Tuesday, July 8. 2008

About some rumours surrounding the UltraSPARC T1/T2

Every once in a while a blogger, sales rep of a competitor or a misinformed journalist write "The UltraSPARC T1/T2 cores are just UltraSPARC II cores running at 300 MHz". Well, this is a rumour consisting out of two misunderstood points of the architecture. Well informed people already know the stuff in this article but this time your are not the target group of readers. But i think it´s time to write a little bit about all this rumours as i write or tell this answer again and again and again.
Okay ... the UltraSPARC T1/T2 cores are just UltraSPARC II cores ...": This part of the rumour is as old as the UltraSPARC T1 series. But it´s really simple to see, that it isn´t true. Okay, a SPARC CPU is a SPARC CPU is a SPARC CPU ... you will find similar functional units on all SPARC CPUs.

Well, to be honest ... there are some features of the UltraSPARC II that aren´t available in in the UltraSPARC T1 processor: For example, the UltraSPARC II an up are superscalar designs, where as UltraSPARC T1/T2 are scalar designs. The UltraSPARC T1 has short 6 stage pipeline, the UltraSPARC T2 is a little bit longer with 8 stages , UltraSPARC II has 9 stages and the UltraSPARC III has a 14 stages deep pipline. The T1 is a single-issue CPU (it issues one command at a time at pipeline), where as the II/III are multiple-issues CPUs. The cores are pretty much different. So wherever you read this "The N1 core is just an UltraSPARC 2 core", this is simply incorrect.

Okay, you might ask, why didn´t we simply used the UltraSPARC III cores and glued them together: UltraSPARC T1 and II/III/IV/IV+ were designed with different mindsets leading to different designs. The core in the UltraSPARC T1 CPU was developed from the ground up to reach two targets: To deliver a SPARCv9 compatible core in the least possible amount of space with the least possible power consumption (okay, in a given timeframe and budget). Processor design isn´t a single-direction road ... it´s more like a tradeoff game.

The first installment of the concept drove this concept even that far, that they outsourced the FPU commands to a single FPU. FPUs aren´t simple, they take a lot of space and they use a lot of power. We want a fscking fast database and web server. We don´t need fscking FPU commands. Tradeoff game starts. FPU ... there is the door ... please close it from the other side. We´ve learned that some application used FPU where nobody really expected it and now N2 has eight of them.

You want to reach a certain target, look at your given budgets (transistor budget, die size budget, time budget, people budget, money budget) and search for solution fitting in your requirements without loosing to much at another front. IBM wanted the 4,7 GHz CPU and sacrified the Out-Of-Order-Execution on that way, Intel wanted the GHz crown as well and designed the ultra-long pipeline Pentium (anybody remembering Prescott?). VIA wanted a lowest-power x86 CPU and sacrified performance for a low power consumption. Sun wanted a many core CPU and sacrified the core complexity in the first steps. You can´t have it all, escpecially with all this walls around us. The thermal wall, the GHz wall, the budget walls, the structure size wall.

BTW: At the the end the laws of nature will stop us all and i think we will see a development like with the race to reach the temperature of zero Kelvin. Every step to half the temperature needs the same energy (i hope the recollection of my school physics didn´t left me there).Every step to get nearer to the walls imposed by the laws of nature for chip manufacturing will take the same amount of money. I think it´s a save bet, that AMD or Intel won´t build a electron accelerator in the size of the Large Hadron Collider just do produce a particle stream for the Higgs-Boson lithography to build Core10Quad. I think it´s more probable that we will see a Core4Hekaton based on a stacked structure of multiple dies but with a structure size in the range of reasonable economic investments.

We getting to the point where nobody can afford to build a fab for smaller structure sizes only for one or two generations of processors. The timeline of Intel procs is really interesting at this part. Each process technology was in use for two generations. 90nm, 65nm and 45nm (at least when you look at the official roadmap). This is a ruinous game at the long run. And this is the reason why the entire industry put it´s research dollars, euros or yen into manycores.

I strongly believe that all proc vendors will opt for manycore designs. The question is only how to handle the legacy single-thread code. You may end up with a few (4 or 8) heavyweight high-speed cores like Intel or AMD, 16 heavyweight high speed cores with scout cores like Suns Rock. So everybody has the same problem. I´m aware of the developments at Sun to solve this problem and they look really promising, i´m aware of other initatives at other vendors to solve the same issue for their respective technlogies.

