The Myth of Precision Planning:
Understanding Capacity in an Age of Virtual Parallelism

Dr. Tim R. Norton
Inovant, LLC, *A Visa Solutions Company*
CMG 2007 Session 556

Virtualization is the current hot solution for a variety of computing problems. However, the complexities it introduces create additional problems when trying to precisely plan capacity. Virtualization is really the application of techniques to increase parallelism, either actual or perceived, and has been used in many different ways for a very long time. This paper explores many of the techniques used to virtualize and parallelize resources, the impact of those techniques on capacity and the resulting changes in the perception of precision for both planning needs and measuring usage.

1. Introduction

The key to understanding capacity planning has always been two-fold. First is the understanding of the demand for resources. Second is the understanding of the available resources. Ever since the early days of computing there has been a push/pull relationship between demand (workloads consuming resources) and supply (resource capacity). Workloads grow and push the need for more capacity. The complexity resulting from larger workloads increases the desire for planning precision to avoid the expenses of over-provisioning. Capacity becomes less expensive and pulls bigger workloads out of developers. Reduction of cost decreases the need for planning precision because over-provisioning is less costly than the effort required for greater precision. Moore's Law, which states that the number of transistors on a single integrated circuit will double every 18 months, has been the enabler of capacity potential by providing consistently increasing capacity at consistently decreasing cost, but limitations in theoretical design and in practical manufacturing threaten to disrupt the consistency of the cost/value curve. (Tummala 2006) While advances in semiconductor technology have continued to allow manufactures to achieve the Moore's Law transistor density, the design problems created by the extreme complexity of using that many transistors have forced all of the major processor manufacturers to use more parallelism. The Electrical Engineering and Computer Sciences department at the University of California at Berkeley conducted a two year multiple university research project on this topic. The published results, *The Landscape of Parallel Computing Research: A View from Berkeley*, is a 56 page report with 134 references that provides compelling insights into using parallelism and the direction of the latest processor designs. "This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures." (Asanovic, et al. 2006, p. 5) The main focus of the Berkeley research was to understand all aspects of what they see as a change in the industry from uniprocessors to "multicore" processors to "*manycore*" processors, the idea being that it is both more efficient and more effective to use a very large number of simpler processors than try to design and build ever more complex scalar processors. For example, Intel has demonstrated a single chip with 80 processors (Greene 2007) and Cisco is shipping a product with 188 processors (Asanovic, et al. 2006, p. 7).

How does such a shift from scalar uniprocessors to increasingly parallel environments affect the capacity planning process? It increases the demand for resources because of additional work that must be done for synchronization and communication. It decreases the capacity because some resources are not available, either because workloads do not parallelize well or because of actual dynamic capacity changes. This paper introduces the new term "hidden consumers" for the former and the new term "hidden resources" for the latter. However, before looking at these capacity issues a discussion of the techniques to implement the shift is required. Section 2 discusses the Enhancing Techniques that are being used to increase parallelism, from virtualized instructions to virtualized systems. Section 3 discusses Capacity Inhibitors, which are beneficial aspects of the Enhancing Techniques that either negatively impact actual capacity or mask it so that it cannot be measured.

2. Enhancing Techniques

The designers of computing systems and components have developed a number of parallelizing techniques to improve overall system performance and capacity. While manufacturers have unique marketing terms for their implementation of specific functionality these improvements fall into four general categories, introduced here as: Thread-Internal, Core-Internal, Multi-Core And Multi-System. This section explains what these improvements are and Section 3 shows how they complicate capacity analysis.

The Berkeley study has taken an interesting approach by identifying 12 "Conventional Wisdoms" to illustrate how they see the changes to computing. (Asanovic, et al. 2006,

p. 5-6) While this study covers many different aspects of this extremely broad topic, including application design and new benchmark techniques, several of these conventional wisdoms are related directly to understanding the impact of the changes on planning capacity. CW#1 ("Conventional Wisdom" #1) is the "Power Wall" and is the change from thinking of power as free to power being expensive. This is the direct result of the massive increase in the number of transistors in processor chips. The improved manufacturing techniques that have allowed the increase have also reduced the cost of a single transistor to almost nothing. However, each transistor consumes power and a large number of them consume a lot of power. CW#7 is the "Memory Wall" and is the change from thinking of any complex function, like multiply or divide, as slower than simple memory access, like load or store. The speed of modern processors has increased so much relative to the speed of memory, even L1 cache, that the difference in processing cycles to execute these instructions is completely overwhelmed by the number of cycles lost waiting on memory. For example, a typical processor can do a multiply in four cycles and while a load only takes one cycle to execute it can take 200 cycles to access memory. CW#8 is the "ILP Wall" and states that there are diminishing performance returns for the added complexities associated with increased ILP (instruction level parallelism). CW#9 is the "Brick Wall" (the combination of the Power Wall + the Memory Wall + the ILP Wall) and states the while the doubling of uniprocessor performance had taken 18 months, it now takes over five years because of design complexities. CW#11 is the change from thinking of increased clock frequency as the primary method of improving performance to the new view that increased parallelism as the primary method. CW#12 is the change from thinking that less than linear scaling is a failure to the realization that any cost effective implementation that improves application performance is a success.

These Conventional Wisdoms illustrate the growing change toward increased parallelism and the use of virtualization to leverage it. The focus of virtualization today is at the system level where multiple operating system images are run in parallel on a single hardware server. However, virtualization has been implemented at several different levels to achieve increased parallelism and thus improve application or system performance. We can look at these improvements in four broad categories: Thread-Internal, Core-Internal, Multi-Core and Multi-System.

