WHITE PAPER

Hyper-Threading and Multiprocessor System Performance

ON SERVER 2003

Should you enable Hyper-Threading?

June 25, 2003

TMurgent Technologies

ABSTRACT[I]

Recent processor developments include concepts such as adding multiple "logical" processors on a single die, often providing seemingly two separate processors, which share some components on the chip. Intel's Pentium Xeon processor family introduced this concept as Hyper-Threading.

Due to issues with operating system support, especially in the area of product licensing, most technologists (including those at Intel) recommended disabling the feature when it first started appearing in systems. As a result, this technology is not well understood.

This paper looks at Hyper-Threading in a multi-processor system, specifically with the Windows Server 2003 Operating System with an eye to making a recommendation to customers running such servers as to whether the feature should be enabled or disabled. Observations made in this paper do not reflect results under the Server 2000 operating systems.

The paper also takes a look at the benefits of using our TMuLimit product with Hyper-Threading enabled or not.

[I] Numerous references to names and trademarks of the following companies appear in this document: Intel, Microsoft Corporation, TMurgent Technologies, Dell, and Citrix Systems.

INTRODUCTION

Moore's Law[1], pontificated many years ago, held that semiconductors would double in density every year. This doubling of capacity has led to the doubling of the performance of processors every year since. Few expected that trend to continue as long as it has. Of course, developments in software to use these faster procesors have continued to keep pace, causing performance to continue to be an issue.

Over the years, advances in processor designs have certainly dramatically increased the processing capabilities of the central processing unit. Most of this increase has been the result of shrinking the size of the designs, allowing clock speeds to increase so that a single instruction occurs faster. The speeds involved are so fast, that the time to execute a single instruction (or more properly, a single micro-instruction) is becoming less of an issue. Chip advances today include expensive strategies to get the instructions to the execution unit for execution. This includes concepts such as multi-layered memory caching and pre-fetching, pipeline re-ordering, and branch prediction.

We do not intend to cover these detail here (Intel makes some excellent references on these topics available on their web site [Ref 1]), however we do want to convey that processor performance today is increasing more about making sure that the execution unit of the CPU is constantly doing something useful. This is what Hyper-Threading is all about.

The CPU today is a complicated beast. Rather than think of it as executing an instruction, it is best to consider a stream of activities that occur to execute instructions for the current software thread. Actions of various portions of that stream will be worked on at any given moment. We can simplify this into three phases as is shown in Figure 1 below.

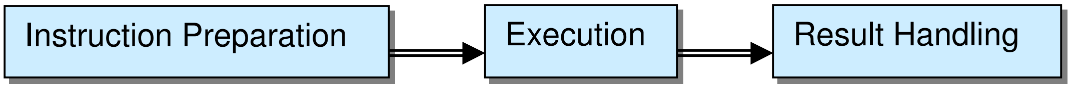

Figure 1 - Phases of instruction execution

In the Instruction Preparation phase, instructions are chosen. If they are not in the local cache, they are obtained. If they refer to data that is not in the cache, that is also taken care of. Instructions may also be chosen outside the natural order of execution, if that makes sense. Complex instructions are broken down into their micro-instruction components.

[1] Gordon Moore, co-founder of Intel, is credited with making this observation in 1965 [Ref 7] while at Fairchild Camera and Instrument Corporation. His estimate that the capacity of semiconductors would double each year (while cost per transistor would half) would last for ten years has been far extended.

Hyper-Threading involves adding the ability to handle a second stream (program thread) in parallel. To contain costs (both power and money), not all components are duplicated. Working from the earlier figure, the current Hyper-Threading solution might look like shown in Figure 2 below:

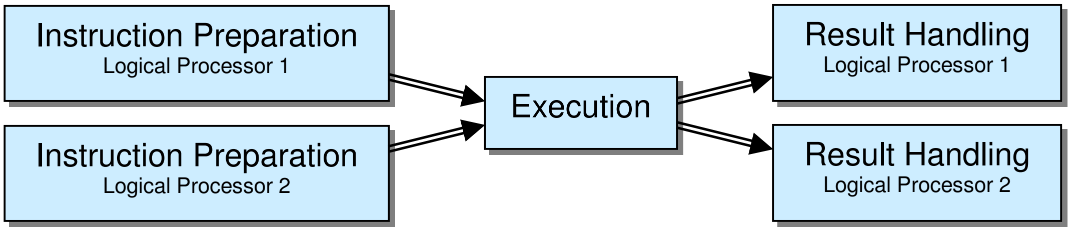

Figure 2 - Phases of execution in Hyper-Threading

A much more technical, and accurate, depiction of Hyper Threading is [Ref 2]. The shared execution stage will alternate between streams, unless one of the streams is not ready (for example due to a L2 cache miss).

