|    L1D_REPLACEMENT    |   PMC0  | 5.490278e+07 | 3.927353e+07 | 2.364295e+07 |
|      L1D_M_EVICT      |   PMC1  | 2.920200e+04 | 2.876600e+04 | 2.861000e+04 |
|     ICACHE_MISSES     |   PMC2  | 4.649000e+03 | 4.984000e+03 | 5.321000e+03 |
+----------------------+--------+-------------+-------------+-------------+
```

How do things work with LIKWID

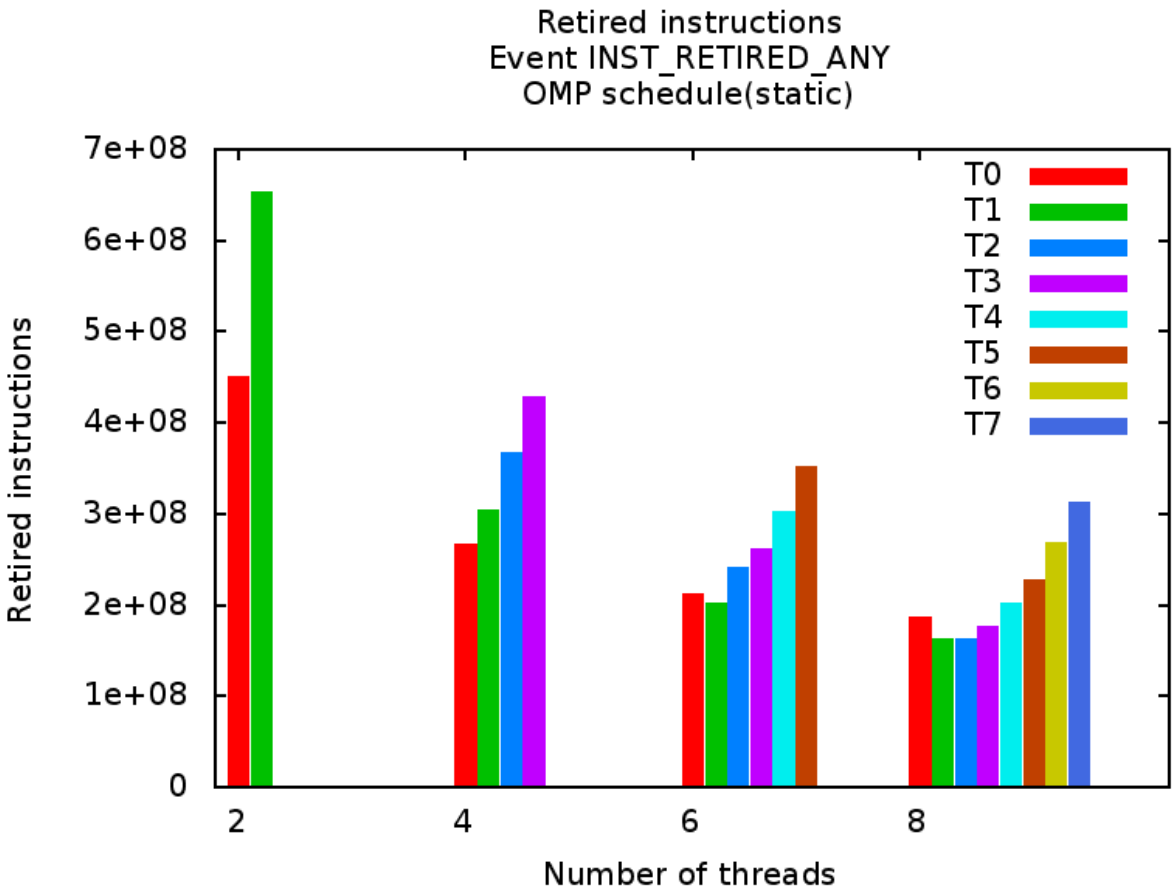
- **Multiplication of upper triangular matrix with vector**
  - #pragma omp for not enough. Work imbalance between threads
  - Common used metric Cycles-per-Instruction not reliable



CPI:
Lower is better