2.1 Thread-Internal

Thread-Internal refers to functionality that increases the number of architectural instructions that can be completed for a programming thread of execution in a given amount of time. Including techniques referred to as ILP (instruction level parallelism), the techniques in this category strive to complete more instructions per clock cycle. When the average number of instructions completed per clock cycle is greater than one, the processor is considered super-scalar. Most modern processors are implemented with micro-instruction designs to increase ILP. Architectural instructions are visible to programmers as opposed to micro-instructions which are used in the actual execution unit logic. David Patterson, one of the co-authors of the Berkeley study, is a co-author of an excellent text, *Computer Architecture, A Quantitative Approach* (Hennessy 2007), that provided most of the in-depth information for the following discussion of ILP techniques. Thread-Internal techniques provide a virtual architecture that hides the underlying transistor logic and the various ILP methodologies. Some of the major Thread-Internal techniques are:

- **Pipelining:** This is the technique where the execution of multiple architectural instructions is overlapped by implementing them in stages. Each stage, such as fetch, decode, execute, etc., uses a different section of the transistor logic in the processor. A stall is when an instruction cannot move to the next stage in the pipeline because it is waiting on something, like a memory access or the results of another instruction. Vendors often measure processor performance in terms of the theoretical instructions per second or by using highly tuned benchmark applications. The effectiveness of pipelining depends heavily on the skill of both the application programmers and the compiler designers. Pipeline stalls mean fewer instructions executed which effectively reduces system capacity.
- **Out-of-Order Execution:** This is the technique where one architectural instruction is executed before one that precedes it in the program but that has stalled. The effectiveness of out of order execution also depends heavily on the skill of both the application programmers and the compiler designers but tends to improve performance by finding useful work when an instruction stalls.
- **Instruction and Data Cache:** This is the technique where much faster (and therefore much more expensive) memory is implemented directly on the same chip as the processor (or closer to it than "main" memory). The combination of faster memory access plus reduced connectivity delays greatly reduces the time to fetch instructions or data from memory. The additional overhead for managing cache and finding instructions or data in the cache memory is usually significantly less than the application performance improvement but poor application or operating system design can overwhelm even multi-level cache designs. Cache generally increases the number of instructions executed in a given time because even the best pipelined and out-of-order execution designs eventually have to get more instructions and data from memory.
- **Branch Prediction:** One of the problems with pipelined and out-of-order execution is the delay caused by a conditional branch in the program

thread of execution. Until the result of the condition is known, the processor does not know which instructions can be selected for execution. Many processor designs use sophisticated techniques to predict which path the branch will take and thus be able to do productive work instead of stalling until the conditional computation is finished. If the prediction was wrong, then that work is thrown away and instructions in the correct path are fetched and executed. The overhead for branch prediction is generally low so it usually improves performance but a programmer or compiler designer that does not understand the prediction assumptions used by the processor can significantly impact performance with poorly designed software.

- **Speculative Execution:** This is the technique where both branch paths are executed until the branch condition is known, at which time the results of instructions from the incorrect path are discarded. Speculative execution adds considerable complexity to the processor design but usually improves performance. Because it is not as sensitive to program design, it usually increases the overall capacity of the processor unless the rate at which the program branches exceeds the number of speculative execution paths in the processor design, at which point the pipeline will stall.
- **Vector Processing:** This is the technique where a single instruction acts on multiple data elements at one time. Vector processing is referred to as SIMD (single instruction, multiple data) using Flynn's Taxonomy for parallel architectures. (Hennessy 2007, p. 197) This can greatly improve performance but it is generally limited to specific workloads, like image processing and graphic display. Performance improvements are often overstated based on specific performance tests so the impact on capacity is very workload dependent.

2.2 Core-Internal

Core-Internal refers to functionality that increases the number of architectural instructions that can be completed by a processor core in a given amount of time. Core-Internal techniques provide a virtual processor architecture that hides underlying implementations used to reduce cost and power usage.

- **Alternate Pipeline:** This is the technique were the execution of multiple architectural instructions is overlapped by implementing additional execution unit logic or entire pipelines for common architectural instructions. For example, if a processor implemented a second ALU (arithmetic unit) then two 'add' instructions could execute at the same time. The Intel Pentium processor implemented a second pipeline, called the V-pipe, which allows two integer instructions in the same execution stream to execute at the same time under the right conditions. (Abrash 2001) This approach reduces the numbers of stalls caused by not having an execution unit available for an instruction but it seldom performs even close to the "two instructions at once" expectations of the marketing department. Because the additional execution resources are specific to selective instructions, the performance improvement is highly dependent on the mix of instructions in the program. Few processors provide measurement information to show how these additional resources are being used. Even when the processor does provide a way to get these measurements, very few of the performance reporting utilities collect them because of the lack of standardization across processors, even from the same vendor.
- **Hyperthreading:** This is also a technique where the execution of multiple architectural instructions is overlapped by implementing additional execution unit logic. The difference from pipelining is that with hyperthreading the instructions can be selected from different threads of execution to avoid stalls from lack of enough of the right type of instructions in a single program or stalls from trying to access common memory locations. The most serious capacity problem with hyperthreading is how the additional resources are made available. The implementation of an alternative pipeline is relatively transparent to programs, including the operating system, because all of the instructions are selected from a single thread of execution. Hyperthreading selects instructions from different threads, which means the operating system must have some way to select which threads the instructions can come from. This is generally done by presenting the additional resources as another processor to the operating system (referred to as two complete "architectural states") that effectively allows switching between two threads of execution without operating system context switch overhead. The problem with this approach is that the very nature of hyperthreading is that there is common transistor logic between these two 'virtual' processors that acts as a bottleneck preventing 100% utilization of both at the same time. Therefore, from a capacity standpoint, a hyperthreaded processor will always appear to have capacity that is not being used. Unfortunately, neither the processors nor the performance utilities provide information as to extent of the bottleneck or when the processor is truly saturated. In addition to being dependent on the mix of instructions within each thread of execution, hyperthreading performance is also dependent on how well the operating system selects threads to interleave. Because the operating

system sees the hyperthreading resources as a second, and equal, processor it seldom, if ever, makes the correct selection.

2.3 Multi-Core

Multi-Core refers to functionality that increases the number of architectural instructions that can be completed by the total number of processors in the system in a given amount of time. Multi-Core techniques provide a virtual system architecture that hides specific characteristics of the underlying processor chip and memory implementations.

