Memory Bandwidth Monitoring in Linux for HPC Applications

Kanaka Juvva
Linux Con North America, Seattle
August 2015
Outline

- Objectives of Memory Bandwidth Monitoring
- Intel Xeon’s Shared Resource Monitoring Technology
- Implementation in Linux
- User Interface
- Case Studies and Applications
- Summary
MBM Objectives
(Why Monitor Memory Bandwidth ?)

- Number of cores per socket continue to increase
  - Moore’s Law
  - Huge degree of concurrency and parallelism for user space programs
    - Workload is split among multiple cores and sockets: e.g. DPDK
    - Socket Scalability for CPU Bound
    - Memory accesses will impact the ultimate performance
      - Higher the concurrency, more Memory BW
  - More Levels of Multi-core cache Hierarchies
    - Last Level Cache (LLC) and Memory is shared across all cores on a Die

- Dynamically watch the two most contended resources - Cache and Memory BW Utilization
  - Orchestrate Data Movement
  - Reduce Memory Transfers and Optimize NUMAs
MBM Objectives Contd...

- **Manage High Performance Applications**
  - Data Center, Cloud, and Extreme Science

- **Data Center**
  - Virtualized IT services drive new levels of resource utilization
  - Continually changing business demands
  - Any workload delivered anywhere - Migration

- **Cloud**
  - Alibaba: E-commerce
  - Baidu: Data Management
  - TenCents: Data Analyze

- **Extreme Science**
  - Medicine
  - Scientific Computing
Intel® Xeon® processor E5-2600 v4 product family’s built-in Platform Resource Monitoring support
- Shared resource monitoring: L3 and Memory BW
- Track L3 Occupancy and bandwidth out to the next level of the hierarchy, e.g., Memory BW treated as external BW of cache resource
- OS/VMM can track the memory BW usage similar to cache QoS Monitoring
- Identify memory bandwidth conflict issues and enable application migration
- Monitoring at different levels of granularity
  - E.g., VM, Workload, Process and Thread

Microarchitecture Features:
- Cache Monitoring Technology
- Memory Bandwidth Monitoring

Intel QuickPath Interconnect:
- 2 QPI 1.1 channels

Cores Per Socket: Up to 22
Threads Per Socket: Up to 44

Last Level Cache (L3): ~55MB
Intel Xeon’s Shared Resource Monitoring Technology

- Resource monitoring capability in each logical processor i.e. core. Two monitoring technologies are supported.
  - Cache Monitoring Technology (CMT): Allows an OS, Hypervisor or VMM to monitor the usage of cache by applications
  - Memory Bandwidth Monitoring Technology (MBM): Allows OS, Hypervisor or VMM to monitor bandwidth from one level of cache hierarchy to the next – L3 cache which is backed directly by system memory
  - Either of them or both features can be present in a given CPU model. Have some common sub-features.
- OS can check the presence of the above features via a CPUID feature bit
- Details of each sub-feature via CPUID leaves and sub-leaves
- A mechanism for the OS or VMM to set a software-defined ID known as Resource Monitoring ID (RMID) for each software thread
- A mechanism to monitor cache and bandwidth stats per RMID basis
- OS can Read back the collected stats such as L3 occupancy and Memory BW for an RMID during runtime
Resource Monitoring ID (RMID)

- RMIDs abstract an application, thread, or cgroup. For each logical processor, only one RMID is active at a given time.
- IA32_PQR_ASSOC MSR specifies the active RMID of a logical processor. Writing to this MSR by OS changes the active RMID of the logical processor from an old value to a new value.
- Max #RMIDs is model specific and is detected run time via CPUID sub-leaves feature.
- Sharing of RMIDs: same RMID can be used to monitor different event types.

The resource monitoring hardware tracks cache utilization and misses as a result of memory accesses per the given RMID and reports monitored usage via an MSR i.e., IA32_QM_CTR.

Different event types are to distinguish the resource metric
- e.g. LLC_OCCUPANCY, TOTAL_BW, LOCAL_BW
- Software configures the event selection MSR (IA32_QM_EVTSEL) to specify the event or metric to be reported.