Intel is quick to point out the advantages of this strategy. In keeping the most critical phase "fed", the overall performance can be improved. At least with the 2003 OS we tested, the "idle task" that would be running in the second logical processor has a minimal impact.

The flip side of those advantages occurs due to the separation of the OS from the logical/physical processor alignment. This can cause the OS to assign two active threads to logical processors on the same physical processor while other physical processors are dormant. If this second thread were placed on a separate physical processor, the execution of each thread would be improved. While Intel does acknowledge this, it is not reasonable to expect their marketing engine to give it quite as much attention.

These two effects will be demonstrated later in this paper. All the results shown in this paper are taken from tests performed on a server from Dell. The system includes a dual 2.4Ghz Xeon processor with Hyper-Threading. The system BIOS includes the ability to enable or disable the Hyper-Threading, as is shown in figure #. All tests were performed with original release of Windows Server 2003 (Enterprise Edition).

With Hyper-Threading enabled, the system appears to the operating system to be a four-processor system. When disabled, the system looks like a dual-processor system. In our testing, we were unable to detect that the operating system performed any actions that took advantage of knowledge of which logical processors were on the same physical processor. We should point out that this testing was not designed to conclusively prove that this is the case - however we suspect that it is so. We should note that Microsoft, in [Ref 4], hints to possible unspecified performance optimizations in what is now Server 2003, even though we failed to find any optimizations in our testing.

The results in this paper are exclusively related to Windows Server 2003. We are currently running the tests used in the development of this paper under Server 2000. We can verify reports of performance and stability problems with Hyper-Threading on Windows 2000 Server, and at this time recommend customers disable Hyper-Threading under 2000. We are working on a solution that would enable HT under 2000, and expect to publish a White Paper with those results at some future date.

The results in this paper are exclusively related to Windows Server 2003 ... We can verify reports of performance and stability problems with Hyper-Threading on Windows 2000 Server, and at this time recommend customers disable Hyper-Threading under 2000.

LICENSING ISSUES

Because licensing issues can have the biggest impact on choosing whether to enable Hyper-Threading, we should discuss this first.

While software may be licensed for use under an endless variety of ways, here we are concerned with software that is licensed on a "per processor" basis. If you have a single physical processor with two logical processors enabled - does that count as one processor or two? Fortunately Microsoft took an early lead in declaring that the correct answer should be that it counts as one processor.

Unfortunately, the Microsoft Operating Systems prior to Windows XP and Server 2003 counted the logical processors from the Advanced Configuration and Power Interface (APCI) table [Ref 3]. Thus some versions of the OS that should run on a given configuration would fail due to licensing reasons. Microsoft fixed this issue for the OS in Server 2003.

While it is up to each application vendor to define how their products are licensed, we expect **almost** all vendors that license on the bases of number of processors to apply that against the count of physical processors, not logical processors as well.

Typically, when an application licensed on a per-processor basis starts, the application calls an API (Application Programming Interface) to the OS to determine the number of processors. Under older **and** newer Microsoft OS's, the response is the number of logical processors from APCI. To obtain the number of physical processors, applications would need to add code. The new code would either query the chip directly [Ref 5], or (with the release of the Feb 2003 Software Developers Kit from Microsoft), call a new API to determine the number of physical processors [Ref 6].

Unfortunately, applications designed without this knowledge continue to count logical processors against the license count. In some cases, the vendor may have released a patch that may be downloaded, in others, they may grant the user additional licenses until a newer version is available. We recommend to our customers that they contact application vendors to determine their policy. It is unfortunate that there will be cases of vendors requiring the customer to update to the latest shipping version (at cost) for a solution.

Customers considering Hyper-Threaded processors should work with their application vendors to resolve licensing issues early on. Upgrades to mission-critical applications are time-consuming, however, eventually they must be dealt with. Taking the time to correct, and test, newer versions in the test lab before rushing into a major upgrade project continues to be the best policy.

EXAMPLE OF PERFORMANCE INCREASE AND DECREASE IN CPU INTENSIVE THREADS DUE TO HYPER-THREADING ON "LIGHTLY LOADED" SYSTEM

To demonstrate actual results in increased and decreased performance we devised some situations that might occur on a machine from time to time that would exasperate the best and worst of a Hyper-Threaded processor.

In our first example (see Figure 3), we run tests on the upper end of a "lightly loaded" system. To demonstrate this, we use one "CPU intensive" thread per physical processor on our dual physical processor system. The "CPU Intensive" task is a number crunching task that operates in tight loops. Its design results in few cache misses once it starts up. The task measures wall clock time for processing a fixed number of calculations. Measurements from the first and last third of the test are excluded from the results to avoid startup and end condition cache miss issues. Two of these tasks on a dual processor system without Hyper-Threading represents a 100% load.

