MULTITHREADING AN INDUSTRY-STANDARD TELECOMMUNICATIONS APPLICATION

Optimization to take full advantage of the UltraSPARC® T2 processor

White Paper

January 2009

Abstract

This Sun Microsystems white paper is oriented toward a technical audience. It assumes the reader is familiar with the broad concepts of multicore/multithreaded microprocessors, and with UltraSPARC® T2 processor architecture.

Table of Contents

Chapter 1
Introduction

In recent years, the telecommunications industry has seen rapid expansion of its customer base, especially in the emerging economies of Asia and Latin America. The size of the added population served is in the hundreds of millions, in both fixed and mobile networks. These new subscribers demand complex service offerings, while at the same time being extremely price conscious and requiring rock-solid reliability in small form factors.

The solution has been to support the new users with converged services. For efficiency, these need to be based on a single access-independent unified platform that provides vastly greater performance. The increased throughput for these integrated networks has to be achieved without appreciable increase in power, space, heat dissipation, and, most important, equipment cost. This has led to high demand for consistent and reliable software stacks running on highly efficient blades. These must adhere to the ATCA chassis form factor, which is supported by almost all voice and network equipment manufacturers.

One such industry-standard stack is provided by Continuous Computing, a global provider of integrated services that helps telecommunications equipment manufacturers deploy next-generation networks. Among the company's key offerings is the Trillium Software suite, a line of more than 60 standards-based telecommunications protocols. A key product among these is the session initiation protocol (SIP), which forms the backbone for Internet-based communications.

SIP is the industry-standard signaling protocol for VoIP, widely used for setting up and tearing down multimedia communications sessions such as voice and video calls over the Internet. It is designed to be independent of the underlying transport layer, whether it is transmission control protocol (TCP), user datagram protocol (UDP), or stream control transmission protocol (SCTP). It is widely used for Web-based applications such as videoconferencing, instant messaging, streaming multimedia, and online games — and the Trillium SIP is the industry leader. Sun's advances in CMT have taken Trillium to new levels of throughput. This paper will examine how this revolutionary advance was brought to bear on the industry.

Chapter 2
Chip Multithreading (CMT) Doubles Throughput

The SIP software offering was originally offered in a single-threaded mode. As processor frequencies became capped due to heat dissipation and resultant inefficiencies, Trillium sought ways to meet its telecommunications customers' needs for vastly greater throughput at little or no extra cost. An innovative path was offered by the thread-level parallelism that is integral to the multicore architecture of Sun's UltraSPARC T2 processor.

In May 2008, with a little code revision, Continuous Computing was able to run the Trillium multicore SIP on a Sun Netra™ T5220 server using an eight-core, multithreaded UltraSPARC T2 processor, and thereby achieve a breakthrough rate of 6,000 calls per second. This was more than twice as many calls as any other processor that they had previously tested on. Most important, the application effectively utilized all 64 available threads to scale performance through higher CPU utilization, rather than by addition of more CPUs. This meant the company needed less than half its previous hardware footprint to serve the same number of customers, thereby reducing infrastructure and power costs while improving compute density.

Chapter 3

Attaining the Breakthrough: Step by Step

Preliminary planning phase

The overall project needed the engagement of two software engineers for a total period of three months. One had experience in application architecture and did the actual code changes, while the other took responsibility for the QA function. They began by using tools and documents recommended by Sun. The results of this general analysis helped them profile their application and determine where they would get the most impact in return for changing aspects of their code. Once they had addressed the high-return changes, they were able to proceed to fine-tuning.

Testing suitability of application conversion to multithreading

The first step was to examine how suitable the application was for conversion to multithreading. Sun's CoolThreads™ Selection Tool (cooltst) was used for this purpose. The cooltst observes a running workload and applies various heuristics to assess whether it is suitable for systems based on UltraSPARC T1 and T2 processors. The tool bases its recommendations on two main criteria. The first is an analysis of the percentage of floating-point instructions. If this percentage is high, the workload is more suitable for an UltraSPARC T2 processor than an UltraSPARC T1 processor. More important, cooltst also evaluates the degree of potential thread-level parallelism, as measured by the spread of CPU consumption among software threads. It also evaluates instruction-level parallelism, as measured in cycles per instruction (CPI).