- **Dual/Quad Core:** This is the technique where multiple processors are fabricated on a single chip. Multi-core chips virtualize the use of some chip transistor logic to improve parallelism such that the performance increase is greater than the increase in manufacturing costs or power usage. One of the advantages of this technique is that interprocessor communication and shared cache access are much faster because they are direct between the components and avoid use of the slower system bus. How well the operating system places processes that communicate or share memory on processors in the same core can make a significant difference in performance.
- **NUMA:** NUMA (Non-Uniform Memory Access) is the technique where each section of main memory is physically packaged with a group of the processors in the system (usually one to four processors). This causes different memory access times depending on the location referenced. Most NUMA systems use large and complex cache memory with complex cache coherence techniques to reduce the average memory access time but there is still a significant penalty for a cache miss to a memory location in a different processor group. (Hennessy 2007, p. 202-224) NUMA systems virtualize the underlying memory architecture to reduce the communications costs that increase when scaling systems with a large number of processors. Application performance will vary depending on application design (memory sharing and communication between components) and cache management (which mitigates the remote access penalty). Some operating systems have very sophisticated process placement algorithms to detect memory sharing and/or communication between processes and move them to a common processor group. Many new Multi-Core processor designs use NUMA techniques in systems with multiple processor chips.
- **Symmetric Multi-Processor:** SMP is the technique where a system is implemented with multiple equal processors (i.e., interchangeable from the perspective of the programmer) that have equal access to main memory (also called UMA or Uniform Memory Access). SMP systems virtualize the processor/memory environment to reduce application complexity and development costs. Specific application designs to maximize the parallel use of multiple processors require additional support, such as High Performance FORTRAN, to expose the underlying architecture. Interprocessor communication, also known as the MP effect (multiprocessor effect), reduces the incremental capacity when another processor is added to the system (the overall capacity increase to the system is less than the capacity of the uni-processor added). The extent of the MP effect depends on the nature of the interprocessor communication and can be so severe that adding more processors actually reduces overall system capacity. (Gunther 1996) In addition, how the operating system supports the additional processors is critical to overall capacity and performance. For example, the Microsoft Windows 95 operating system was designed for use on single processor systems so any additional processors are simply ignored. Even if both the hardware and the operating system fully support multiple processors, an application not designed to use them will not perform better as processors are added to the system. In fact, because of the MP effect, application performance usually decreases.
- **Asymmetric Multi-Processor:** AMP is the technique where a system is implemented with multiple processors that are not equal (i.e., interchangeable from the perspective of the programmer). The degree of asymmetry can be anywhere from minor functional differences to totally different instruction architectures. Access to main memory can be either uniform or non-uniform, depending on how the different processor types are implemented. Generally one type of process is seen by the operating system and/or application programs as the “primary” architecture and the other types are used to off-load functionality. AMP systems virtualize the specific functions supported by the unique processors. Use of asymmetric or off-load processors significantly complicates operating system and application design but can significantly improve performance. Unfortunately, measurement of the use of these processors is extremely difficult which makes quantifying improvement also extremely difficult.

2.4 Multi-System

Multi-System refers to functionality that increases the total amount of work that can be completed by an application (or set of applications) in a given amount of time. The systems can be symmetric or asymmetric and there is no requirement for homogeneity in the environment. Each system generally implements one or more service functions based on standard protocols. Multi-System techniques pro-

vide a virtual system architecture that hides specific characteristics of the underlying system and communication implementations.

- **Clusters:** This is the technique where multiple independent operating systems jointly provide the supported services through operating system implemented communication and synchronization. Clusters can be limited to load sharing or to failover or they can provide both. Application complexity varies, depending on the requirement for maintaining a global state across all of the application components. Communication between systems increases as the amount of information that must be synchronized increases, which reduces overall capacity and increases application complexity. Even when each component of the application on each independent system is able to function autonomously there is still the problem of distributing work to the systems. There are many techniques to accomplish the distribution of work, including additional load balancer systems in front of the cluster that simply redirect work to application systems based on some predefined criteria. How well they achieve true balance has a profound effect on the overall capacity of the environment. Poor balance means that additional capacity will not be fully utilized so the overall capacity requirement must be increased to compensate for the imbalance.
- **Distributed Applications:** This is the technique where multiple independent systems jointly provide the supported services. This is similar to Clusters above but communication is implemented completely in the application instead of in the operating system. Actual implementations are often a combination of true cluster and distributed application designs. Where clusters are almost always implemented with multiple identical or very similar systems, distributed applications can be implemented with systems using quite different architectures.
- **Distributed Operating Systems:** This is the technique where a single operating system image is deployed across multiple physical systems. Distributed operating systems virtualize the underlying implementation to present a single uniprocessor view to applications, thus masking them from changes in the underlying environment. There are many approaches to distributed operating systems with vastly different designs, each trying to compensate for problems, such as communication between the systems, load balancing, process placement or memory access. Capacity planning for distributed operating systems is extremely difficult and immature. Fortunately, the complexities of implementing distributed operating systems have limited their use to very specialized cases, mostly in academic research, so lack of planning methodology has not been a major issue. However, many distributed operating systems concepts have been incorporated into database systems, clusters, and distributed application designs so the problems related to these complexities are starting to be seen in commercial environments.
- **Grids and Networks-of-Workstations:** This is the technique where multiple workstations are used for large computational problems. A Network-of-Workstations (NOW) temporarily uses systems that are idle during non-prime times. A Grid is usually deployed with dedicated systems. Both provide a virtual supercomputer at a significantly reduced cost either by using existing, but idle, resources (NOW) or by using significantly lower cost hardware (Grid). The most notable use of this approach is the SETI@Home project where individuals install a special screen saver that communicates with the project servers to do computations for the Search for Extraterrestrial Intelligence (SETI, see http://setiathome.berkeley.edu/). There are significant issues when processing capacity of workstations is appropriated, either during off hours or when the workstation is idle. Issues include the way additional work is scheduled, process placement (and if redeployment of an already placed process is allowed and when and how it can be moved), and communication requirements. Capacity planning is significantly more complex because it must include not only the plan for the virtual supercomputer that is composed of a very dynamic group of workstations but also the impact on performance when the owner of the workstation wants to use it. (Menascé 1996)

3. Capacity Inhibitors

Capacity inhibitors are anything that keeps an application from using the full capacity potential of a resource. Inhibitors usually result from solving a significant problem which makes avoiding the inhibitors extremely difficult because the original problem has an even greater impact. The outcome of all of the capacity inhibitors is that planning precision is reduced by the introduction of uncertainty or variability.