We further use processor affinity (the ability to assign a process thread to a specific processor) to cause the two tasks to be assigned to either the same physical processor or different ones. We also let the operating system choose processors "randomly".

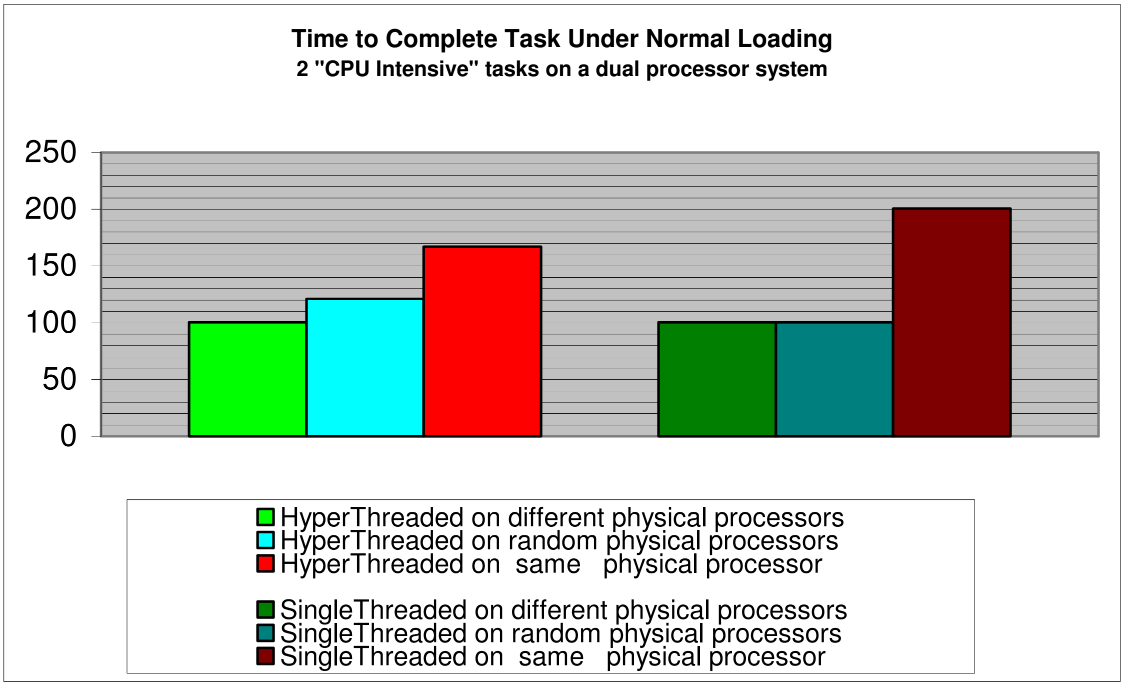

Figure 3 - Hyper-Thread effect on "CPU Intensive" Threads Under Light Load

As can be seen in the graph, when we assign the two threads to different physical processors (green bars), it does not matter much whether Hyper-

Threading is enabled (measurements were well within measurable tolerance). We have scaled the results to indicate 100 shorter of the two values.

Alternatively, the red bars show the results when we force assignment to the same physical processor (In the Hyper-Threading case, this means assigning to different logical processors on the same physical processor). As is expected, without Hyper-Threading (labeled Single-Threaded), assigning to the same physical processor results in slightly more than twice the time (the time above 200% should be attributed to context switching) over the case of Single-Threaded and assigned to different processors. However, comparing with Hyper-Threading enabled, this same situation demonstrated a 17.5% improvement over the Single-Threaded result. This is in line with Intel's claim of 15 to 20% improvement.

With Hyper-Threading enabled, this same situation demonstrated a 17.5% improvement over the Single-Threaded result.

The results when we left it to the operating system select which processor to use demonstrates that flip side of Hyper-Threading. Because the OS acts unaware of the relationship between logical and physical processors, it has a chance of assigning the second thread to the same physical processor. When the second thread is assigned, it has a "one in (number of logical processors minus one)" chance of being assigned to the same physical processor. In the Figure 4 below, we show that when presented with a one task per physical processor, the odds of any thread sharing a physical processor increases towards 50% as the number of processors increase.

Physical Processors	Single	Dual	Quad	8-way	16-way
Odds	0%	33%	43%	47%	48%

Figure 4 - Odds of OS Selecting Same Physical Processor

With Hyper-Threading enabled, this means that 1/3 of the time the second thread was placed in the same physical processor (while the other physical processor had two idle threads), and indeed the results in the blue bar (Figure 3) reflect this. With Hyper-Threading disabled, the OS has no choice but to properly assign the second thread to the second processor. Thus when the OS randomly selects threads to processors this is a case of better performance with Hyper-Threading disabled. (But please remember that this is a closed and contrived test to demonstrate a point, not necessarily a live running system!!!).