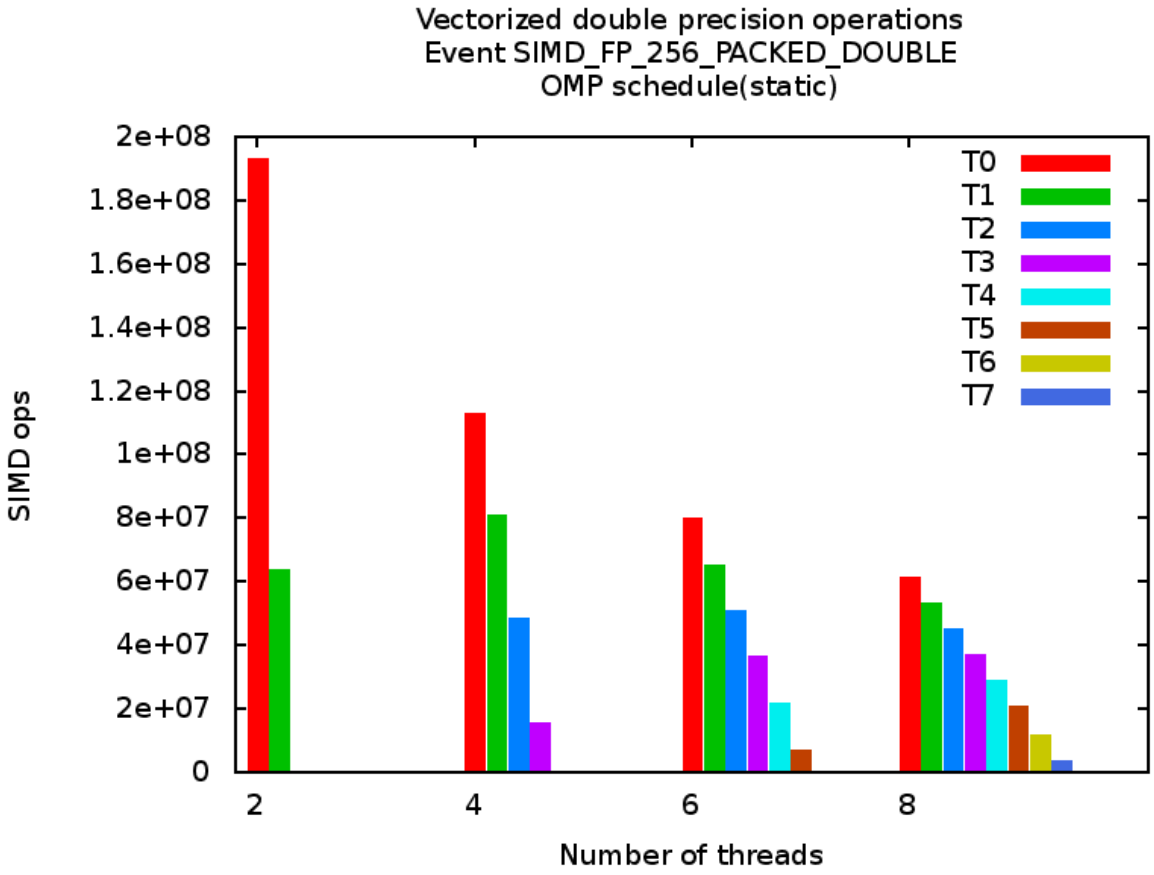
How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**
  - Retired instructions missleading
  - Waiting in implicit OpenMP barrier issues many but short instructions



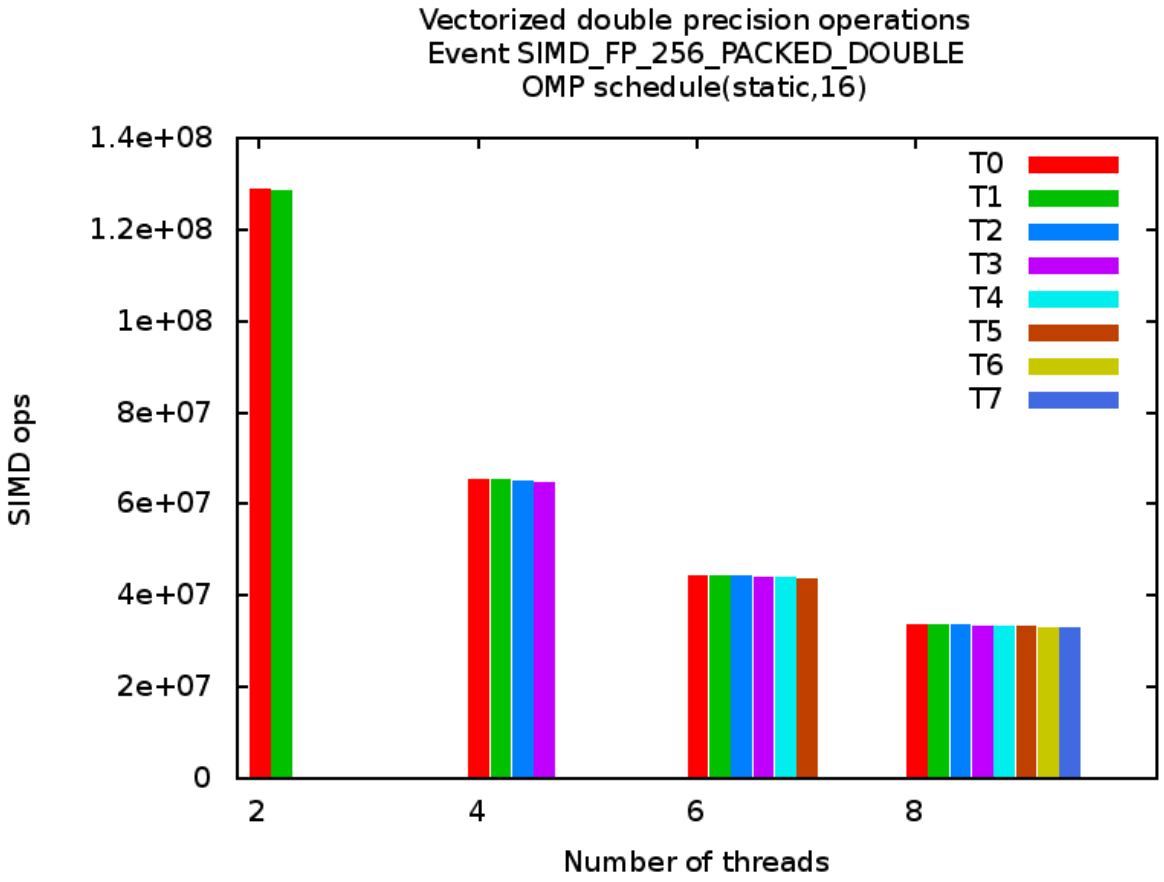
How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**
  - Floating point instructions reliable ↔ useful work metric
  - But floating point instr. counters since SandyBridge only tendentially correct



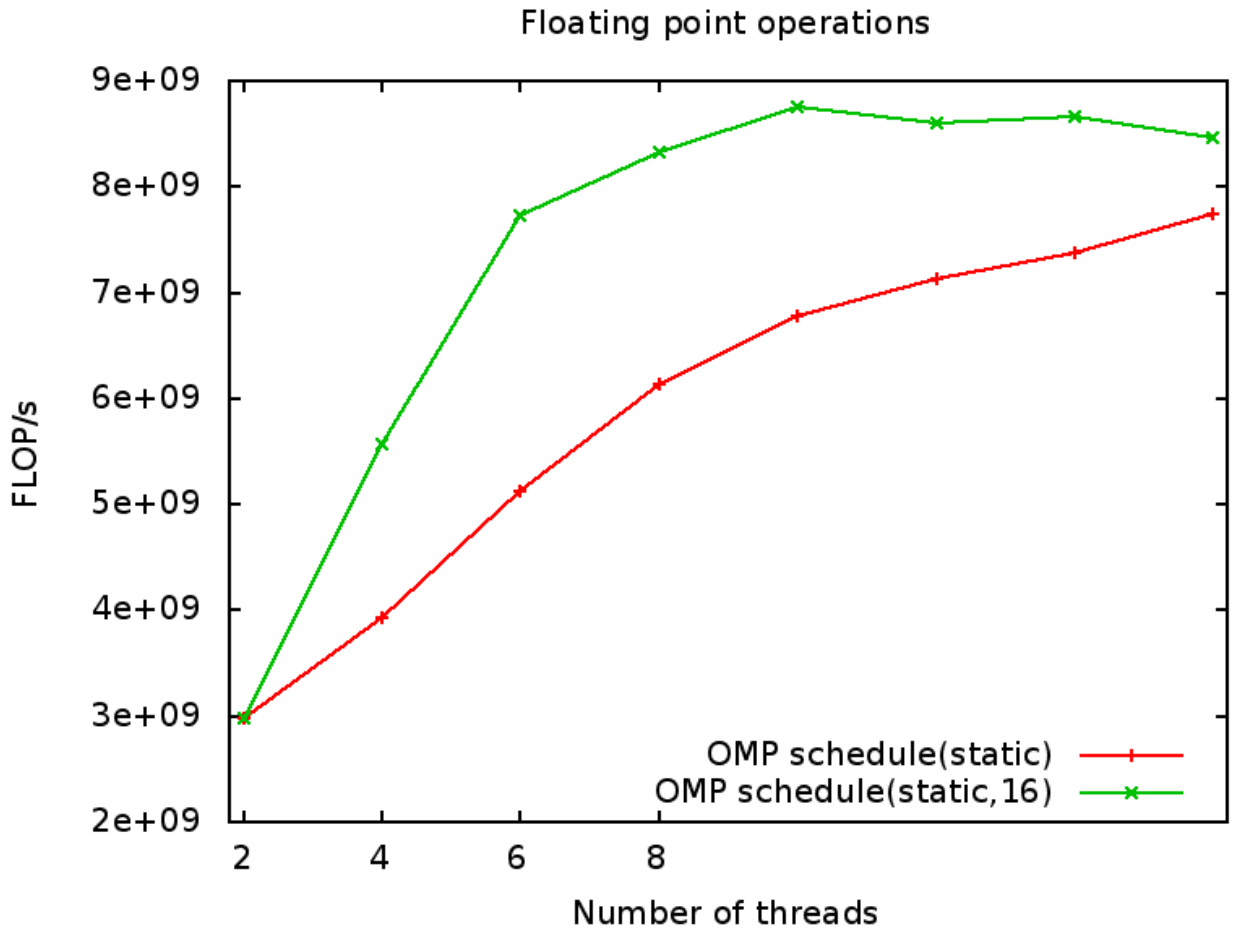
How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**
  - Changing static schedule to static with chunk size 16
