PETER BAER GALVIN

Pete's all things Sun: T servers—why, and why not

Peter Baer Galvin is the chief technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is co-author of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://www.galvin.info and twitters as "PeterGalvin."

pbg@cptech.com

SUN USES THREE CLASSES OF CPUS AS the basis for its products: SPARC VI and VII, SPARC T1 and T2, and x86. Choosing the best CPU, in the best system, to solve a problem becomes more challenging the more choices there are. Frequently, I'll be asked to recommend a best-fit solution. Sometimes I'll need to debug the performance of a system to determine where its bottlenecks are and if it is the best fit for the workload.

At the lower end, Sun has x86 (Intel and AMD) CPUs. Those are solid, fast, general-purpose, cost-effective CPUs that are used in systems with one to eight sockets. Those CPUs fit into the X-server line. At the other end of the spectrum are the SPARC VII CPUs, co-produced with Fujitsu. These are also solid, fast, and general-purpose. They are more expensive than the x86 CPUs, but in exchange for that cost they scale to very large systems, from one to 64 sockets. The SPARC VII CPUs have a maximum of four cores, and each core can run two threads concurrently. Effectively, these systems give you eight "hot" threads per core. These CPUs fit into the M-servers.

That leaves the third CPU line, code-named "Niagara." These CPUs are more special-purpose. There have been two major generations, the "T1" CPU and the "T2" (and T2 Plus) CPUs. For simplicity I'll refer to all of the servers that run Niagara CPUs as "T" servers. This family includes the T1000, T2000, T5120, T5140, T5220, T5240, and T5440 servers, available in the Sun Blade 6000 blade chassis as the T6300, T6320, and T6340. These systems are cost-effective, especially considering the number of "hot" threads they provide, and in some ways they are general-purpose, but in other ways they are not. That issue is the genesis for this column.

This column comes not to praise nor to bury the T servers. Rather, the goal is to correct misperceptions and help ensure that the systems are purchased for the right reasons, and not for disappointment-causing wrong reasons. Many times over the past few years, I've helped customers (and even non-customers) drill into the performance of their T servers. Typically, the problem is summarized as "we expected the server to deliver X performance, but we're seeing Y." The problem is that X is greater than Y, and not vice versa. Certainly that complaint has been heard many times over the history of computing, but the T servers are particu-

larly difficult to foretell performance on, so even savvy system administrators are surprised when the real results vary from the expected theoretical ones.

The cause of many of these under-performance problems is the theory-to-reality gap. In theory these systems should have excellent performance for a variety of tasks. Instead they end up having breakthrough price/performance for some workloads, but disappointing performance on others. Let's explore how these CPUs work as a prelude to sorting through what they are good at, where they are lacking, how to determine beforehand how a T server may behave, and how to determine under load how well the T server is performing.

Theory

The T servers have one to four sockets. Each socket holds a CPU with up to eight cores. The CPUs currently range up to 1.4GHz in clock rate. Each core can have eight "hot" threads, that is, eight threads can be making progress on the CPU without the system performing a context switch. However, there are not eight computation engines per core. Rather, each of the eight threads is round-robin scheduled on the core. For details of the architecture of the Niagara CPUs, take a look at the Sun Niagara page [1]. An architecture diagram of a single socket of Niagara II CPU is shown here for easy reference.

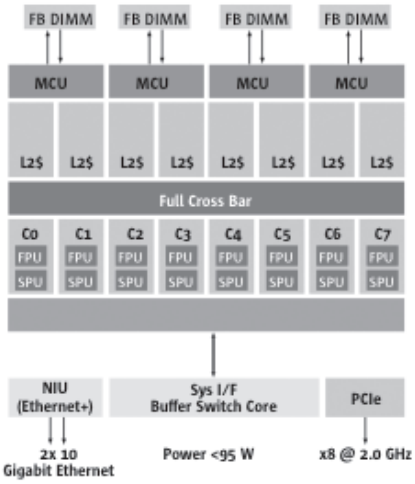

FIGURE 1: SUN NIAGARA II CPU ARCHITECTURE

These T system CPUs are more than just integer units, adding to the expectations of stellar functionality. Each chip also includes eight cryptographic accelerators and eight floating-point units, and in some configurations the systems also have dual 10Gb Ethernet ports. Finally, Logical Domains, or LDOMs, are an included virtualization technology that allows, at the maximum, a virtual machine per thread. The T systems have won many benchmarking records, including world record single-socket SPEC integer and floating-point benchmarks.

