F5 White Paper

Clustered Multiprocessing: Changing the Rules of the Performance Game

Because improvements in practical computer power lag far behind these exponential hardware improvements, the key to maintaining processor performance has become multiprocessor or multicore design. This white paper details how the Clustered Multiprocessor (CMP) from F5, in combination with TMOS, provides customers with scalability, extensibility, adaptability, and manageability.

By KJ (Ken) Salchow, Jr.
Manager, Technical Marketing

Contents

Introduction 3

Typical Multiprocessing Implementations 3

Issues with Typical Multiprocessing Appliances 5

The Logical Solution 7

The F5 Way: Clustered Multiprocessing 8

Changing the Rules 11

Introduction

Since Gordon E. Moore's landmark observation in 1965, the entire technology industry has been rooted in the concept that the complexity of integrated circuits doubles every 18 months (originally stated as every two years). However, many people incorrectly interpret "Moore's Law" to mean that the overall productivity of computer-based processes increases at the same exponential rate.

In reality, improvements in practical computer power lag far behind these exponential hardware improvements. While Moore's Law continues to be a fair indicator of the complexity of integrated circuit design, this complexity is not being applied towards single, faster, larger, and more productive computing units with increasingly larger buses. Instead, it's being implemented in processors with multiple computing units in smaller chips. Also, while the complexity and overall computing power of the processor stays true to Moore's Law, the increased complexity of software design due to multicore processors leaves the end result less than ideal.

Manufacturers of dedicated appliance solutions, or software built to run on off-the-shelf hardware, intending to ride the "Intel power curve" to consistently increase the overall performance of their products, quickly hit the performance wall of a single processor (core). Individual processor speed, which had for many years increased dramatically and consistently, began to stagnate. The key to maintaining performance became multiprocessor or multicore design.

Typical Multiprocessing Implementations

On single-processor, multipurpose machines (like home computers), multitasking and multithreading resembles multiprocessing by enabling the single processor to context-switch between processes or between multiple threads within the same process. For instance, a single-processor home computer can seemingly run both a web-browser process and a word-processing process simultaneously. The single processor can only run one process at a time, but with multitasking, it can quickly switch between the two processes to give the appearance of simultaneous execution. Similarly, individual threads within a process can be treated in the same way. On a single-processor system, however, the number of processing cycles is still finite and the processes share that single resource.

Multiprocessing is a computing architecture that allows the simultaneous use of multiple processors (cores) in order to increase the overall performance of the machine. In multiprocessor machines, both the processes and the threads can run simultaneously on different processors instead of just giving the appearance of simultaneous execution. In general, there have been two predominant methods of achieving this goal: Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (ASMP).

SMP is very similar to the multitasking used on single processor systems. The processes themselves are unaware of the existence of multiple processors. The underlying operating system kernel employs a scheduling process to virtualize the processors and decide which process or thread executes on each processor for any given cycle. This, in effect, still uses multitasking process context-switching; each processor is not guaranteed to continually service the same process (or thread). However, this is the easiest way to gain access to all processing cores with minimal impact on the software design—and it is supported, out of the box, by most operating systems. This is generally more applicable to multipurpose computing platforms (PCs, servers, and so on) although many special-purpose appliances still rely on this form of multiprocessing.

ASMP relies less on generic kernel-level virtualization to provide optimal use of multiple processors and puts the control directly into the hands of the developer. Instead of "load balancing" the processes across all processing cores, the application is written to target specific processing cores to handle specific processes. Process A can be dedicated to core 1 and process B can be dedicated to core 2. This significantly reduces or eliminates the need for process context-switching. It also allows the system to take advantage of special-purpose processors (network processors, graphics processors, and others) to augment general-purpose processors much more efficiently. This is more applicable to purpose-built appliance computing platforms, such as dedicated routers, Application Delivery Controllers (ADCs), firewalls, and so on.

Both of these methods can significantly improve the performance of an application, but at a cost.

Issues with Typical Multiprocessing Appliances

Both SMP and ASMP have some significant issues—especially when used for dedicated applications—that prevent them from fully utilizing the additional processing power of multiple processors, particularly as the number of available cores increases.