3.1 Processor Throttling

To address the "Power Wall" many processors have implemented some form of throttling to reduce power consumption. This can be done either by the operating system when it enters an idle state (when no processes are ready to be dispatched, or run, on a processor) or by the ACPI (Advanced Configuration and Power Interface, see http://www.acpi.info/) support chips, or by both. The processor clock speed can be reduced or the processor can be halted or placed in some form of reduced power state.

For example, the power state of the Intel Duo Core processors can be controlled independently by the operating system or the ACPI mechanisms and placed into a number of different states (halt, stop clock, deep sleep, etc.). (Gochman 2006, p. 93) The critical issue is how the operating system and/or performance measurement software account for processor usage in these situations. If overall processing time is not accumulated for a processor when its clock is stopped then it will appear that the capacity of the system has been reduced by a processor for that measurement interval. However, if time *is* accumulated then the processing time for a process using that processor could be overstated by the amount of time the processor was stopped. If the processor clock speed is reduced at lower system utilizations then the capacity usage of an application can be overstated and the response time elongated. The exact criteria used to initiate these changes are usually not exposed in performance measurements.

The impact of this inhibitor on capacity planning is that measurements of both system capacity and application usage of that capacity become dependent on system load. As the system gets busier its capacity increases and the application gets work done faster, so using a traditional utilization threshold may trigger a premature capacity increase.

3.2 Accuracy of Measurements

Any capacity analysis relies on measurements of resources, both usage and potential. Virtualization at any level tends to generalize theses measurements because the point of the virtualization is to abstract the underlying resources. Problems arise when the entity collecting the measurements, be it the operating system, an application or a performance measurement utility, doesn't understand that the measurements are of the generalized resource instead of the underlying actual resource. A measurement technique must make assumptions about what is being measured in order to create a practical implementation, but these assumptions can cause significant problems when the resources are virtualized. For example, many operating systems measure the time a process uses the processor by recording the time from the system clock when the process is dispatched and again when the state of the process is saved so another can be dispatched. The difference between the two times is how long the process ran for that dispatch event and the accumulation of those differences over the life of the process is the total time it used the processor. This is a perfectly reasonable approach because the operating system has total control over which processes run on which processors. A process cannot start or stop running without operation system involvement. When the operating system is running as a guest in a virtualized environment then this measurement depends on how the system clocks are virtualized. If the guest operating system uses the actual system clock then any time that a different guest operating system was running will be accounted to whatever process was running (or processes in a multiple processor system). The guest operating system is unaware of the fact that it lost the use of the physical processors for a while and greatly overstates the amount of time some processes used the processor. If the guest operating system uses a virtualized system clock, then how it is virtualized becomes a significant issue. Many operating systems update the system clock using a timer interrupt but virtualization can cause the interrupts to be delayed. When this happens the virtualized system clock can advance in non-uniform increments causing some processes to appear to use more processor time while others appear to use less.

This problem isn't limited to system level virtualization. Measurements of specific sections of the code in a program often assume that the code execution time will be consistent as long as it hasn't been modified. The problem with that assumption is that Thread-Internal and Core-Internal enhancing techniques change the execution time depending on things other than what is being measured. For example, without a through understanding of the underlying architecture a program profiling utility run in a development environment can recommend changes that perform poorly in a production environment.

This accuracy problem applies to the potential, or capacity, of a resource as well as the use of it. The most common assumption is that a resource can be completely used (i.e., 100% utilization) under ideal conditions. However, the nature of the virtualization of the resource can make that not only impossible but also make it impossible to tell what the maximum utilization really is. For example, the Core-Internal enhancing techniques rely on sharing logic inside the processor core but that sharing creates hidden bottlenecks that make complete utilization impossible. For example, hyper-threading presents a second processor to the operating system but relies on a mix of integer and floating-point instructions for parallelizing the use of the arithmetic logic. Anything other than the exact right mix and complete utilization cannot be achieved. In addition, the instruction fetch, decode and commit logic is shared between the two virtualized processors, which also limits the maximum utilization, and again requires the right, but different, mix of instructions for best utilization. Optimizing use of both the arithmetic and the other logic units requires a workload with mutually exclusive characteristics.

Planning the capacity of anything without really understanding what the true capacity is can lead to serious problems. Many large applications with less than optimal instruction mix, such as Microsoft's SQL Server database, recommend disabling hyper-threading because the small gain from the limited increased parallelism isn't worth the confusion caused by significantly overstated potential capacity.

Because of the accuracy of measurements problem the capacity planner has a choice between two unpleasant options: using erroneous measurements or doing without measurements. Neither of these options is particularly useful and it is not readily apparent which one is the better choice. What complicates understanding of this problem is that the magnitude of the inaccuracies varies significantly

across platforms. The IBM mainframe environment is much more mature and has resolved many of these problems but the Windows and Unix environments are much more problematic because of the number of hardware and software vendors involved. The long-term solution lies in operating systems and other measurement utilities using standardized implementations of new processor virtualization-specific features, once the systems designers implement those processors and the buying public becomes willing to pay for them.

The impact of this inhibitor on capacity planning is that application and resource measurements are less reliable so any projections must include compensation for the increased variability, which usually means including additional capacity.