Thus when the OS randomly selects threads to processors this is a case of 33-50% better performance with Hyper-Threading disabled.

EXAMPLE OF PERFORMANCE INCREASE AND DECREASE IN CPU INTENSIVE THREADS DUE TO HYPER-THREADING ON "HEAVILY LOADED" SYSTEM

We also demonstrate results on a "Heavily Loaded" System. The graph in Figure 5 represents results of a four "CPU Intensive" tasks on our dual physical processor system.

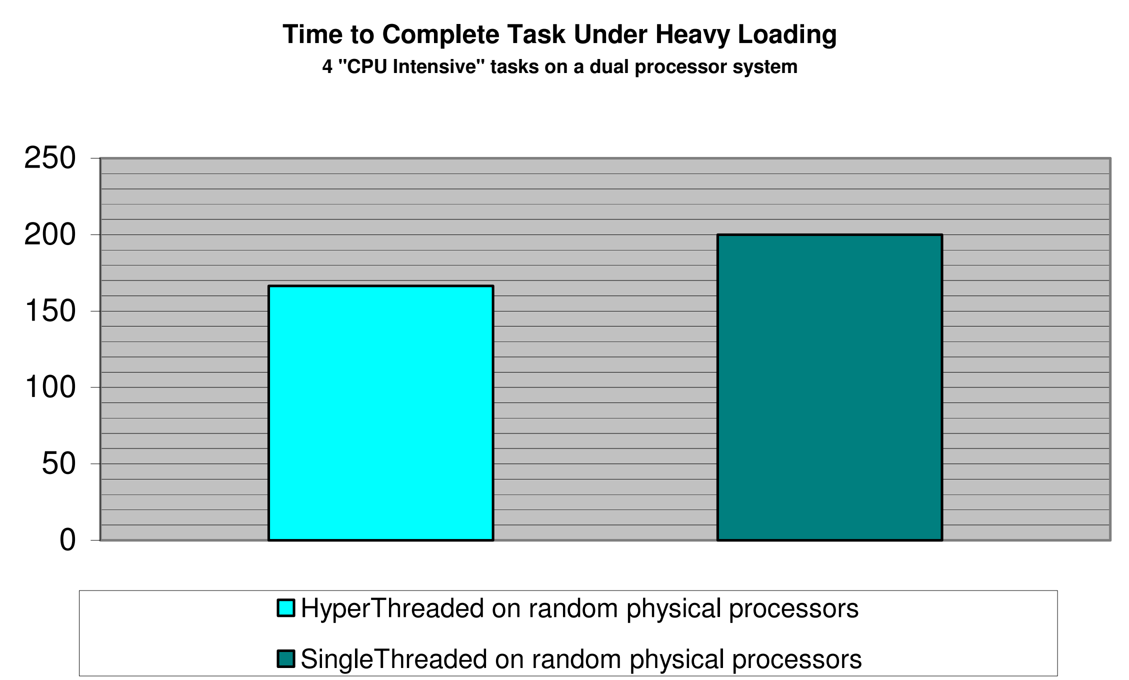

Figure 5 - Hyper-Thread effect on "CPU Intensive" Threads Under Heavy Load

Because there was at least one active task for the OS to assign per processor, it was not necessary to use processor affinity. The time shown is calibrated to the numbers in the previous chart. In the heavily loaded system, we see the same 17.5% performance gain with the use of Hyper-Threading, in spite of the OS. This is in line with the results that Intel advertises for the chip.

In the heavily loaded system, we see the same 17.5% performance gain with the use of Hyper-Threading, in spite of the OS.

EXAMPLE OF PERFORMANCE INCREASE AND DECREASE IN TYPICAL USER REQUEST DUE TO HYPER-THREADING ON VARYING LOADED SYSTEM

As we deal with servers used for multiple simultaneous users, especially systems configured for Terminal Services (and often Citrix Metaframe), we wanted to measure performance more typical of users.

The following test uses a script that repetitively launches a GUI application. The application displays a dialog box, sleeps, and exits. The script measures the amount of time to complete the loop. Again, we use only the middle third of the results. While not representing a heavy use of CPU cycles, this test demonstrates the serialized effect of many small CPU delays among multiple dependent threads each time the thread is in the processor. The script is run under various loads.

In the graph shown in Figure 6, the horizontal access indicates the number of a second type of competing CPU intensive threads (this CPU intensive thread also performs considerable math, however in a less predictable method from a cache miss perspective) also present in the system. The OS was allowed to assign processes as needed, without coercion of affinity.