SMP has significant overhead associated with the arbitrary distribution of process execution. First, the scheduling process itself requires processing cycles that are not available to the application for which the device was built. As the number of processing cores increases, so does the number of cycles required to handle process scheduling and inter-process communication. In addition, without specific interaction from the application developer, SMP can have significant overhead when context-switching is required—a very costly, cycle-intensive process. While process scheduling has continued to improve in efficiency and purpose-built appliances generally do not run as many unique processes as multipurpose computing platforms, generic SMP still has significant overhead that affects the available computing power.

The most significant issue for ASMP is the need to rewrite and design the specific application to accommodate multiple processing cores. This can add substantial development time, especially when trying to adapt old code. It also increases the complexity of the software (and thus the cost of the developers) and requires code revisions whenever the number or type of available processing cores changes. For example, if the system goes from dual-core to quad-core processors, it needs to be accounted for. Another drawback of ASMP is that, since processes are not load balanced, a single core might have idle cycles while another is incapable of handling its requests—a probability that increases with the number of cores. The efficiencies gained by eliminating context-switching can be quickly eaten up by the inefficiencies of processor usage or the complexity of development.

This is not to say that neither model does not provide increased processing capability but, rather, that both models suffer from a case of diminishing returns. A dual-processor/core system does not perform twice as fast as a single-core system. Each core added to the system adds a diminishing amount of computing resources, eventually reaching the point at which all the computing power of an additional core is eaten up by managing and implementing that core. This results in no appreciable increase in overall computing power.

This is, to some degree, explained by Amdahl's Law. Named for Gene Amdahl (father of F5's first CTO, Carl Amdahl), Amdahl's Law essentially states that the amount of performance increase that can be expected by parallelizing a process is a factor of the amount of the process that can truly be parallelized. If a process requiring 10 units of time can only be 50 percent parallelized, the process will never run in less than five units, even if the parallelized portion is processed instantly. As a result, the entire process can never be more than twice as fast.

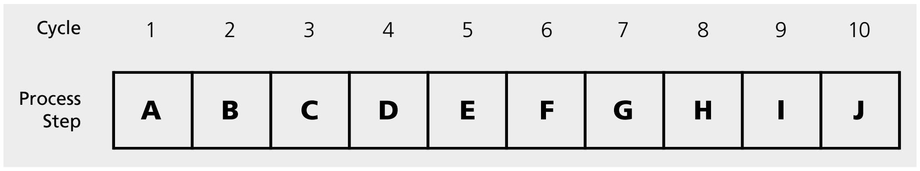
The problem, therefore, is that both traditional multiprocessing methods are tightly coupled, suffer from a shared-memory model, and the need for significant inter-process communications. Regardless of whether you virtualize a single process across multiple processing cores with SMP or attempt to break the process across multiple cores with ASMP, both solutions typically share memory between threads or processes and must allow communication between them. This means that in order to avoid race conditions and data corruption, the entire process must be painstakingly orchestrated—thus, the "tightly coupled" definition. For example, any memory access must issue a lock to prevent other processes or threads from simultaneously acting on the same data. Issuing memory locks is not only expensive in terms of cycle times (if using the same data, other processes must wait until the lock is cleared to continue execution), but the entire system can be throttled by the number of locks that can be maintained per unit of work. If we have to process 1 million transactions per second and take out three locks per transaction, at 300nS per lock, 90 percent of the CPU time is taken up on locking—leaving little for actual transaction processing.

Consequently, while most manufacturers have focused on increasing the multiprocessor capabilities of their products, the tightly coupled nature of both SMP and ASMP has limited the proportion of their systems that can be parallelized. With the remaining serialized portion of the system no longer improving in performance, it is easily seen why most purpose-built appliances continue to see a diminishing return on multiprocessor implementations. They have been continually improving the performance of only part of their system.

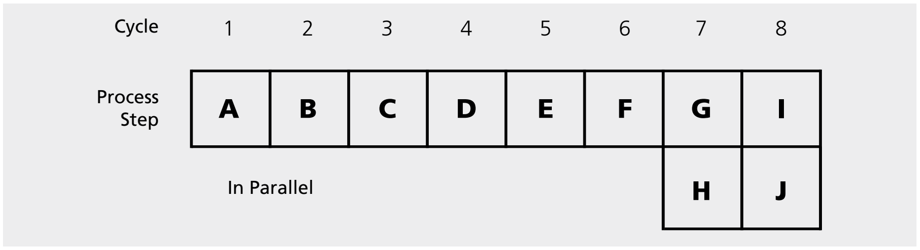
The Logical Solution

If you accept that there is little to be done about the performance improvement of the serialized portion of a system and you recognize the fact that Amdahl's Law demonstrates the futility of continuing to improve the performance of a static parallelized portion, there remains only one way to improve the overall performance of the system in any substantial way. You will need to alter the amount of the process that can be parallelized in proportion to that which remains serialized.