3.3 Lack of Application Parallelism

It is generally accepted that designing a multi-tasking application is significantly more difficult than writing a single program. How well the application design uses available parallelism has a direct bearing on how well it uses resources in a parallel environment. Most mainframe capacity planners can relate a story regarding the upgrading of a system from a uni-processor to an SMP system and seeing minimal, or even negative, improvement in transaction response times. The reason for this is that the common transaction environment was designed as a single-threaded process, which means that it could not use the additional capacity. The negative impact to response times was caused by the increased communication and synchronization delays between the processors even when the application could only use one at a time. How the application uses parallel resources is extremely difficult to determine and requires a deep understanding of the applications involved and some amount of empirical testing. Some general assessment may be possible by measuring the underlying programming model, such as the "dwarfs" discussed in (Asanovic, et al. 2006, p. 7-19) or by use of compiler optimizations. (Asanovic, et al. 2006, p. 34-37) Other programming models at a higher level of abstraction, such as Microsoft .NET Web Services, can also be used. "A programming model must allow the programmer to balance the competing goals of productivity and implementation efficiency. Implementation efficiency is always an important goal when parallelizing an application, as programs with limited performance needs can always be run sequentially." (Asanovic, et al. 2006, p. 31) Therefore, a common solution is to design the application as many smaller components and let the operating system provide the parallelism rather than the application design. Unfortunately, these issues apply to the operating system as well and the degree to which the operating system takes advantage of parallel resources varies between vendors and even versions.

The impact of this inhibitor on capacity planning is that applications may not scale well on newer larger systems because they cannot take advantage of the increased, but more parallel, capacity. Plans will require additional capacity any time there is an increase in system parallelism in case the application cannot use it effectively.

3.4 Benchmark Mismatch

Benchmarks are applications that can be executed under controlled circumstances and are repeatable so that results can be compared. Benchmarks are designed to mimic a particular type of processing but when the application design doesn't match the benchmarks used to define the capacity units of the resources (SpecINT, TPC, MIPS, etc.) then the application cannot ever achieve full capacity use. It is extremely difficult to determine which benchmark an application most closely matches and that analysis should be redone for every release of the application. In addition, multiple applications, or even components of one application, on a single system may have vastly different characteristics so that it is impossible to match to a single benchmark. While this type of analysis can be done for a few important applications or systems, it is not practical for hundreds of systems in a large enterprise and trying to manage such a large environment with different capacity units for each system and/or application would likely be cost and effort prohibitive.

The impact of this inhibitor on capacity planning is that the understanding of how an application will perform on a new system becomes more approximate. If an application (not including the operating system overhead) currently uses 90% of a 150 SpecINT rated system then the replacement system would need to be a 186 SpecINT rated system to get the application utilization down to 70%. However, because of the Benchmark Mismatch inhibitor the application could actually use more or less but without knowing which a planner would need to be conservative by recommending a larger system (how much larger would depend on the risk sensitivity of the given application).

3.5 Hidden Resources

Hidden resources are those resources that can significantly impact the performance of a system or application but are hidden by some form of virtualization so they cannot be directly measured. Compensating for the hidden nature of these resources can be extremely complex and requires a much deeper understanding of how the resources are measured.

The impact of this inhibitor on capacity planning is that critical resources are not visible in performance and usage measurements to explain the variability in application performance.

3.5.1 Instruction Level Parallelism

Instruction Level Parallelism (ILP) is the set of techniques implemented inside the processor chip to increase the number of instructions completed per second. Most processor manufacturers do not disclose many of the details of these techniques let alone expose measurements of them. Many of the Thread-Internal and Core-Internal enhancing techniques are specific to ILP because they overlay the dif-

ference stages of instruction execution. What is not obvious is that while a processor may appear "busy" executing an instruction it may in fact be waiting for some pipeline stall condition to clear. The length of that delay is time that the processor could have been doing productive work if the mix or order of instructions had been different. Operating systems don't capture this time even if the processor should provide some way to access the measurement of it. This condition is exacerbated by techniques like hyperthreading because it relies on sharing some common transistor logic between two "processors" but no measurements are provided as to when the parallel logic units are saturated and the serialization through the common units becomes the limit to instruction execution. Because of the sharing of the common logic units it may not be even theoretically possible to get 100% utilization from both processors and, to make the situation worse, the maximum utilization of each of the virtualized processors in a hyperthreaded chip is dependent not only on the workloads using them but also on the timing of how the instructions get interleaved. A less than optimal instruction mix, which is highly likely, will mean that the maximum utilization will vary from one processor to the next, from one minute to the next.

The "*ILP Wall*" means that newer processor designs will likely incorporate evolutionary, rather than revolutionary, changes but some processors are already on the market use radically different techniques. For example, the early 64-bit Mecerd processor from Intel was an implementation of a VLIW (Very Long Instruction Word) architecture. It was followed by the Intel Itanium that uses a variation of VLIW called EPIC (Explicitly Parallel Instruction Computing). Both VLIW and EPIC rely on the compiler to pack multiple operations into each instruction word, ideally one operation for each execution unit. If a valid instruction isn't available then a NOP (no-operation) instruction is used which causes effects similar to a pipeline stall. While these approaches can significantly increase performance under the right conditions they are very dependent on the skills of the complier designers as well as those of the application developers.

The impact of this inhibitor on capacity planning is that there is no longer a clear upper bound to the capacity of the system. At best the maximum capacity of a processor, measured as completed instructions per second, will vary around some average that will be application workload dependent. At worst the maximum capacity of a processor will vary widely and erratically depending on the timing and mix of several workloads. This means that a low priority process on one of the processors of a hyperthreaded processor pair can significantly impact the performance of a higher priority process on the other processor of the pair. Additional capacity will be required to compensate for the variability of both the measurements of usage and the upper limit of what can be actually be used.