Trillium's output from running this test was green, indicating the SIP application was a highly parallel workload that could exploit the hardware parallelism of CMT processors to achieve high throughput.

Understanding architectures that support multithreading

The next step was to examine the various strategies outlined in the Sun BluePrints™[1] document for multithreaded applications. The document describes many ways to maximize system throughput, illustrating how the UltraSPARC architecture makes very efficient use of pipelines to overcome the negative effects of memory latency. If a particular thread is stalled waiting for memory, its cycles can be directly and immediately used to process other threads that are ready to run. In addition, if many threads run the same application, they can benefit from well-designed sharing of the Level 2 cache.

1. "Developing and Tuning Applications on UltraSPARC T1 Chip Multithreading Systems," by Denis Sheahan, sun.com/blueprints/1205/819-5144.pdf.

Code profiling

The SIP application was profiled using the standard utilities gprof and OProfile, as well as IBM Rational Quantify. The results provided a snapshot of where the program spent the majority of its runtime, indicating the code sections that were prime candidates for faster execution. In this case, two portions of code lent themselves to multithreading — one piece was related to the transport layer, the other to processing. The transport layer was stateless, and therefore easier to convert. In networking applications, it is generally simpler to write multithreaded data plane code, since it is usually related to store-and-forwarding, without decisions having to be made based on data content. The remaining bottleneck after that was the CPU-intensive SIP processing code. Since this was stateful, it took more time and care to make it multithreaded, since it was necessary to monitor the data states at all times — as is usually the case for control plane code.

Thread distribution

In the original application, a single thread was handled by a single processor. With the multithreaded version, there were now n instances of threads. A decision had to be made as to the distribution of threads between the different cores: Did one continue sequentially as before, in a round-robin fashion, or design some kind of intelligent apportioning based on packet flow? The best answer was to let the application function itself determine the way threads were distributed.

In the case of SIP, the incoming packets represented thousands of independent telephone calls, none of which needed to share data with other calls in the stream. However, the packets related to each particular instance of a call required an awareness of its state — therefore, each connection had to be encapsulated within a thread, and the distribution was determined by packet flow. The first message in the header packet often indicates what kind of data follows, helping lead to a decision as to which available core will be assigned its thread.

A statistical analysis of calls indicated that 90% were of a standard size, especially in cases such as IP forwarding, and they could all be assigned threads in the same fashion. The remaining 10% were nonstandard — having five or 10 extra packets per feed, for example. These were distributed evenly against all threads.

Preventing deadlock

For multithreaded computing, it is critical to eliminate any point of contention for a single thread. The simplest way to prevent locking is for each thread to store its calls and context. In this case, the software designer kept track of the various calls via a hash table or linked list, and an UltraSPARC T2 processor thus had 64 copies of each data structure. In more complex cases, such as UDP servers — in which data is shared — messaging between the 64 copies would ensure that the data they carried were synchronized. If, for example, the server were going down, the messaging would ensure that one copy contained the master data, while all other copies were terminated before failure.

Fine-tuning for efficiency

Multithreading is easiest to do when all threads are uniform, so a further adjustment had to be made, depending on the thread type. In this particular case, the transport threads had different profiles than the processing threads, requiring different treatment. Instead of mapping both types of threads — e.g., two transport and six processing — onto the same core, Trillium found they realized greater efficiencies by mapping all the transport threads to some cores and all the processing threads to other cores. They also found it was better not to fully load cores designated for interrupt processing, as they ultimately supported fewer threads — three or four, instead of eight.

Application-level scalability

Applications with less shared data are more scalable because it is less necessary to lock while memory is accessed. SIP systems deal with independent phone connections that almost never need to share their content, so this proved to scale almost linearly.

At the application level, one should always strive to increase the instruction per cycle (IPC) count to one or more, ensuring the pipeline is used efficiently and not stalling while waiting for memory access. If cache utilization is not at least 70–80%, this indicates there is too much thrashing — excessive movement of information between system and virtual memory — and that data structures should be redesigned to minimize cache misses. To prevent data structures from having access to and possibly underutilizing two or three caches, the processor enables specific data structures to be locked to designated caches. In the specific case of SIP, the connection control logs were cache aligned.