RDMSR and WRMSR instructions are used to read and write the monitoring MSRs.
Usage Flow

Resource Association
- On Context Switch*
- RMID (for resource tracking)
- PQR Register

Resource Marking
- Application Memory Request
- Tag with RMID
- Memory Request

Resource Reporting
- Per Application Monitoring Request
- Event select for RMID
- Read monitoring counter
- Event request

Cache Subsystem
- Resource Monitoring
  - 1 Monitoring counters
  - 2 Monitoring counters
  - 3 Monitoring counters
  - 4 Monitoring counters

Example: LOCAL BW/TOTAL BW

PQR RMID identifies application for QoS resource-aware cache unit
Implementation

- Feature Detection at Runtime
  - Intel defined CPUID LEAF and QoS Sub-leaf

- RMID assignment Module
  - Per thread/application/VM

- High Resolution timer for each logical processor

**On OS/VMM Initialization**

1. Feature Detection: using CPUID
2. Create HR Timers
3. Initialize Sliding Window

**On context switch**

1. RMID Assignment:
   - Set RMID for the scheduled Application

**Periodically measure BW**

1. On Timer Callback:
   - Read MSR and Calculate BW

**MSRs**

- CPUID
- IA32_PQR_ASSOC MSR
- IA32_QOS_EVTSEL MSR (Res, RMID)
- IA32_QM_CTR MSR (data)
Implementation Contd..

- Read MSR periodically on timer callback and do the processing
  - Check for Counter Overflow and handle it
  - Calculate Local BW: BW Consumed on same socket
  - Calculate Total BW: BW Consumed on same socket + QPI BW for NUMA

- Store the monitoring samples in a sliding window
  ```
  struct sample {
      u64 bytes;       /* mbm counter value read*/
      u64 cum_avg;     /* current running average of bandwidth */
      ktime_t prev_time;
      u64 index;       /* current sample no */
      u32 mbmfifo[MBM_FIFO_SIZE_MAX]; /* sliding window for last ‘n’ bw values */
      u32 fifoin;      /* mbmfifo in counter for sliding window */
      u32 fifouout;    /* mbmfifo out counter for sliding window */
  }
  ```
  - configurable window duration: (10sec - 300sec)

- Calculate Running Average of BW for the whole window duration
User Space APIs

- **Perf API**
  - `perf_event_open()`
  - `struct perf_event`
  - **Performance Monitoring Unit**
    - Fill and Register `struct pmu`
    - Call back functions to start, read and stop stats

- **Set the sliding window duration. Default is 10sec**

- **Two perf events are exported to userland**
  - `LOCAL_BW`
    - `perf stat -e intel_cqm/llc_local_bw/ -a "my_application"`
  - `TOTAL_BW`
    - `perf stat -e intel_cqm/llc_total_bw/ -a "my_application"`

- **Work in progress**
  - User Space Libraries
  - Integration with Cloud and Data Center
    - Virtualization Libraries to monitor containers
Case Studies and Applications

➢ Profiling a Data Center

Without Memory Bandwidth Monitoring

- High BW App 1
- High BW App 2
- Low BW App 1
- Low BW App 2

With Memory Bandwidth Monitoring Through BW Aware Scheduling

- High BW App 1
- High BW App 2
- Low BW App 1
- Low BW App 2

Balanced utilization
Case Studies and Applications Contd....

- **Memory Modeling** for Molecular Medicine e.g. CT image processing in 1 mt for $10
  - HPC using Multi-Socket and Multi-Core: Measure BW utilization per Socket, per Core, NUMA BW, and Module Level BW - Utilization i.e. VM, Service, and Workload
  - Memory Hierarchy Optimizations: find saturation point for memory concurrency and #concurrent cache misses
  - Micro-architectural Analysis of Software Stack and NUMA
Summary

- Memory Bandwidth Resource Monitoring Scheme
- Intel Xeon Microarchitecture Feature
  - Detect and Enumerate the feature during runtime
- Patches available for Linux 4.x
  - [https://lkml.org/lkml/2015/7/21/893](https://lkml.org/lkml/2015/7/21/893)
- Benefits wide variety of Applications
  - Flexible APIs for user space
  - Cloud
  - Data Center
  - Scientific Computing
- Future Work
  - Frameworks for Alibaba, Baidu, Tencents, and Openstack