Okay, i´ve lost the topic ... back to the deliberate tradeoffs:
For example: The UltraSPARC T1 has a 6 stage pipline. For modern procs this is really small pipeline. But it was a deliberate design decision. We could easily use a longer pipline to reach higher clocking, but here we get to the trade of game again: Yo may reach higher clock frequencies, but the longer the pipline is, the higher the penality for a thread switch gets. You have to empty the pipeline and reload the pipeline and very step in the pipeline takes a cycle.

Let´s assume that you have a 15 stage pipeline. Let´s assume that the current thread stalls in stage 10 as it waits for data in the memory. You can react on this problem in two ways: Wait for the thread to continue or switch to another thread waiting for execution. When you switch to another thread all commands in your pipeline are not longer valid. You have to unload it and start work through the the pipeline again. The penalty for the thread switch is 9 cycles, as you need nine cycles to bring the first command of the new thread to the pipeline stage before the stage, that had stalled before. Okay, and here begins the gamble: Do you expect that the thread will unstall within 9 cycles or do you think it will take longer. 9 cycles are in the pot. Hmmm, hard decision.

Now let´s calculate this with a 6 stage pipeline. Let´s assume, the thread stalls in stage 3. Your switch penalty is just 2 cycles. It´s a safe bet, that you won´t get your data within 2 cycles thus.

But there is a further difference between normal procs and the Niagara. It has an own register set for every thread. This has important reason two. When you want to switch from one thread to another thread, you have to save the register sets, load a new one and when you switch back you have reload the old one. This isn´t efficient. Such a task can take easily several hundred clock cycles.

Given the fact that Niagara has four indendent register windows to switch between threads at no costs, you can use a another trick to fill that bubble in. Try to issue command from every core in a round robin fashion. Even when the pipeline stage stalls, the next command in the thread is perfectly excutable, as it´s from another nonstalled thread. As long the thread is stalled no new commands will be issued to the decoder stage. This is meant by thread switching without any costs.

This design choice have an different, but really interesting implication as well. Let´s assume we have a four core four proc x86 system and a 64 thread Niagara II system. Let´s further assume you run an application with 64 threads. This leads to an interesting effect: Context switches are incredible expensive operations. A context switch takes place when you want to run a different process on your CPU. You remember the the cycle of storing thread A registers, loading thread B register sets, storing thread B registers, loading A register set. When you have 64 processes but just 16 cores and thus 16 register sets, you have to go through the cycle four times to execute all processes. When you have 64 processes and 64 threads and thus 64 register sets ... how many context switches do you have to do? Correct ... none. There are several benchmarks that suggests that a UltraSPARC T2 will keep the same performance over a larger thread count, whereas other architecture may start faster, but will break-in after the thread count is larger than the core count.

And this led to other design decision: Do you really need Out-of-order execution logic (complex, thus power hungry), when you can simply switch to another thread in case of a thread stall? The same for branch predictors. Is it nescessary to have big caches, when you augument the thread switching with four memory controllers and a fast crossbar? You can´t use your x86 knowledge and use it for SPARC. You can´t use your SPARC knowledge for x86?

And there we get to the second part of the rumour. It´s "... running at 300 MHz": I know several presentations of our beloved competitors stating that each hardware thread has only 300 MHz worth of computing power. I knew that this bullet would be fired by our competitors at the moment i saw the first presentation about this chip. They used the following logic: The proc (one version of it) runs at 1.2 GHz A core runs 4 threads in parallel. 1200 MHz divided by four is 300 MHz. I assume that is the source of the UltraSPARC II rumour. The USII was clocked in that range. Well ... this logic is correct ... to a very limited point. When all cores work at full speed and all data fits into the L1 caches this is correct. For example executing the incrementation of a register ad infinitum. The problem: This isn´t a real workload. It isn´t even near of a real work load.

Real workloads are: Loading data from RAM or disk, work with it, save it to RAM or disk. And now a normal CPU does something quite usual for modern CPU ... it waits. Memory is slower than the CPU. Period. And when you clock at 3 Ghz even the first level cache miss and second level cache hit is a high latency event. This is even the case at some extend for clocks at 1 GHz.

As i told you before the Tx series has some logic to skip stalled threads and it issues commands to the decoder round robin. The next cycle is one with effective workand not spend with doing nothing. It´s all about handling the one truth: You have to wait for memory.