Reality

In many instances, T servers are deployed in environments where they are doomed to have poor performance. For sites that understand the architecture of the T server and want to attempt to determine ahead of time whether a given workload will perform well there, the cooltst tool [2] is the first step.

This tool runs on x86 or SPARC hardware, on Solaris or Linux operating systems. It gathers more data if run as user "root" but can be used by non-root users. Obviously, for best results it should be run on the target system while under a usual or high load. It runs for five minutes by default and gathers data about floating-point operations (important for T1-CPU-based systems) and multi-threading. It then outputs a summary of the analysis it performs, including a basic "green, yellow, red" rating of the workload. Unfortunately, this simple rating system is, well, too simple. A "green" rating will not ensure that the same applications, running under the same workload, will run well on a T server.

There are specific cases, which turn out to be quite common, in which a multi-threaded workload will run slower than expected on the T servers. Let's have a look at each of these problem scenarios.

First, the cores used in the Niagara CPUs are rather basic. They do not have the advanced features of CPUs with fewer cores, such as multi-stage pipelining and branch prediction. These advanced features help those CPUs accomplish more in a given clock cycle. Conversely, the lack of those advanced features decreases the amount of work done by a CPU. Before the Niagara CPUs, we were used to a SPARC CPU being similar to other SPARC CPUs. That is no longer the case. The Niagara CPUs do less work per clock cycle than other SPARC CPUs. Clock rate is no longer a good indicator of how fast a CPU is or how much it can perform compared to other CPUs. By combining data from a variety of sources, I've determined that the Niagara CPUs perform a task at about 70% of their core clock rate, on average. That is, a given thread running on a 1.2GHz Niagara core will finish in about the same time as it would have on a single SPARC core (e.g., an UltraSPARC III) running at 800MHz. The percentage difference varies depending on workload, so, as always, a real, well-run benchmark based on your actual workload is the best way to determine performance.

Second, overestimating how multi-threaded a workload is can be painful. If the workload isn't highly multi-threaded, then a chosen system can end up with a lot of unused CPU resources. The highest-end T server has four sockets of Niagara T2 Plus CPUs, each with 64 hot threads. Thus, the system can reasonably run 256 concurrent threads. Of course, the load would be less "reasonable" if each thread was performing high I/O—256 threads performing high I/O would overtax many networks or SANs. Most developers and system administrators consider dozens of threads to be highly threaded, not hundreds. Cooltst helps some in determining the number of active threads, but some other system tools can be more useful. On Solaris, observe the "r" column generated by `vmstat 10 10`. The resulting number represents threads that were in the run queue, on average, per second for ten seconds. As with all the `*stat` commands, the first line of output is the average since system boot and is usually ignored. Note that the run queue contains all threads that are ready to run but are not yet running. So to determine the number of active threads, add the number of CPUs to the number in the first column. The result is a good indication of how many threads were active on the system during that time. Perhaps an easier way to determine the long-term number of active threads is to look at the output of `uptime`. The load averages are the one-, three-, and five-minute average number of active threads. If these numbers are low—say less than 20—then this workload is not a good candidate for a T server.

You can also use prstat(1) to determine if your application processes are threaded and how active the threads are. Just running `prstat` with no arguments provides a dynamically updated list of all running processes, with

the process name and number of threads in the process shown in the last column (PROCESS/NLWP). If the NLWP value is larger than 1, the process is threaded. Active threads per process can be determined by selecting the PID of an application process and running `prstat -Lmp PID`. This instructs `prstat` to look only at that process, and display a row of output per thread. If the threads show some non-zero values in the USR or SYS column, the threads are spending some time executing on CPU. If most or all of the threads are showing mostly SLP time, the threads are not that busy. Please be aware that there are many reasons a threaded application may show most threads sleeping, and the pattern of the threads behavior can change dramatically if the platform changes, the environment changes, or, of course, if user behavior changes. These are just high-level guidelines and are not intended to produce hard conclusions about an application's concurrency.

Third, even a highly threaded workload may not run well on a T server. Consider a job in which one thread calls another and waits for it to complete its work before continuing. Even though this is a multi-threaded task, it is essentially "sequentially multi-threaded." The threads depend on each other and cannot independently make progress. Multiply this by dozens of instances and a seemingly highly multi-threaded workload actually uses only a small amount of CPU resources.

