Multiprocessor Scheduling

Ampili Jayaprakash
Anand Venkat

The Issue

- Scheduling on Multiprocessors
  - Large variety of shared memory architectures
  - Traditional Uniprocessor Scheduling strategies not efficient on these systems
  - Operating System Schedulers unaware of underlying shared memory organization

The Issue

- Scheduling on Multiprocessors
  - How to efficiently utilize all nodes for computation?
  - How to maintain fair resource allocation in the new scenario?

Scheduling

- Goals
  - Minimization of Inter–Process Communication and Synchronization Overhead
  - Minimization of Application Partitioning Overhead – Task Allocation
  - Achieving the Optimal Partitioning
Dynamic Scheduling

- Dynamic/Job Scheduling - not decidable a priori
- Requires run time scheduling. On-the-fly decisions taken according to the state of the system
- Has to take the cost of scheduling overhead into account

What is Hyper-threading?

- Intel's term for simultaneous multi-threading.
- The first commercial implementation of SMT was in Intel's Pentium 4 processor.
- Common in servers and desktops.

Hyper Threading

How does Hyper-threading work?

- Hyper-threading requires only that the operating system supports multiple processors.
- For each physical core present, the operating system addresses two virtual processors and shares workload between them.
- Duplicates the sections of the processor that store the architectural state.
How does Hyper-threading work?

- Each logical processor executes one heavy weight thread.
- Usually transparent to the OS and applications.
- OS awareness is necessary in some situations:
  - Two physical processor system having two processes running on the two logical processors of one package while the other package remains idle.

Performance Estimation

- Processes sharing a physical processor, run slower than they would do if they had exclusive use of the processor.
- Increased system throughput at the expense of individual process’ throughput.
- Throughput is expressed by a performance ratio.
- Performance ratio: Rate of execution under hyper-threading versus rate of execution when the process has a processor to itself.

Performance Metrics

- The performance metric used is speedup.
- Speedup is the sum of the performance ratios of the pairs of processes.
- Speedup is improved by using hyper-threading aware schedulers.

Example: Process A takes 60 seconds to execute in the exclusive mode and 100 seconds to execute with another process B in hyper-threading mode.

Performance ratio = 0.6 Assuming performance ratio of process B also as 0.6, the system speedup = 1.2.

Therefore, a 20% speedup is achieved due to Hyper-threading.

Hyper-threading aware process scheduling heuristic

- Goal: Introduce process awareness to the scheduler.
- Concept developed by James Bulpin and Ian Pratt.
- Existing scheduler was modified to create process awareness.
- Implementation of two new scheduling algorithms:
  - Tryhard
  - Plan
Try-hard Algorithm

- Biases the goodness of a candidate process by how well it has performed previously with the process on the other logical processor;
- Uses a hash table that contains the recorded estimates of the pair of PIDs of the candidate process and the process currently running on the other logical processor.

Plan algorithm

- Uses a userspace tool to process performance ratio data. Produces a scheduling plan.
- For every process ‘p’, plan records three other processes that have the highest estimated speedups when running simultaneously with ‘p’.
- A userspace tool reads this data periodically and selects pairs with the highest system speed-up.
- Scheduler heavily biases goodness in favour of the process pair that produces the best speed-up.

System speedup

- Performance metric: System speedup, i.e. sum of the performance ratios of the two processes running on the logical processors.
- For measurement purposes, pairs of SPEC CPU2000 benchmark applications were executed in an infinite loop on each logical processor.
Fairness and starvation avoidance

- Any processes not in the plan or those created after the planning cycle will run once the processes in the plan have blocked or have exhausted their time slice.
- Process time slices are preserved so that no starvation occurs.

Hardware Architecture exhibits a CMP–SMP–SMT organization – Implications?
- Threads that heavily share data not located on the same chip
- Sharing of memory (cache) resources is non-uniform
- Data required by a thread on a particular core might have to be brought in from a remote cache.
**Goal**

- Minimization of inter processor thread crossovers
  - Motivation – Better overall system performance
  - Cross Chip Cache accesses are typically an order of magnitude slower than local cache accesses
  - Clustering – Threads that exhibit similar data access patterns are scheduled to run on the same processor

**Methodology**

- 4 Step Algorithm
  - Monitoring stall breakdown - HPCs are used to analyze stall cycles and when cross chip communication is performance limiting
  - Detecting Sharing Patterns – using a structure called the shMap, that provides a signature of data regions accessed for all threads.
  - Thread Clustering – where the the shMap is analyzed and similarity in threads’ data sharing patterns is detected and such threads are clustered
  - Thread Migration – The OS scheduler migrates threads of the same cluster onto the same node.

**shMap**

- Address space of the system is divided into a number of regions
  - Each Region has an associated counter in the shMap
  - Each Thread has a vector of counters
  - The counter is incremented each time there is a remote access to the corresponding region from a thread
Data Collection and Analysis

- Sampling of cache in time and space
  - Spatial Sampling carried out using a shMap filter that has a one to one correspondence with the shMap vector.
- Threads' data access profiles are compared using a similarity metric
  - Essentially a dot product of pairs of shMap count vectors

Algorithm

- Scan through all available clusters
  - If no clusters match the thread then a new cluster is created.
  - Complexity of Algorithm is $O(Tc)$ where $T$ is the number of threads suffering from remote cache accesses and $c$ the total no. of clusters
- Clusters are ordered in descending cluster size
- The largest cluster is scheduled onto the processor with the least no. of threads
  - If Imbalance is detected then threads in cluster are evenly assigned to each chip
  - This step is iteratively done for each cluster

Results

Application Reported Performance improvements of upto 7%
Thread clustering helped reduce stalls due to remote cache accesses and improved performance by 5% by removing these stalls

Conclusion

- General Dynamic Scheduling Strategy
  - Choice of a metric that correlates to the entity under optimization
  - Eg. Measuring the rate of instructions committed using a H/W Performance counter to give an indication of overall system throughput
  - Dynamic Assessment of and actions taken based on the metric value
  - Threshold or critical region of metric value which activates the desired algorithm
  - Desired Algorithm – Hyper-threading or Thread-Clustering
References
