Behavior of Benchmarks on Commercial Server Systems

Tales from the Trenches

INFORMIX

Systems Used to Collect Information

- Processor Types:
  - DIGITAL Alpha
  - HP PA-RISC
  - Intel Pentium® Pro and Pentium® II Xeon™
  - MIPS
  - UltraSPARC™
- System Architectures
  - SMP
  - NUMA
  - Large and Small SMP clusters

Software and Benchmark Setup

- The software used for testing was Informix's relational database IDS AD/XP version 8.x.
- The TPC-D™ benchmark was used to generate the workload. At all times the workload was bound by at least one of the following characteristics of the benchmark.
  - CPU and memory intensive sorts, joins and aggregations.
  - Disk IO intensive, large sequential data scans.
  - Mixed Read/Write random disk IOs.

The Five Generic Problem Areas

- Optimization based on compilers and other software tools.
- Processor Optimizations.
- The Processor L2 Cache.
- Memory Management (including the OS).
- I/O Subsystem Architecture.

Compiler/Tool-based Optimization

- Typical comments from hardware vendor:

  "Why don't you compile with level 17 compiler optimization and secret random code manipulation? Use feedback data and turn on the latest instruction set flag even though the binary will not run on any hardware older than three months. By the way, with all the cool stuff, it might take seven days to produce a binary."

Compiler/Tool-based Optimization

- Commercial software profile:
  - Large foot print (5-15 megabyte binaries).
  - A single binary is generally built from large number of libraries developed by different organizations.
  - Large source base with multiple architecture support.
  - Multi-threaded and multi-process, internal schedulers and memory managers.
  - Occasional assembly language and considerable dependency on system calls.
- Scope of a single software function is not limited to a single source file or even a single set of libraries.

Compiler/Tool-based Optimization

- The reason we can’t always use the fancy compiler options is:
  - The compilation process can’t be a complicated research project.
  - We release on all hardware platforms multiple times per year.
  - The source changes constantly, bug fixes are applied frequently and maintenance releases are done regularly.

Compiler/Tool-based Optimization

- The reason we can't always use the fancy compiler options is (continued):
  - Releasing six different binaries for the same brand hardware is cost prohibitive.
  - Compilers are not well tested with complex software packages.
  - Last time I tried, it didn't work!
- When designing your processor, please don't assume a "smart compiler"!

Processor Optimization

- Commercial software is growing in size at a fast pace with no limits in sight.
  - Complex commercial software contains code for:
    - Dealing with error cases that may never happen.
    - Handling Exception cases.
    - Managing backward and forward compatibility.
    - Providing seldom used functionality.
  - It is not unusual to find 10,000 “if” statements in 100,000 lines of RDBMS code.

Processor Optimization

- What commercial applications need from the processor:
  - Better branch prediction and predication.
  - Speculative load/stores.
  - Better integer multiplication and division.
  - Variable register window with larger window size.
  - More integer registers (who needs floating point anyway?)

The L2 Cache - The Problem

- Code misses are one of the biggest performance problems in commercial software.
- Unlike SPEC benchmarks, commercial code has a very large foot print and is far more sensitive to code misses than data misses.
- For example, when processing large volumes of transient data, L2 text misses increase and performance is significantly impacted.
- Also, high IO rates also mean more execution in kernel-mode, which introduces conflict between application and OS.

The L2 Cache - Size Matters, But...

- Processors that support split L2 instruction and data caches or have high degree of cache associativity perform better, as the main sources of conflict are avoided.
- Reducing conflict also makes performance more consistent, as run-time page mapping determines the degree of conflict between the application and OS.
- Typically, a 2MB set associative cache will yield better performance than a 4MB direct-mapped cache.

The L2 Cache - A Solution

- It is not possible to write perfect software and compile it in to a packed set of frequently executed instructions that can fit in ten consecutive memory pages.
- **We need large, highly associative, split instruction and data L2 caches.**
- Although this point has been made before, some vendors still do not have products that reflect this knowledge.

Memory Management

- In the world of benchmark systems, 16GB of memory is now common. At the high end, benchmark systems are loaded with 64GB of memory.
- Large memory systems exist both in 32 bit and 64 bit architectures.
- RDBMS software makes extensive use of the shared memory model to manage huge volumes of in-memory data.

Memory Management

- The overhead of managing large shared memory segments by the OS must be reduced.
- Currently benchmark systems allocate large segments of shared memory and lock the segments in memory to reduce paging and other VM system overheads.
- In addition, locked segments provide other advantages such as large pages which can further reduce overhead.
- Without locked segments performance is non-existent.

I/O Subsystem: Requirements

- The IO requirements of a high performance system are:
  - The ability to sustain high number of IO operations.
    - 1000-1500 IO operations per average processor.
  - The ability to sustain high throughput.
    - 50-75 MB/sec per average processor.
  - Efficiently support IO sizes ranging from 2k-512k
- We exclude disk and disk array technology from this discussion.

I/O Subsystem - Architectural Issues

- In order to support the performance requirements, systems must be efficient throughout the entire IO path.
  - IO bus to system bus connectivity:
    - Impact on system bus during large IO transfers.
    - Impact on system bus with large number of outstanding requests.
  - IO controllers:
    - Impact of controller on IO bus.
    - Support for multiple IO controllers per IO bus.
    - 64 bit address support.
    - DMA to all available system memory (no copies please).

I/O Subsystem - Architectural Issues

- In order to support the performance requirements, systems must be efficient throughout the entire IO path (continued):
  - IO switches and bridges
    - Any external device on the IO path should be able to support the aggregate volume of IO for connected devices.
  - Hardware and OS Interrupt handling
    - The ability to distribute interrupts on all CPUs or consolidate them on one CPU as required by the configuration.

I/O Subsystem - Architectural Issues

- In order to support the performance requirements, systems must be efficient throughout the entire IO path (continued):
  - OS system call interfaces
    - Kernel support for asynchronous IO is a must. The OS must support large number of outstanding requests and provide an efficient poll/notification of completed requests.
- While these points appear straightforward, **most systems fail in one or more of the above areas!**

Summary

- For good performance on commercial benchmarks:
  - Don’t assume a “smart compiler.”
  - Optimize for a high rate of branching.
  - Provide large, set associative, split instruction and data caches.
  - Provide large pages sizes and the ability to pin shared memory segments.
  - Don’t neglect the IO subsystem.

Related Information

- For a complete description of the TPC-D™ benchmark please refer to http://www.tpc.org
- For detailed information about Informix IDS AD/XP database product please refer to http://www.informix.com.