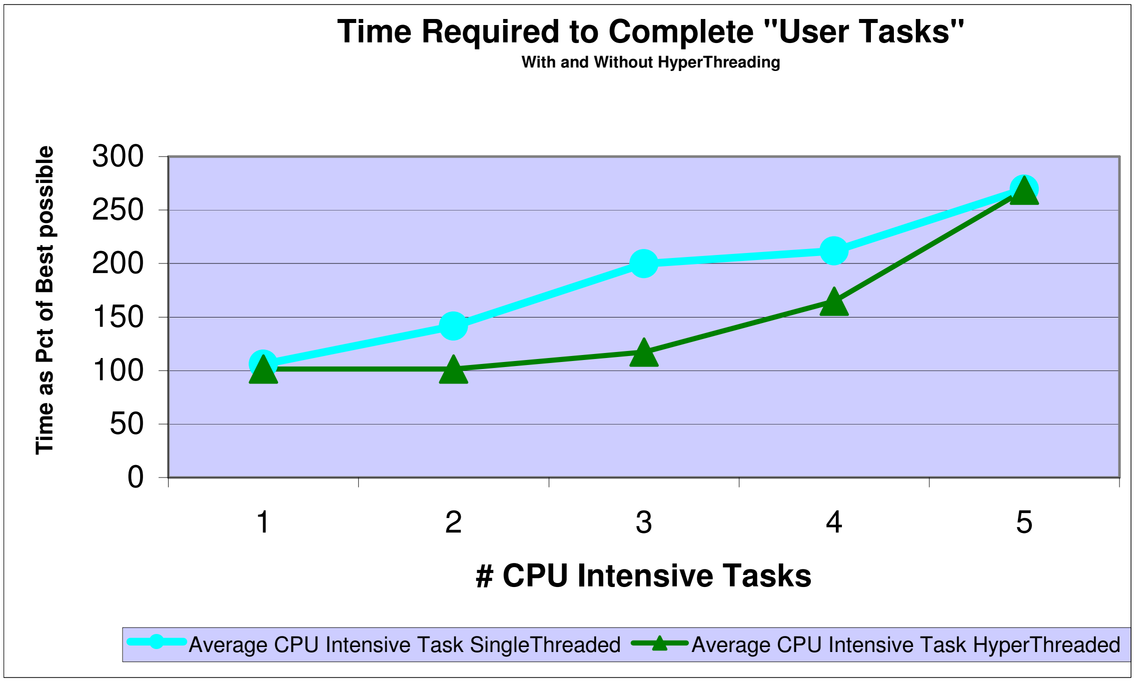

Figure 6 - Hyper-Thread effect on "User" Threads Under Various Loads

The results at the extremes are as one would expect. With only one CPU intensive task present on the dual processor system, in the Single-Threaded case is running at about 50% and Hyper-Threading provides little benefit. With five CPU intensive tasks running, the system is overly saturated and again Hyper-Threading shows only marginally differing results. It is in that in between range of system loading - from one CPU intensive per physical processor to one per logical processor- that turning on Hyper-Threading reduced response time of a "typical user request" by 20 to 50%. The test obviously accentuates the gains to be had by reducing the effect of cache misses.

It is in that in between range of system loading…that turning on Hyper-Threading reduced response time of a "typical user request" by 20 to 50%.

In Figure 7, we present the average calibrated results of the newer CPU intensive tasks performing in the scenario presented above.