3.5.2 Memory Access

The "Memory Wall" is no longer about the ability to implement very large amounts of memory but about how long it takes to access the installed memory. How many processor cycles per byte (average and peak) as opposed to how many bytes are accessed. Chip manufactures are increasing both processor speeds and the number of transistors on the chip. The amount of on-chip memory will also increase because that is one of the easiest way to use additional transistors and it also keeps access times somewhat in line with the processor speed. If that memory is used as cache then the cache coherency problems are compounded but if it used as main memory then the NUMA latency problems increase. Therefore, by attempting to solve one aspect of the "Memory Wall" problem another aspect is exacerbated. L1/L2 cache-coherency (maintaining synchronization between levels of cache) can significantly reduce the gains from increasing the size of the additional on-chip memory. (Asanovic, et al. 2006, p. 27)

The impact of this inhibitor on capacity planning is that applications may not exhibit a linear relationship to processor increases because memory access constricts the flow of program instructions. A processor capacity increase that appears to be a large enough to meet the future needs of an application may not because memory access delays elongate application response times. Additional processor capacity may be required to compensate for the effects of this inhibitor but because the extent of the delays is not measurable the additional capacity will have to be approximated.

3.5.3 Remote Access

Distributed application and operating system designs can mask the usage of remote resources or services. When access time to remote resources varies over time or because of dynamic changes in the configuration then it may be impossible to understand what the full capacity of the resources truly is.

The impact of this inhibitor on capacity planning is that application performance is not dependent on just the system where it runs but on the performance of other systems in the enterprise. However, the extent of that dependency is not directly measurable. Additional system capacity may not provide the expected application performance improvement because of critical path delays from poorly performing remote systems.

3.5.4 Processor Interconnection

System components (processors, main memory, timers, support chips, I/O controllers, etc.) must be connected for data to flow through a system. These interconnections in general, and specifically between processors, can be a limiting factor as processor speeds increase. In the early days of microprocessors the interconnect latency across a relatively simplistic system bus was more that adequate. However, processor clock rates now exceed three gigahertz and even more sophisticated interconnections (high speed busses, cross-bar switches and advanced interconnections like Hy-

perTransport; http://www.hypertransport.org/) introduce significant latency. As processor designs change from single-core to "many-core" interprocessor communication will become more asymmetric as the relative cost (increased latency and reduced bandwidth) of going off-chip increases. (Asanovic, et al. 2006, p. 27)

The impact of this inhibitor on capacity planning is that, as the numbers of processors increase, the capacity usable by applications may not scale at the same rate as overall system size. Additional capacity may be required to compensate for the increased communication between processors, especially if the applications are designed using parallelism to take advantage of multiple processors. Such increased parallelism will increase communication delays due to memory sharing between application components.

3.5.5 Asymmetric Processors

A system has asymmetric processors when all of the installed processors are not exactly alike and interchangeable. A system is usually defined by the general purpose processors that implement a specific instruction architecture and the special, or asymmetric processors, use a different set of instructions. A few systems allow mixing processors with the same instruction architecture but different features. For example, the HP Tandem Non-Stop supports two different processor clock speeds. When some processors have a totally different instruction architecture they are usually not exposed directly as system processors but are used through operating system functions (such as graphics or I/O processors) or with extended instructions (such as floating-point processors like the Intel 80287 which added specific instructions to the 80286 general purpose processor). A more complex example is the use of a special purpose processor integrated into a network interface card to offload most of the processing for the TCP/IP protocol stack. This network card, called a TOE NIC (TCP Offload Engine Network Interface Card), requires changes to the implemented network software in the operating system or to the application to use the alternate protocol stack interface. Without one of those changes there is no improvement when a TOE NIC is installed but it still costs more to purchase and power.

The impact of this inhibitor on capacity planning is that application performance can vary much more than expected from one system to another because the extent to which the application uses the special purpose processors is not known. Indeed, most systems do not provide any external measurements of how such processors are used, let alone which processes are using them. Most system measurements use some form of averages which will mask differences between visible processors (such as when clock speeds are not the same) or simply ignore the processing done by the special purpose processors, which understate the resources needed by the application. Often system vendors will change the implementation of special processors without providing substantive details. In extreme cases some features may be implemented in software unless the hardware feature is purchased (at additional cost, of course). Because of this uncertainty, performance tests on one system cannot be used to predict performance on a different system and capacity planning must be based on the assumption that the application will not get any benefit from the special processors. This assumption can lead to over provisioning when special processors are available and the application actually does utilize them.

3.5.6 Control Structure Access

Control structures are structured memory areas that are used by an operating system or an application to control behavior. As a greatly oversimplified example, an operating system could use an array to keep track of the running process where each element has a place for the process identifier, the last program counter, register values to save, the priority, the user, etc. Applications also use control structures, especially when they are multi-tasking. How the operating system and application components implement mutual exclusion (shared resource or producer/consumer) when accessing the structures, and the resulting delays caused by waiting for access, has a significant impact on the perceived capacity of a system. The "hidden resource" here is parallelism, which is reduced as more components contend for the control structures and more work is serialized. A single stand-alone component would not be delayed so that is 100% capacity. Mitigations include the traditional techniques to reduce the MP effect plus newer techniques such as transactional memory (Asanovic, et al. 2006, p. 28) and full-empty bits in memory (Asanovic, et al. 2006, p. 28).

The impact of this inhibitor on capacity planning is to change how workload characterization is viewed. The standard assumption is that the application workload should include those processes that are directly involved in application processing so that non-volume related processes are not increased in the planning projections. However, that assumption no longer applies because increased application volume can increase synchronization delays. Because control structure access delays are not easily measured it is almost impossible to determine when they will become the dominate component to response time. Most likely they will not increase in the same proportion as the application growth so additional capacity will be required as systems get larger and more complex to compensate for the increased variability in system overhead.

3.5.7 Portability

Portability can be viewed as a form of virtualization because it masks the actual underlying hardware and software by defining an abstract environment for the application that can be implemented across multiple configurations. Techniques to increase portability tend to generalize the underlying hardware which disallows the use of specific capacity and performance improving features. (Asanovic, et al. 2006, p. 37) Portability techniques rely on the skill of programmers and require both political and technical tradeoffs among development costs, implementation costs and

performance. One technique to mitigate the problems with portability that shows promise is the use of autotuners. Autotuners are benchmark type utilities that run repeated tests while changing configuration parameters to discover the optimal library implementations for each target environment. This allows the application to be moved without being changed but still use specific features on each system. While promising for complex single system hardware, no successful autotuners exist for parallel systems. (Asanovic, et al. 2006, p. 38)

The impact of this inhibitor on capacity planning is that application performance will scale more slowly than system performance because the application does not use the advanced features of a newer system. This relationship is not easily measured so when moving an application to another system additional capacity should be planned to allow for this inhibitor but the end result may be an underutilized system.