  - But floating point instr. counters since SandyBridge only tendentially correct



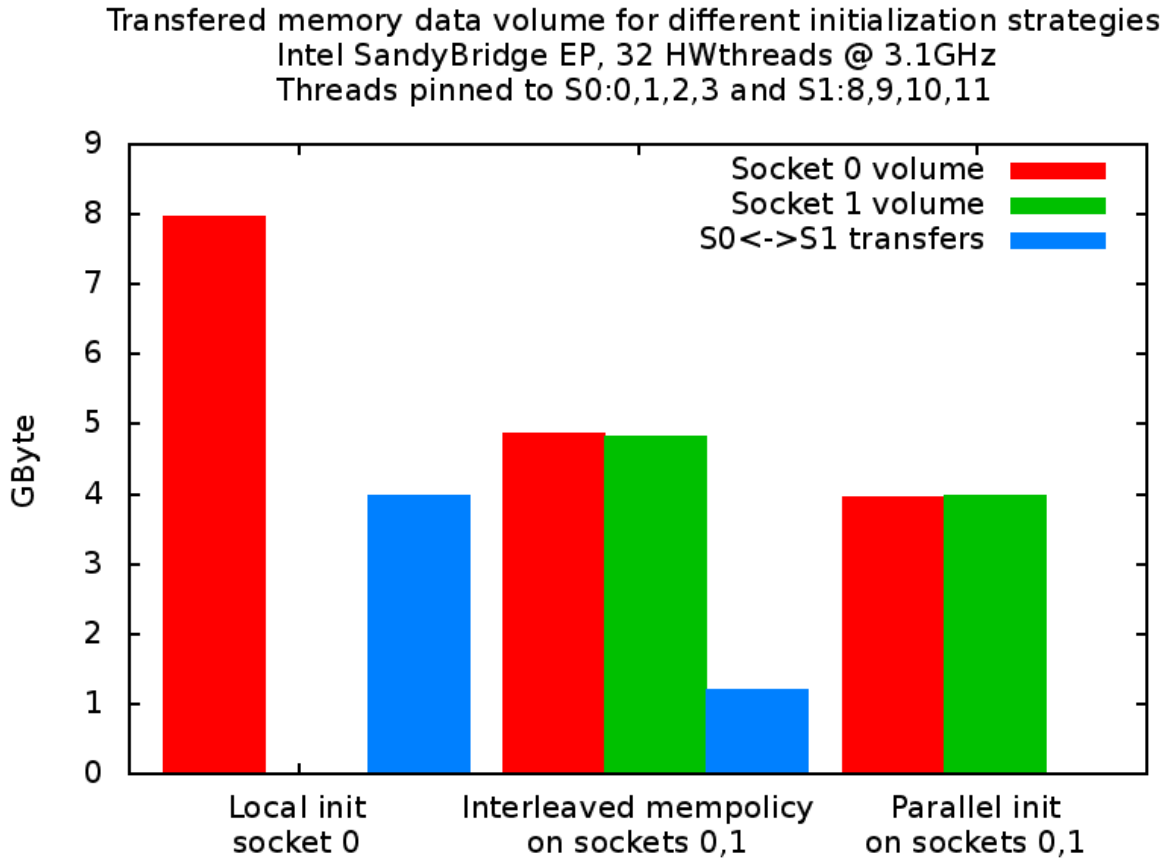
How do things work with LIKWID

- **Multiplication of upper triangular matrix with vector**
  - Comparing floating point operation performance shows benefit



How do things work with LIKWID

- **Multiplication of dense square matrix with vector**
  - Serial initialization not beneficial, all data on one socket
  - Memory traffic limits performance



WHAT‘S COMING?

…in the next versions?

Version comparison

Function	LIKWID 3	LIKWID 4
Topology code	CPUID, Procfs	Hwloc, CPUID, Procfs