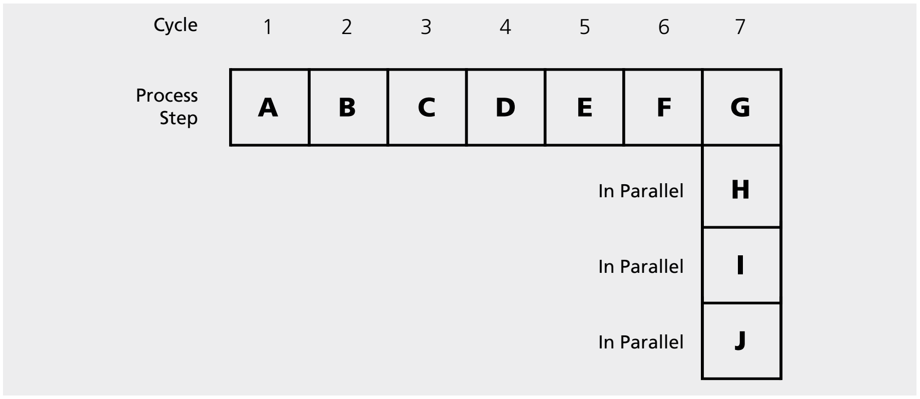
The math is pretty straightforward. Let's imagine a simple 10-step process. A fully serialized version will take 10 cycles to complete:



Now, let's say that the process can be 40 percent parallelized and you have two cores that can execute the process. That might look like this:



Sixty percent, or six steps, must still be done in sequence, but the remaining four steps can be executed simultaneously on two processors (requiring 2 cycles: 2 cycles x 2 processors = 4 steps executed). This process now only takes eight cycles to complete, for a 20 percent improvement in overall performance. However, if you add two additional cores, it would look like this:

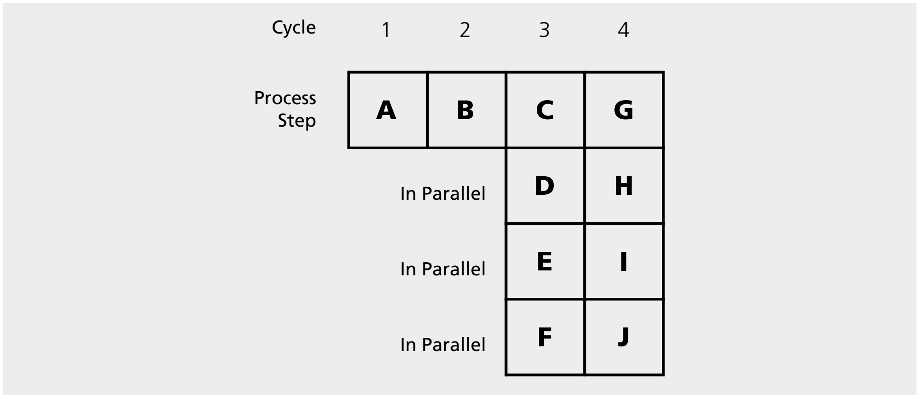


Déjà Vu All Over Again

This problem of parallelization conjures feelings of déjà vu at F5. It is remarkably similar to a problem we've seen—and solved—before. In the early days of Application Delivery Controllers, when they were known as "load balancers," F5 competed against many host-based software solutions. F5 invariably outperformed these systems when the pool of servers exceeded more than a few systems. Why? Because the amount of overhead necessary to communicate state information between the hosts quickly exceeded the performance improvement of adding the additional systems; they suffered severely from diminishing returns.

Although simplistic, the analogy can be made that the servers were much like the processors in a multiprocessor implementation and the state information represented the shared-memory model and inter-process communications implemented by SMP and ASMP. F5's BIG-IP system increased the proportion of parallelization of the system by removing the need for the shared state—significantly improving the performance of the overall system.