3.6 Hidden Consumers

Hidden consumers are components, software or hardware, which use a resource in such a way that is it not easy, or even possible, to measure that usage. Hidden consumers reduce the available capacity of a resource in such a way that the reduction is not captured.

3.6.1 Operating system features

Some new operating systems are trying to address perceived usability issues with background processes (defrag, indexing, virus scanning, etc.) that are done as part of the operating system idle loop. The problem this creates is that they are excluded from the operating system computations of system busy because they viewed as only consuming a resource that would not have been used anyway. Even though use of the processor can be discounted, the use of other resources, such as power, memory, memory access, cache or I/O, is not considered. If the operating system bases its decisions on processor usage then these other resources can be overcommitted because processor usage is understated but the other resources are still being consumed.

File system features (RAID, encryption, compression, etc.) can also have an impact because their use is not controlled by the application. In fact, how they are configured may not be exposed to the application or captured in any measurement data. Sudden changes in capacity or resource usage could be a result of a change to a configurable feature rather than an increase in application volume or installation of additional resources.

The impact of this inhibitor on capacity planning is that detailed configuration changes should be tracked, but seldom are, and correlated with changes in capacity and resource usage. Without understanding how such configuration changes affect capacity and usage, the planner cannot make meaningful predictions. One alternative is to assume measurements reflect enabled features and risk performance failure if that's not true and the features are later enabled. The other alternative is to assume measurements reflect disabled features and risk underutilizing resources if that's not true and the features a later disabled. In either case, the business trade-off between value of using such features and their impact on capacity is complex and often very political.

3.6.2 Virtualization Implementation

The key concept with system level virtualization is that the underlying resources are shared in a way that increased parallelism. In other words, two or more systems appear to be using the same physical hardware at the same time. This is a very complex topic, that cannot be covered adequately here, and this statement by Michael Salsburg, et al. provides some insight into those complexities:

> For example, what is the basic overhead of running a hypervisor on which the OS images dwell? How does this overhead change as a function of the number of virtual machines and physical CPUs? Can we accurately predict the effects of queueing both at the physical and virtual CPU levels? What is the impact of I/O activity? How about the impact of allocating specific quanta of CPU cycles to each machine? How is performance affected by the selection of a specific virtual technology?
> (Salsburg 2006)

The trick to successful virtualization, regardless of the techniques used, is to maximize the use of resources without negatively impacting application performance. Virtualization raises questions about the overhead of the hypervisor (virtualization control software), clock synchronization and granularity, processor dispatch granularity (does the hypervisor dispatch processors individual or does a guest operating system wait until there are as many physical processors available as defined logic processor units for that guest), and the impact of interrupt delays on the guest operating systems. All of these, and other, topics act as hidden consumers of resources because they are usually not seen and measured by the guest operating systems and the hypervisor does not provide detailed enough measurements to understand their impact at the workload and process level.

The impact of this inhibitor on capacity planning is that the environment becomes much more complex and much more dynamic. Planning at the resource level becomes impossible for two reasons. First, the resource usage measurements are imprecise and unreliable at best and may even be incorrect in some cases. Second, what drives resource consumption is no longer just the application business drivers but everything running on all of the other guest operating systems sharing the same physical resources. The lowest priority work in a guest with a large share of the physical resources can easily run before, or even instead of, the highest priority work in another guest.

4. Far Reaching Effects

The accumulation of all of the Capacity Inhibitors has long-term and far reaching effects on capacity planning. Capacity planning in the Age of Virtual Parallelism is like

peeling back an onion. The planner can continually go to yet another level of detail whenever more precision is needed but at some point the cost will outweigh the benefit. Resources, both human and computer, can be expended to gather the required information and measurements for any sufficiently important system or application. The problem comes when trying to apply that same precision across all servers and applications in a large environment. Capacity planning should always be about the business relationship but now, in addition to defining the relationship between the business and the technology, it must also define the relationship between the business and the planning process itself. Planning must change focus from managing resources to meeting application business objectives and it must do so with introspection, knowing that the planning process can easily become a significant cost. Like the old engineering adage, sometimes close is good enough.

4.1 Workload Characterization

Workload characterization used to be a relatively straight forward matter of assigning processes or users or transactions or whatever to workload groups. But now, as more and more resources are shared in ever increasingly complex ways, those assignments are not so simple. Virtualization at so many different levels makes it almost impossible to assign the use of a resource to a single application workload. The standard apportionment techniques (Norton 2004) for approximating how much usage of a resource should be attributed to an application are no longer adequate because they rely of either precise measurements or a consistent ratio of usage over time. Precise measurements are lost for all of the reasons already discussed and the very nature of virtualization is to allocate resources as needed, certainly not in the same ratio from one time interval to the next.

4.2 Building Models

While capacity planning involves much more than building models, the ability to accurately represent a system or application with an abstract model is still a key tool in the planning process. The commercial and Open Source modeling tools available today are all quite capable of modeling complex virtualized environments. The questions are: What is the expected granularity and precision of the results? and What is the required effort to get those results?

4.2.1 Service Time Calculations

Models use abstraction to represent the time something takes at each stage of a process. Each stage is a server or service center and the time is the corresponding service time. Because the service center is an abstraction of a more complex process, the service time is also an abstraction. Different types of models use a variety of techniques to achieve a sufficient level of abstraction to make the model practical to solve and yet have a sufficient level of detail to give the results meaning. As shown in the previous discussion of Enhancing Techniques, there is almost always more complexity at the next level down. While theoretically possible to build a model of an entire environment, from the behavior of the application to the way the network passes data to the operating system services to the management of cache to the pipeline of the microprocessor to the speculative execution of the underlying micro-op instructions, such a model would most likely take forever to build and somewhat longer to run. The success of a model lays in the ability to cost effectively approximate the behavior at each service center while producing results in enough detail to allow for meaningful predictions.