HPM events	<event>:<counter>	<event>:<counter>:<options> Filtering/Matching/Refining
Uncore support	Static initialization to zero 548 events (IvyBridge EP)	Dynamic initialization 1068 events (IvyBridge EP)
Overflow handling	No	Yes, Core + Uncore
Multiplexing	Experimental	Yes but no interpolation
Overhead	86 Writes at Init	1 Writes at Init
RDPMC	Starting 3.1.3	Yes
Accuracy	Measurements partly include LIKWID code	LIKWID code excluded
API functions	8	60 C-API, 71 Lua-API
Performance Groups	Compiled in 182 groups	Dynamic reading 199 groups

Thanks for your attention!

Any Questions? Opinions?
Feature requests?

Webpage:

`http://code.google.com/p/likwid/`

User Mailing List:

`https://groups.google.com/forum/#!forum/likwid-users`

Bundesministerium für Bildung und Forschung

FEPA

Kennzeichen 01IH13009

What would be nice? (For discussion)

- **Graphical interface?**
- **Further reduce overhead using own kernel module?**
  - Batch read/write operations
  - MSR and PCI
- **Store each measurement phase separately for evaluation?**
- **Output Timeline mode data:**
  - after # Instructions?
  - after # UOPS?
- **More system topology information? TLB, Branch predictors, Dispatch Ports, IPMI ?**

Not planed:

- **Virtual counters**
- **Generalized events**