Node-Level Performance Engineering and Analysis

Jack Ogaja
jack.ogaja@gwdg.de

8. April 2024

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Learning Objectives

- To help develop ideas on how to use performance tools to explore the optimization space of widely used computational kernels in common computer architectures.

Multi-core, Multi-socket Complex Systems

Node-Level Performance Engineering and Analysis

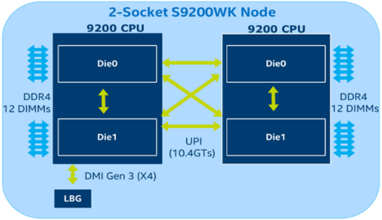
LIKWID Toolset

Multi-core, Multi-socket Complex Systems

Multi-core, Multi-socket Systems

Overview of modern complex systems with multiple NUMA domains.

Intel® Xeon® Platinum 9200 Processor Overview

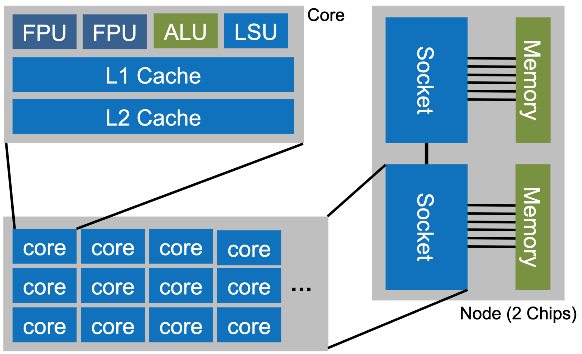


- Intel® Xeon® Platinum 9200 Processors consist of two die in a BGA package
- Multi-chip processor with single hop latency for any of the CPU die to memory in a 2S node
- Key IO/mem features include:
  - 12 ch DDR4 2933 MT/s per CPU
  - 4 UPI x20 wide at 10.4GTs per CPU
  - x80 PCIe G3 lanes per 2S Node in Intel® Server Systems S9200WK*

* Intel® Server Systems S9200WK supports 2 x16 Gen3 slots (per 1U node); 4 x16 Gen3 slots (per 2U node)

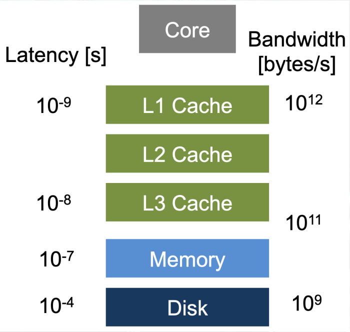
Example Node Topology

Modern systems are characterised by computational units with complex topology of tens of cores.



Memory Hierarchy

Most optimizations require data load from fast memory layers



- `Cacheline` (CL): Smallest unit of data that can be transferred with the memory hierarchy. (64Bytes in X86)
- `Cacheline states` :
  - Modified (dirty): CL is only in current cache and dose not match main memory
  - Exclusive (clean): CL is only in current cache and matches main memory
  - Shared (clean): CL is unmodified and may be stored in other caches
  - Invalid: CL is unused
- `Cacheline evict` : A cache miss triggers a CL allocation. The replaced CL is evicted from the cache

Memory Bandwidth

MEM node \ CPU node	0	1	2	3	4	5	6	7
0	32.4	21.4	21.8	21.9	10.6	10.6	10.7	10.8
1	21.5	32.4	21.9	21.9	10.6	10.5	10.7	10.6
2	21.8	21.9	32.4	21.5	10.6	10.6	10.8	10.7
3	21.9	21.9	21.5	32.4	10.6	10.6	10.6	10.7
4	10.6	10.7	10.6	10.6	32.4	21.4	21.9	21.9
5	10.6	10.6	10.6	10.6	21.4	32.4	21.9	21.9
6	10.6	10.7	10.6	10.6	21.9	21.9	32.3	21.4
7	10.7	10.6	10.6	10.6	21.9	21.9	21.4	32.5

- Example: Bandwidth measurements [Gbytes/s] for a 2-socket system (8 chips, 2 sockets and 48 cores)

GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Node-Level Performance Engineering and Analysis

Node-Level Performance Engineering and Analysis

- Modelling: Derivation of a model based on the functionality and topology of interconnected elements of a computational unit of a specific architecture.
- Measurements: Collection of events data through program instrumentation and events sampling.
- Visualization: Usage of performance tools to visualize collected events' data and traces.

Node-Level Performance Engineering and Analysis

Modelling

Performance models are important in application's performance engineering and analysis. Models are key for:

- Comparing application performance against the machine capabilities
- Evaluating the optimality of application
- Identify possible bottlenecks in application computational performance
- Identifying software and hardware limitations

Node-Level Performance Engineering and Analysis

Measurements: Machine and Application Characterizations

1. Data Collection and Sampling
   - Automatic instrumentation - increases overhead, e.g. Compilers, Score-P,
   - Manual instrumentation. e.g. Print-statements, Score-P
   - Binary instrumentation - requires re-addressing, replacements and patching of instructions and memory accesses, e.g. Gprof, Valgrind, GDB
   - Sampling - execution is interrupted at regular intervals to sample addresses of executed instruction, e.g. LIKWID, Gprof
2. Data Processing
   - For simple applications with small amount of events, events can be counted and performance data can be processed and displayed in a graphical viewer in real-time.
3. Data transfer and storage
   - For complex applications, events data are stored in disks, e.g. Score-P + Vampir

LIKWID Toolset

LIKIWD

- LIKWID: A toolset for performance-oriented developers and users:
  - `likwid-topology` : Probes system (thread/core/cache/NUMA) topology
  - `likwid-pin` : Pin threads to cores according to system's topology (for maintenance of spatial locality)
  - `likwid-bench` : Provides a set of micro-benchmark kernels including stream, triad and daxpy, to check system features as *FLOPS*, bandwidth and vectorization efficiency.
  - `likwid-perfctr` : Measure hardware events during application runs and show derived metrics including *FLOPS*, bandwidth, TLB misses and power. integrates the `likwid-pin` functionality.
  - `likwid-mpirun` : MPI wrapper for `likwid-pin` and `likwid-perfctr` . Profiles MPI and Hybrid applications. Utilizes `likwid-pin` and `likwid-perfctr` at the backend.