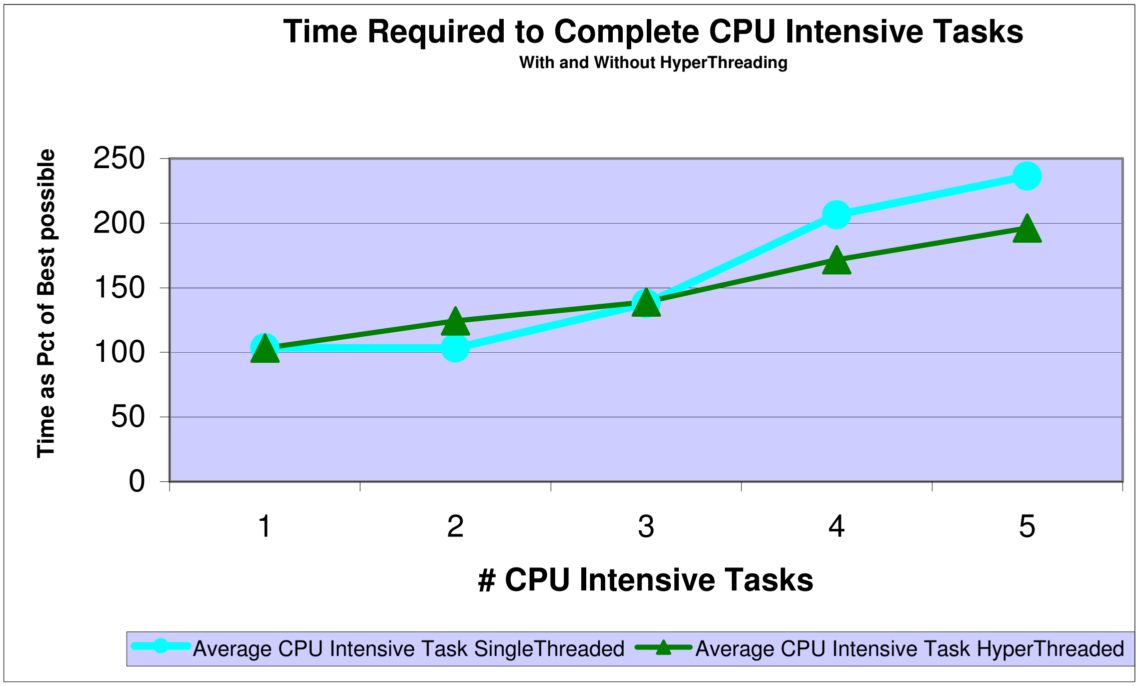

Figure 7 - Hyper-Thread effect on "CPU Intensive" Threads Under Various Loads

Consistent with our earlier results, under a lightly loaded system (one CPU intensive task per physical processor) we can achieve better results with Hyper-Threading turned off. While under increasing loading the Hyper-Threading improves performance of these tasks.

We do not believe that the lightly loaded results above present cause for enterprises to disable Hyper-Threading on their Terminal Servers. Our works on the operating system prioritizations in the past have shown that the real gauge of system performance is how many users can comfortably use the server simultaneously. With that as a benchmark, the scalability of the system is most affected by the serialized delays in performing routine "user tasks", not heavy-duty CPU intensive number crunching activities.

We do not believe that the lightly loaded results above present cause for enterprises to disable Hyper-Threading on their Terminal Servers.

In these two previous graphs, we present a balanced look at the gains and pains of enabling Hyper-Threading in a multi-processor, multi-user server, such as is typical of a Terminal Server. We must however, remember that these cases are derived to illustrate the effect of Hyper-Threading under certain conditions.

In a production server - such as a Terminal Server, there are many more than one "User Task" competing with "CPU Intensive" number crunching tasks. On an 8-way system, there may be thousands of typical "User Tasks" running to only a handful of the "CPU Intensive" applications. It is in this real-world environment that leads us to weigh more heavily the results the results of "User Task" testing than that of the "CPU Intensive" applications. Thus we would recommend to customers that they enable Hyper-Threading on their servers, assuming licensing and upgrade issues do not exist. On average, we expect customers to experience an average of 15% (ranging at times from equal performance to 30%) improved performance in real-world situations when Hyper-Threading is enabled on their processors.

Thus we would recommend to customers that they enable Hyper-Threading on their servers, assuming licensing and upgrade issues do not exist.

While we have not shown testing related to the maximum number of simultaneous users sustainable with and without Hyper-Threading, we would expect an average of about 10% increase when Hyper-Threading is turned on. In a paper done by Citrix Systems, [Ref 8], we can see results in this range for a dual processor system. Although the results in that paper for a quad-processor system in that paper showed a negligible improvement with Hyper-Threading, we feel that this may be due to the limitations of the hardware used. Although we have not verified this, we would expect to see measurable Hyper-Thread benefit in a quad processor system of more recent vintage. Of course, we should mention that application mix in play would affect your results.

EFFECTS OF PRIOTIZATION VERSUS HYPER-THREADING

As mentioned earlier, we do work in the space of improving operating system performance by modifying the multi-tasking selection algorithms of the OS. As the result of this work can be similar - to improve typical user task responsiveness - we wanted to compare results.

We re-ran the prior tests in the presence of TMuLimit, a software service that improves the scalability of Terminal Servers by monitoring and altering task prioritizations. This software reduces the serialized delays that occur to well behaved software applications that need only occasional small CPU slices, by assigning task priorities based upon CPU usage. For these tests we disabled all but the prioritization features of the service.

The chart in Figure 8 shows a combination of the prior results for both "User Task" and "CPU Intensive Task" with Hyper-Threading disabled under varying loads. To this we have added the results of the identical tests when TMuLimit is used to augment the operating system scheduling algorithms.