System-level scalability

Operating systems (OSs) enabling atomic operations afford the ability to minimize locking. The Solaris™ Operating System supports such instructions, such as load and store, without interruptions. Using these instructions enabled programmers to take maximal advantage of the UltraSPARC processor design.

The OS itself can sometimes dominate cache usage. It is important to separate the waste from unnecessary housekeeping system calls and to eliminate them by use of stripped-down versions of the OS. An example of this is the Sun Netra™ Data Plane Software Suite, which is bundled with a lightweight runtime environment to achieve line-rate packet processing speeds.

At the system level, some architectures do not offer a large enough Layer 2 cache for efficient execution. It is possible to use special system calls to change page sizes, from the standard 4 Kb to 4 Mb, for example, so that all the required data fits into a single page without swapping. This greatly reduces virtual memory management overhead.

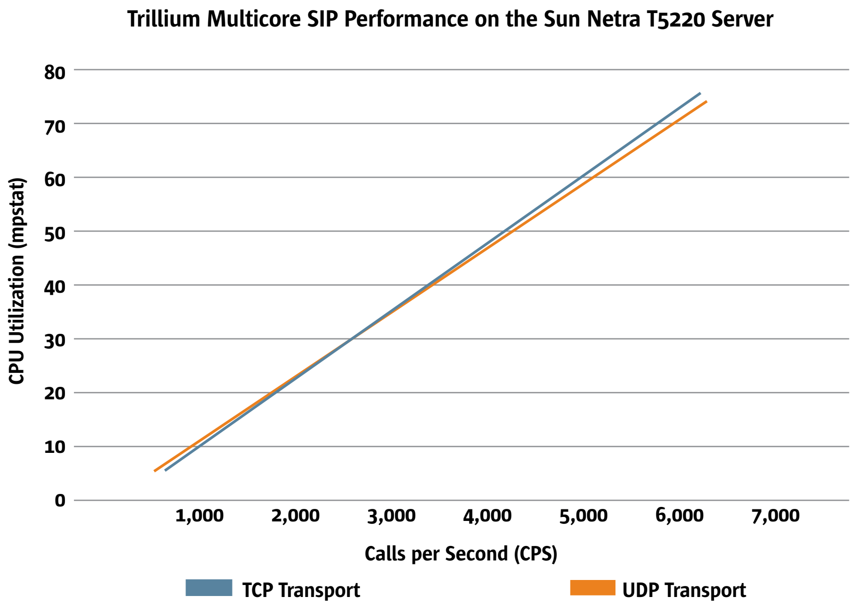
Scheduling

By using the mpstat and corestat commands, Trillium's programmers were able to analyze and decide the best way to schedule threads using Solaris OS system calls. Performance gains were also obtainable with efficient interthread communication. At each step, the programmers' goal was to make threading configurable, so that not much effort was needed to scale from 64- to 128-way multicore processors in the future. Telecommunications applications usually call for such customization. Enterprise and Web applications, on the other hand, are far more uniform and can make use of the standard optimization flags that are part of the compiler.

Results

With a Sun Netra T5220 server powered by the eight-core, 64-thread UltraSPARC T2 processor, Trillium multicore SIP was able to effectively use all available threads and achieve a record 6,000 calls per second — twice as many as any previous processor tested.

Calls per second	Messages per second	TCP processor utilization	UDP processor utilization
1,000	12,000	11	12
2,000	24,000	23	24
3,000	36,000	35	36
4,000	48,000	49	48
5,000	60,000	63	62
6,000	72,000	78	76



Trillium multicore SIP performance on the Sun Netra T5220 server scales linearly with load.

Chapter 4
Conclusion

The SIP application scales linearly with increased load when taking advantage of the UltraSPARC T2 processor's CMT features. This enables it to provide predictable performance under load, while significantly reducing costs for revenue-generating SIP-based services, ensuring that operators can meet their service-level agreement (SLA) guarantees to their end users. The UltraSPARC T2 processor provides the basis for an ideal solution for telecommunications vendors to start small and scale fast while maximizing efficiency, performance, and customer satisfaction.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 1-650-960-1300 or 1-800-555-9SUN (9786) Web sun.com