Fourth, if response time is an important component of a computing task, the T servers may not be a good fit. If all threads that are responsible for response time are short-lived, then the job will likely run well on a T server. On the other hand, if many tasks are short but there are one or more longer tasks that are important in overall response time of the task, then the job does not fit well. For example, a MySQL database that executes read and write calls from an indexed database will likely perform well, but add a table scan to the mix and the user waiting for that scan to finish will likely be unhappy.

In essence, the T servers are trucks, not cars. They can move a lot of computing from start to finish, but any given compute job does not move quickly. Web servers tend to be a perfect fit on T servers, and the further a job moves from that "many short-running threads" scenario, the less likely it is that the T server will provide satisfactory performance.

Tuning

Some unhappy T-server experiences can be made into happy ones by tuning the system. Sun has created a tool called cooltuner [3], which performs a bunch of tuning steps automatically. Also, using the FX scheduler rather than timesharing (TS) is usually a win, as is creating a dynamic resource pool (DRP) for all applications, leaving the kernel and Solaris daemons in the default pool along with all system interrupts and I/O.

If the application is home-grown, then it might be possible to persuade the developers to increase the parallelism of the application, using more threads to have the task run in a shorter amount of time (on multi-CPU systems). Certainly the future of computing is increased multi-core, driving more use of multi-threading. But developers usually have other priorities than making their system administrators happy.

But there are some workloads that will not run well on T servers, in spite of tuning. If a workload doesn't seem to be performing well, then the corestat tool [4] can help determine if the CPU is the bottleneck or if it is being underutilized. In this example, the CPU is not being taxed:

# corestat
Frequency = 1167 MHz

Core Utilization for Integer pipeline

Core,Int-pipe	%Usr	%Sys	%Usr+Sys
----------	-----	-----	--------
0,0	0.09	0.35	0.44
0,1	0.00	0.01	0.01
1,0	0.13	0.09	0.23
. . .			
14,0	0.00	3.26	3.26
14,1	0.00	0.01	0.01
15,0	0.98	0.01	0.99
15,1	0.00	0.01	0.01
----------	-----	-----	------
Avg	0.21	0.17	0.38

FPU Utilization

Core	%Usr	%Sys	%Usr+Sys
----------	-----	-----	--------
0	0.00	0.00	0.00
1	0.00	0.02	0.02
. . .			
14	0.00	0.00	0.00
15	0.00	0.00	0.00
----------	-----	-----	------
Avg	0.00	0.00	0.00

This underutilization is further shown via mpstat. In the following example, cores 60–63 were busy but the rest were idle.

mpstat 5 5

. . .

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wtidl	
0	0	0	9	6	2	7	0	0	11	0	7	0	0	0 100	
1	0	0	1	3	2	2	0	0	10	0	5	0	0	0 100	
2	0	0	1	3	2	1	0	0	10	0	5	0	0	0 100	
3	0	0	1	3	2	1	0	0	6	0	4	0	0	0 100	
. . .															
60	2	0	31	33	0	78	4	17	5	0	444	50	1	0	49
61	1	0	11	40	0	106	3	19	4	0	292	48	1	0	51
62	1	0	14	28	0	67	5	16	2	0	265	81	1	0	18
63	1	0	14	35	0	107	5	16	8	0	591	46	5	0	50

Conclusions

Why is it worth fighting the battle of determining which workloads are right for T servers? Sun's T servers have many aspects that separate them from Sun's other servers (and the industry's servers as well). They use extremely little power per thread and can run many threads concurrently. If a workload needs a truck to move it from start to finish, then the T server may be the best truck going. Just be sure a truck is what is needed before deploying a workload on the T servers.

Random Tidbits

Both Solaris and OpenSolaris have received major updates over the past couple of months. OpenSolaris has impressive new features such as built-in clustering and network virtualization. Both are well worth checking out at www.sun.com.

The CTI Strategy blog (to which I contribute) now has two important FAQs available. One is about the Sun Storage 7000, and the other is about Solaris System Analysis. Both are found at ctistrategy.com.

REFERENCES

[1] http://www.sun.com/processors/UltraSPARC-T2/.

[2] http://cooltools.sunsource.net/cooltst/index.html.

[3] http://cooltools.sunsource.net/cooltuner/.

[4] http://cooltools.sunsource.net/corestat/index.html.