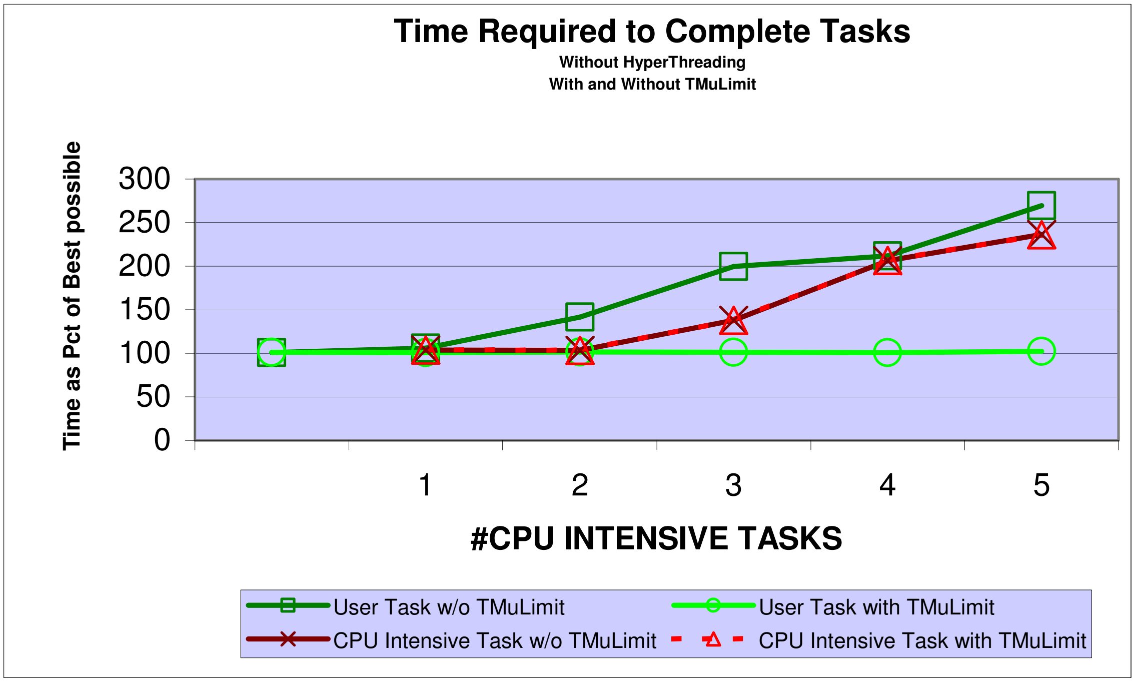

Figure 8 - Effects of Task Prioritization without Hyper-Threading

As can be seen in the chart, responsiveness of typical "User Tasks" are improved. This occurs because the priority given threads that are consuming too much CPU is reduced. In this instance, Terminal Server users would not even notice the presence of the additional system load presented by the "CPU Intensive" tasks. While we know that the time to complete the CPU intensive tasks should increase with TMuLimit when there is heavy load on the system, however the differences in this scenario fall within our measurement thresholds. This provides a dramatic example of the effect of serialized delays has on typical user tasks.

But how would TMuLimit operate under Hyper-Threading? In Figure 9 we present the results the same tests run with the Logical Processors enabled.