LIKWID: Topology

Check the options using `likwid-topology -h`

```
$ module load likwid
$ likwid-topology -h
likwid-topology -- Version 5.2.0

Options:
-h, --help       Help message
-v, --version        Version information
-V, --verbose            Set verbosity
-c, --caches         List cache information
-C, --clock      Measure processor clock
-O           CSV output
-o, --output             Store output to file. (Optional: Apply text filter)
-g           Graphical output
```

LIKWID: Affinity

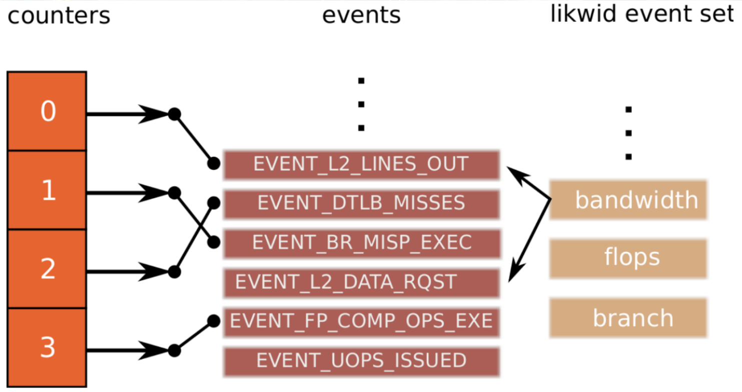
- Provides thread-to-core pinning for an application for maintenance of spatial locality.
- `likwid-pin` accepts 6 options for processor lists:
  1. **physical numbering**: processors are numbered according to the numbering in the operating system
  2. **logical numbering**: processors are logically numbered over the whole node - N
  3. **logical numbering in socket**: processors are logically numbered in every socket - S#
  4. **logical numbering in cache group**: processors are logically numbered in last level cache group - C#.
  5. **logical numbering in memory domain**: Processors are logically numbered in NUMA domain - M#
  6. **logical numbering within cpuset**: processors are numbered inside Linux cpuset - L

LIKWID: Hardware Performance Counters 1/2

- Uses the Linux msr module to access the model specific registers stored in `/dev/cpu/*/msr` then calculates performance metrics, *FLOPS*, bandwidth, etc, based on the formula defined by LIKWID or customized by user.
- `likwid-perfctr -a` lists performance metrics and/or groups supported by LIKWID
- `likwid-perfctr -e` lists all hardware events or counters available
- `likwid-perfctr -E <perf group>` shows the events or counters used to calculate a particular performance group.
- `likwid-perfctr -H -g <perf group>` reveals the formula being used to derive performance metrics using the performance counters.

LIKWID: Hardware Performance Counters 2/2

Interaction between event sets, hardware events and performance counters.



LIKWID: MPI wrapper

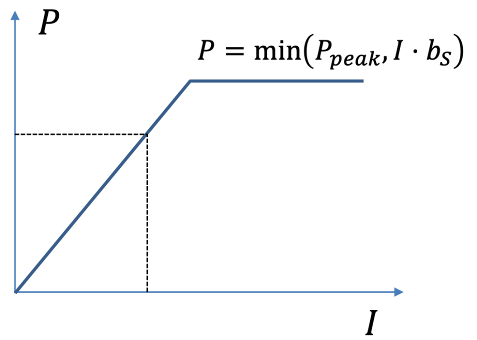
- Detects MPI environments and wraps a job launcher around `likwid-perfctr` to measure performances for MPI and hybrid applications.
- It also integrates the functionality of `likwid-pin`

LIKWID: Micro-benchmarking

- Provides a list of benchmark kernels for users to quickly test some characteristics of an architecture.
- A number of basic benchmark kernels are readily available:
  - `copy` Standard memcpy benchmark. $A[i] = B[i]$
  - `copy_mem` The same as above but with non temporal store.
  - `load` One load stream. This one does some software prefetching you can experiment with.
  - `store` One store stream.
  - `store_mem` The same as above but with non temporal store.
  - stream Classical STREAM triad. $A[i] = B[i] + aC[i]$
  - `stream_mem` The same as above but with non temporal store.
  - `triad` Full vector triad. $A[i] = B[i] + C[i] * D[i]$
  - `triad_mem` The same as above but with non temporal store

Emperical Roofline Model with LIKIWD

Roofline Model: A visually-intuitive graphical representation of a machine's performance (P) characteristics considering two principal performance bounds, computation and communication.[1]



- Memory bandwidth, b_s: Communication is bounded by the characteristics of the machine's processor-memory interconnect.
- Arithmetic Intensity, I [flops:bytes]: The ratio of kernel's computation to memory traffic (volume of data to a particular memory).

[1] Samuel W. Williams. **Auto-tuning Performance on Multicore Computers.** Berkley: University of California at Berkley, 2008.

Tutorial

- Download the exercise
- Login to Scientific Compute Cluster (SCC)
- Load LIKWID module
- Use likwid to measure performance and optimize the given code.

Note: Use slurm to start an interactive session in a compute node.