The process now only takes seven cycles, which represents a total of a 30 percent improvement over the original serialized process, but only a 12.5 percent increase over the eight-cycle version (seven cycles versus eight). This perfectly demonstrates the reason for the diminished returns. Parallelizing the process (as best as possible) and adding a second core returned a 20 percent improvement, but adding two more cores only returned an additional 12.5 percent improvement. In this simple illustration, adding any more cores will do absolutely nothing to improve performance, as all steps that can be run in parallel already are. If, however, you can make the process 80 percent parallelized, that same four-core system can now run the process in four cycles:



That's an additional 43 percent performance increase, resulting in a 60 percent performance increase from the original serialized system. The difference was not in adding more cores to the system, but from increasing the amount of the process that could be run in parallel to more fully utilize those cores.

The F5 Way: Clustered Multiprocessing

F5 realized early on that simply adopting multiprocessor architectures without addressing the proportion of the process that can be parallelized was a short-term, dead-end street. The company invested heavily in developing a way to increase the parallelization of the traffic management process.

The result of this investment is F5's Clustered Multiprocessing (CMP) architecture. CMP combines the benefits of load balancing and high availability provided by SMP and the efficiency of limited context-switching and special-purpose processor utilization of ASMP. This is all accomplished while eliminating the need

Architecture Comparison

Potential Architecture Issues	Architecture Type SMP	ASMP	CMP
Interprocess/Thread Communication	🔴	🔴	◡
Locking	🔴	◡	🟢
Tendency to Tightly Couple	🔴	◡	🟢
Automatic Scheduling Overhead	◡	◡	◡
Manual Scheduling Overhead	◡	🔴	◡

Major	Minor	None
🔴	◡	🟢

Tendency to Tightly Couple – The more tightly coupled the code, the more inter-process communication overhead there is. The F5 implementation of CMP makes it hard to tightly couple threads/processes.

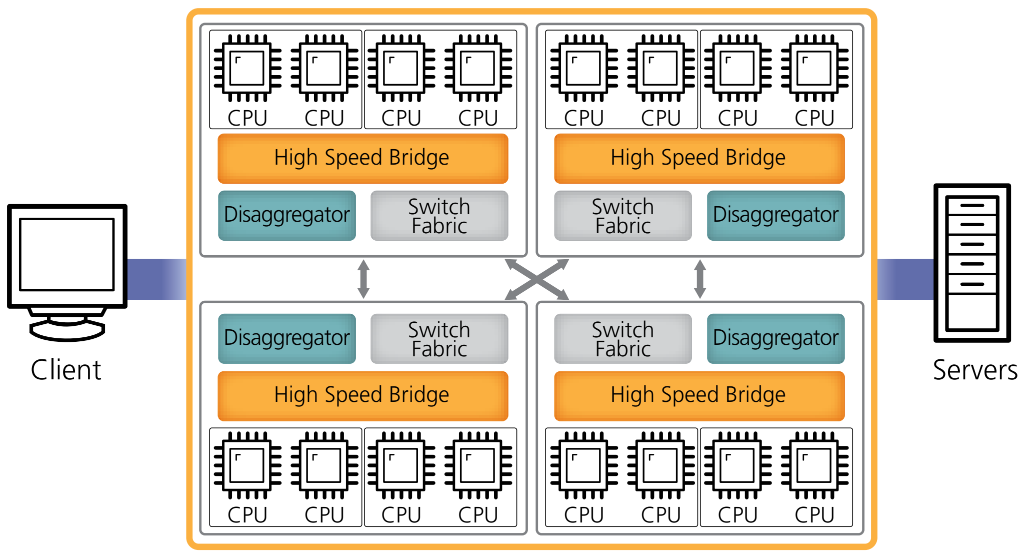
Automatic Scheduling Overhead – This is the scheduling that is done between threads or processes by the kernel. If the number of processes is greater than the number of CPUs, there is a overhead increases.

Manual Scheduling Overhead – This is the re-balancing of the thread/process count for a processing pipeline. It frequently crops up in ASMP designs and sometimes in SMP designs.

for the shared-memory model and reducing the inter-process communications that continue to shackle the performance of other vendors' multiprocessor designs. CMP provides a virtualized processing fabric that delivers industry-changing performance, scalability, extensibility, adaptability, and manageability.