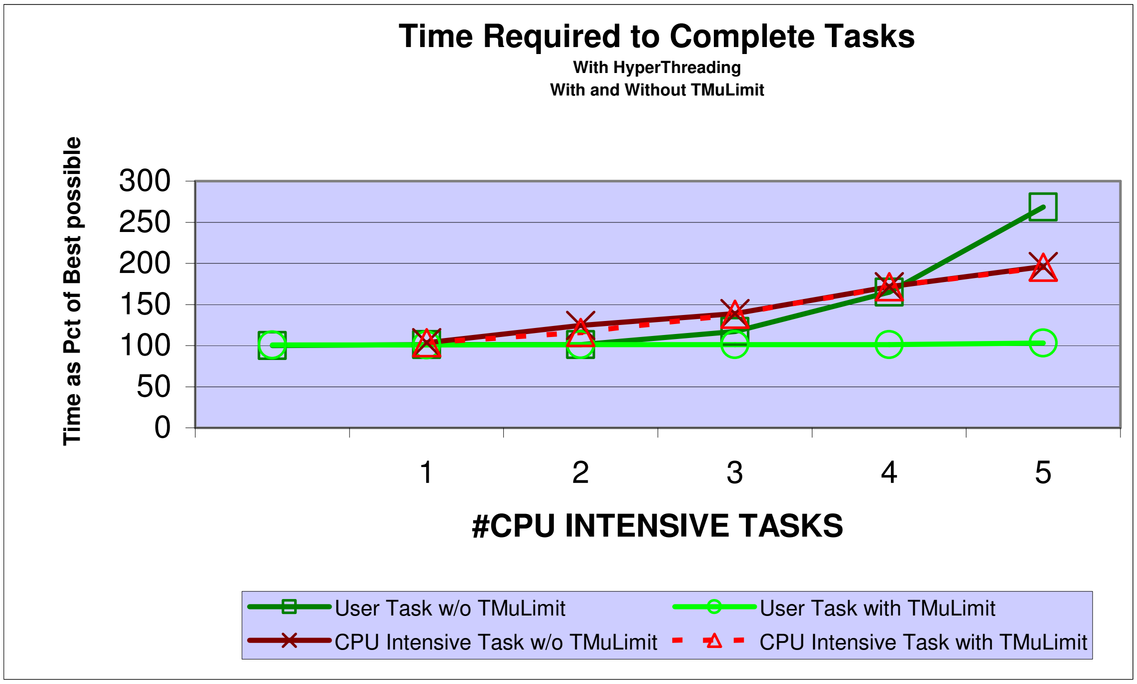

Figure 9 - Effects of Task Prioritization with Hyper-Threading

Again, User Task responsiveness is greatly improved under load with the presence of TMuLimit, with only minor differences in the time to complete the CPU intensive tasks. Together, these two technologies provide the best of both worlds -- improved responsiveness to typical user tasks, plus improved CPU throughput for CPU intensive calculations.

Customers considering system upgrades, or operating system upgrades, to obtain the benefits of Hyper-Threading should consider the benefits of TMuLimit to improve system performance. Using TMuLimit could be a low-risk solution to improve performance for the short run. This could delay costly purchases for a period of time, and provide an opportunity to fully resolve licensing and technical issues

Customers considering system upgrades, or operating system upgrades, to obtain the benefits of Hyper-Threading should consider the benefits of TMuLimit to improve system performance.

associated with a major upgrade prior to deployment. Those ready to deploy Hyper-Threading can still benefit greatly from TMuLimit.

SUMMARY

1) An Intel processor with Hyper-Threading can usually be configured via BIOS to be either enabled or disabled.

2) Licensing continues to be an issue with this technology. The issue is that older software will detect twice the number of physical processors present when Hyper-Threading is enabled. Although, with recent releases by Microsoft, licensing of the operating system is no longer an issue, other applications may be affected. Applications must be tested for license issues, and typically the vendor contacted for a resolution. While we believe most all vendors will be cooperative, in a few cases, it may still be better to disable Hyper-Threading rather than resort to a time and money consuming upgrade process for the application.

3) Under controlled circumstances, Hyper-Threading can reduce performance; under other circumstances it raises performance. We find that in a multi-processor system that is heavily loaded, Hyper-Threading vastly improves the performance. In lightly loaded systems, while the improvement is less, we none-the-less find beneficial increases. In the absence of license issues, we recommend enabling Hyper-Threading in Multiprocessor systems with the 2003 Operating System.

4) The main causes of the performance degradation under Hyper-Threading occurs because the operating system treats all logical processors the same, and the chip treats the priority of the two threads running in the logical processors at the same priority. Here we show how supplementing the operating system multi-tasking selection algorithms can further improve system performance, with or without Hyper-Threading.

5) TMuLimit can be used to delay costly upgrades necessitated by over-worked servers, or to further improve performance on even the newest of servers.

References

[Ref 1] *http://www.intel.com* Intel provides a variety of information on topics that have been helpful in the development of this paper. While some specifics are mentioned in the references that follow, a general reference to the site is in order.

[Ref 2]*Hyper-Threading Technology Architecture and Microarchitecture.* D. Marr et al, Intel Technology Journal, Volume 6, Issue 1, February 14, 2002.

[Ref 3]Hyper-Threading Technology on the Intel Xeon Processor Family for Servers. Intel. 2002. http://www.intel.com/eBusiness/pdf/prod/server/xeon/wp020901.pdf

[Ref 4] *Microsoft Windows-Based Servers and Intel Hyper-Threading Technology*. J. Borozan, Microsoft Corporation. http://www.microsoft.com/windows2000/server/evaluation/performance/reports/hyperthread.asp

[Ref 5] A Solution for Counting Physical and Logical Processors in a 32-bit system. K. Nguyen, Intel. http://cedar.intel.com/cgi-bin/ids.dll/content/content.jsp?cntKey=Generic+Editorial%3a%3axeon_cpu_counter&cntType=IDS_EDITORIAL&catCode=DDZ

[Ref 6] Microsoft Platform Software Development Kit, February 2003. Microsoft Corporation. http://www.microsoft.com/msdownload/platformsdk/sdkupdate/

[Ref 7] *Cramming more components onto integrated circuits*. Gordon E. Moore. Electronics, Volume 38, Number 8, April 19, 1965.

[Ref 8*]* *How Hyper-Threading Affects User Capacity of Metaframe XP Servers.* Citrix Systems, March 2003. http://support.citrix.com/kb/entry!default.jspa?categoryID=118&entryID=2468&