How do the Capacity Inhibitors affect this abstraction of service time? It may be overstated (longer) because of reduced clock speed at lower utilizations (Processor Throttling). It may not be measurable within the needed precision for the desired results (Accuracy of Measurements). It may be understated because the application cannot use all of the components that have been abstracted into the service center (Lack of Application Parallelism). The capacity of the service center may be either overstated or understated because the application is significantly different from what was used to determine the capacity (Benchmark Mismatch). It may be either overstated or understated because the instruction mix in the application doesn't match the processor designer's assumptions (Instruction Level Parallelism). It may be understated because future transaction volumes don't account for the increased latency when the system is expanded and remote, instead of local, resources are used (Memory Access and Remote Access). It may be understated for measurements at lower utilization (Processor Interconnection). It may be understated because it doesn't account for an entire class of resources that could not be available on the target system (Asymmetric Processors). It may be understated because mutual exclusion delays increase non-linearly as the service center utilization increases (Control Structure Access). It may be either overstated or understated because the new system is configured differently (Portability). It may be understated because of interference from other workloads that will not grow at the same rate as the application under investigation (Hidden Consumers).

The truly confounding dilemma is that some conditions cause overstatement of service time to increase as service center utilization increases and others cause it to increase as service center utilization decreases.

4.2.2 Uniformity

Many of the assumptions made when building a model are about how work is distributed to the service centers. For example, the processors in an SMP system (tightly-coupled processors) can be modeled as a single server where the service time is adjusted for the number of processors and the interprocessor communication (Menascé 1994, p. 263-4). Underlying this technique is the assumption the application can actually use all of the processors. Several of the Enhancing Techniques (i.e., hyperthreading) change the validity of this assumption and those changes can vary dynamically as systems load varies. The ability to spread a given workload across all available similar resources may

not always be a good assumption. Loosing the ability to use such simplifying assumptions will increase the time to both build and to run a model.

5. Conclusion

Virtualization ***is*** the current hot solution but along with it comes a host of other problems. Virtualization is not one simple technique but many techniques to increase parallelism, implemented at many different levels. Sometimes these techniques complement each other and sometimes they work against each other. How these interactions impact the planning process is far too complex to predict. This paper explored many of the techniques used to virtualize and parallelize resources and it discussed some of the impacts of each technique on planning. The final conclusion isn't a definite answer about how to mitigate these impacts but rather a heightened awareness of some of the things that can go wrong. The very nature of a discussion about precision raises the issue of the precision of the discussion. Many details were left out or glossed over because of limited space but also because the details aren't the real issue. All of the topics discussed are in very rapidly changing areas of computer science and any specific details would be outdated long before publication. On the other hand, few organizations have only the latest and greatest installed. A mix of old and new technology means that there is no single answer to any of the issues raised here. Hopefully the exploration started here will grow into a discussion that encompasses many different areas and disciplines. The Berkeley study (Asanovic, et al. 2006) is very interesting because the contributors are from so many different areas and different universities.

The final thought is on the future of capacity planning. Given the complexities, uncertainties and variabilities discussed here, what's a planner to do? The evidence is mounting that planning resources will become harder and less precise as more and more of these techniques, and the ones to follow, are implemented. What remains is to go back to the basics of what capacity planning should be about anyway, understanding the needs of the business. Planners must now, even more than ever, think in terms of identifying and reducing bottlenecks that prevent applications from achieving the business objectives. Efficient use of resources isn't the issue because it's almost impossible to understand. Planning should take a holistic view that balances costs, technical, financial and human, against the overall benefits with the understanding that there will always be localized inefficiencies that inhibit achieving global optimization.

6. References

Abrash, Michael. 2001. *Graphics Programming Black Book*. Chapter 21: Unleashing the Pentium's V-pipe. Byte.com: http://www.byte.com/abrash/chapters/

Asanovic, Krste, Ras Bodik, and Bryan Christopher Catanzaro, et al., 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*, Technical Report No. UCB/EECS-2006-183, http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

Greene, Kate, 2007. *The Promise of Personal Supercomputers*, MIT Technology Review, February 23, 2007, http://www.technologyreview.com/printer_friendly_article.aspx?id=18219

Gunther, Neil, 1996. *The MP Effect: Parallel Processing in Pictures*. In Computer Measurement Group, Proceedings: San Diego, California: CMG, Inc. http://www.perfdynamics.com/Papers/njgCMG96.pdf

Gochman, Simcha, Avi Mendelson, Alon Naveh and Efraim Rotem. 2006. *Introduction to Intel Core Duo Processor Architecture*. In Intel Technology Journal, V10 N2:89-97.Intel: http://www.intel.com/technology/itj/2006/volume10issue02

Hennessy, John L., and David A.Patterson, 2007. *Computer Architecture, A Quantitative Approach. San Francisco*, CA: Morgan Kaufmann Publishers.

Menascé, D. and A. Rao, 1996. *Performance Prediction of Parallel Applications on Networks of Workstations*, In Computer Measurement Group Proceedings: San Diego, CA: CMG, Inc http://cs.gmu.edu/~menasce/papers/CMG96.pdf

Menascé, D., V. Almeida, and L. Dowdy. 1994. *Capacity Planning and Performance Modeling: from mainframes to client-server systems*. Englewood Cliffs, New Jersey: Prentice Hall.

Norton, Tim R. 2004. *Workload Information Tutorial: Mapping Businesses and Applications to Servers and Processes*. Half day presentation at the Fifth International Workshop on Software and Performance (WOSP 2005), July 11-14, 2005, Palma de Mallorca, Illes Balears, Spain.

Salsburg, Michael, Peter Karnazes, and Bill Maimone, 2006. *It May be Virtual … but the Overhead is Not*. In Computer Measurement Group Proceedings: Reno, NV: CMG, Inc

Tummala, Rao R., 2006. *Moore's Law Meets Its Match*, IEEE Spectrum Online, June 2006, http://www.spectrum.ieee.org/jun06/3649