So: The calculation clock frequency divided by four is only theoretical because the real world teaches us that cores doing real work will wait most of their time. You can measure almost every application and will find large amounts of cache misses. Did i wrote "You have to wait for memory" before? Well ... nevertheless i write it again: You have to wait for memory most of the time. And it´s an absolute amount of time you have to wait. Given the same kind of memory, it´s not relevant if you core clocks with 1,4 GHz or 3,6 Ghz. The time until the data from reaches you, will stay the same.

You can handle this problem on multiple ways: You can do thread multiplexing like Sun did it. You can but giant caches on your die like Intel or IBM (An Power6 CPU has more cache on die than my first 486 PC had as main memory) You can do out-of-order execution, branch prediction, clever preloading of the caches, scouting. At the end you do all this to reduce the latencies introduced by the speed differences of memory and CPU. And this latencies costs you the most of your performance.

My point is: When you divide the clock frequency of the Niagara by four because of the thread multiplexing, you should reduce the clock frequency of x86 or other high clocked processors by all the ticks wasted to latency. And how do you factor in the cryptographic units of the T1/T2? They work parallel to the cores. The N2 cryptographic units writes back it´s result directly to the cache and don´t use the pipeline ... interesting question and many of the "300 Mhz" people don´t even think about it.

I have to admit: Niagara isn´t a system you can´t use without thinking. At the end: It´s a 1.4 GHz proc with a simplified SPARC core and the biggest advantage of the processor is the achilles heel of the system as well. When you measure single thread performance you certainly measure the wrong thing. When you only load it with a single thread, it´s the wrong processor. When the execution time of a single thread on a system loaded with a single thread is important to you it´s the wrong processor. I have a simple rule, when i support a sales rep: When i don´t know the application, i do not recommend the UltraSPARC Tx series. Period. The problem: The most admins don´t really now their application that good ... but the Coolthread Selection Toolkit solved this problem really nicely. Big kudos to the development team of this tool.

I assume that the people with a negative position to the UltraSPARC T1/T2 chips are exactly the people who tried to use an application on it with the wrong characteristics and were disappointed.

But: When you have several threads in parallel, when it´s important for you that the performance stays at the same level even when 256 threads of Apache do heavylifting data, then UltraSPARC T´s are the correct choice for you. When you want good performance at a high thread count, it´s a good choice. No, it´s the best choice.

You can compare it with a E10k. A UltraSPARC T2 is a E10k on a chip. I wrote in a wiki article some years ago: The T-class systems are huge SMP/near-SMP designs, and they want to be used as such. Don´t let them confuse you by their size. Batoka will give you 256 threads on four rack units. Not long ago you could easily fill a mid sized datacenter with the machines needed for the same amount of cores.

There we get to another problem of modern IT: The availability of notebooks with processors with multitudes of GHz on the developers desktop led to a large heap of bad software. It´s like with the F-4 Phantom. The F-4 phantom was the proof for the concept, that you can fly with a brick, when you put enough thrust at it. And the General Electric J79 for the modern software developer is the Core2Duo at excess of 3 Ghz. The software developer doesn´t get punished for making his/her life easier by using the design pattern of the singleton instead of thinking about proper code scaling on multiple processors. On their notebook with a test load this brick will fly. I spoke to admins full of bitter words about their colleages after they installed an app server per core to get some performance out of their machines. The "good" thing ... all vendors go the way of many cores and the need for good and scaling code isn´t a problem of Sun ... it´s an industry wide problem .. universal punishment for bad code is near. But: In my opinion the problem of bad code will haunt the industry for quite a while. All of us. It will cost all of many headaches and many R&D dollars to find workarounds for this problem.

Okay, the workload for the T class servers are specific ones: High thread count and many small requests but with large datasets, and the T class will run as hell ... This is the reason why we see such good results at customers for OLTP, for webserving, for mail, for enterprise backup any many other loads. This isn´t a niche and it´s the reason why we sell that many of the boxes. Sometimes this poses even a problem for us. Where we selled a big box in the past, we have sell just an T2 system. But better steal your own lunch than giving it undeliberately to somone else ...

I hope i gave you some insight into the differences of USII and UST1/T2 and what led to this design decision. As i´m not a chip guy, i hope i´ve got everything right but please correct me, if i´m wrong. I hope that you got at least an understanding, that the world in chip design isn´t that easy to make simple and thus false comparisions.

Posted by Joerg Moellenkamp in English, Oracle at 21:02