To start with, TMOS, the purpose-built software platform on which F5 products run, is extremely efficient when run on a single core. The Traffic Management Microkernel (TMM) is a single-threaded, non-context-switching process optimized specifically for processing Application Delivery Network traffic. In addition, the TMM is designed to easily facilitate ASMP principles to incorporate performance improvements from special-purpose processors. For instance, when executing encryption processes, the TMM is designed to do it on the general-purpose processor (in software), but if an encryption coprocessor is present, it can offload it to the special-purpose processor. It does this without any change in operation other than the increased performance of the dedicated hardware. The TMOS platform, which F5 also spent significant time and resources developing, consistently outperforms other products in the marketplace and remains the core of CMP.

From that basis, most manufacturers would simply attempt to use SMP to distribute TMOS process across multiple processors—with shared memory, network card, and special purpose processors. Others might attempt to run multiple instances of the TMM on different processors—still with the requisite shared memory, network card, and special-purpose processors. Instead, CMP enables load balancing of multiple processing cores, each with its own dedicated memory, network interface, and special-purpose processors. Each core runs its own, completely independent TMM process. By separating the dependencies between the instances, CMP allows more of the traffic management process—virtually the entire process—to be parallelized. This provides a substantial benefit to the overall performance of the system. The hardware that enables CMP is comprised of two important, proprietary F5 technologies: the Disaggregator and the High Speed Bridge (HSB).



The Disaggregator acts as a hardware-based load balancer, distributing traffic flows between the independent TMM instances and managing flow affinity if or when necessary. Not only does this facilitate a near 1:1 linear performance growth (doubling the number of processing cores nearly doubles the computing power with no diminished returns), but it completely virtualizes the processing cores from the system and the other cores. This provides high availability and reliability in the event that any core becomes non-functional. In the VIPRION chassis, this includes the addition and/or removal of entire blades and their associated processing cores.

The HSB delivers direct, non-blocking communication between the TMM instances and the outside world without the loss normally associated with Ethernet interconnects. It also provides the streamlined message-passing interface that enables TMM instances to share information. This provides the unsurpassed throughput and interconnectivity of each processor's dedicated network interfaces. It also mitigates the performance impact of inter-process communications in the few remaining instances where it takes place. Again, in the VIPRION chassis, this facilitates efficient traffic distribution and message-passing between blades as well as within the cores of each blade.

Changing the Rules

Up until now, the game has been pretty simple—and widely understood. First, it was to optimize your code to run on a single processor as best you can and ride the "Intel power-curve." Then, it was to optimize your code for SMP or ASMP and then build your platforms with as many processing cores as possible. All the while, performance improvements have slowly dwindled to miniscule amounts.

CMP changes the rules of the game. Instead of working to continually improve the performance of a never-changing proportion of parallelized processes, CMP's most basic tenant is to change that proportion. Continuing improvements in performance can only be realized by increasing the amount of the application delivery process that can be parallelized. Only parallelizing nearly all of that process can enable near 1:1 linear scaling—fully utilizing all the processing cores.

In much the same way that F5 redefined the load balancer at the turn of the century with the implementation of SSL offload—starting the evolution of Application Delivery Controllers—CMP redefines the Application Delivery Controller. The ADC is no longer limited by processing capability or network throughput. It is now free to grow with the needs of the organization and has the scalability to adapt to new, unforeseen functionality down the road—all within a single, easy-to-manage package. CMP, in combination with TMOS, provides F5 customers with the scalability, extensibility, adaptability, and manageability to consolidate the data center of today and future-proof the data center of tomorrow.

F5 Networks, Inc.
Corporate Headquarters
401 Elliott Avenue West
Seattle, WA 98119
+1-206-272-5555 Voice
(888) 88BIGIP Toll-free
+1-206-272-5556 Fax
www.f5.com
info@f5.com

F5 Networks
Asia-Pacific
+65-6533-6103 Voice
+65-6533-6106 Fax
info.asia@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
+44 (0) 1932 582 000 Voice
+44 (0) 1932 582 001 Fax
emeainfo@f5.com

F5 Networks
Japan K.K.
+81-3-5114-3200 Voice
+81-3-5114-3201 Fax
info@f5networks.co.jp