Achieving Nanosecond Latency Between Applications with IPC Shared Memory Messaging

In some markets and scenarios where competitive advantage is all about speed, speed is measured in micro- and even nano-seconds. In these situations architects and developers are always looking for new technologies and techniques that will help them squeeze every last bit of latency out of their systems.

One such technique is called Inter Process Communications (IPC), a way of enabling communications between applications that are co-located on a single multi-core system. This eliminates the latency associated with sending information across the network between computers.

This paper summarizes the concepts of IPC and shared memory messaging, and introduces Solace's IPC solution, focusing on its best-in-class performance and the advantages of being part of Solace's unified platform.

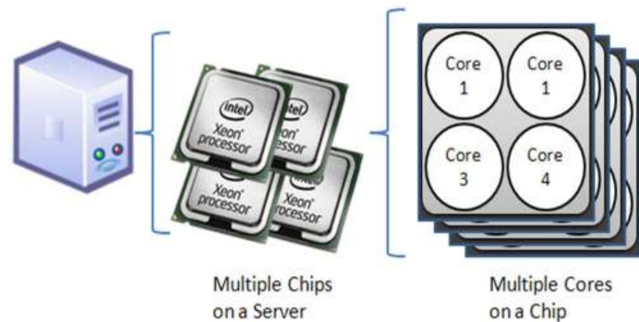
This paper also presents the results of performance tests that show the speed of Solace's solution under a variety of conditions running on Cisco Unified Computing System hardware. Key test results for application-to-application testing between cores include:

- One-way latencies as low as 388 nanoseconds (average) and 480 nanoseconds (99th percentile) at 2.97 million messages per second between two cores of the same CPU.
- Using the full capacity of a single Cisco Unified Computing System server with 2 CPUs, each with 6 cores, testing achieved a maximum of 46.8 million messages per second and154.5 gigabits per second of total throughput.

Copyright © Solace Systems, Inc.
http://www.solacesystems.com

Introduction to IPC and Shared Memory Messaging

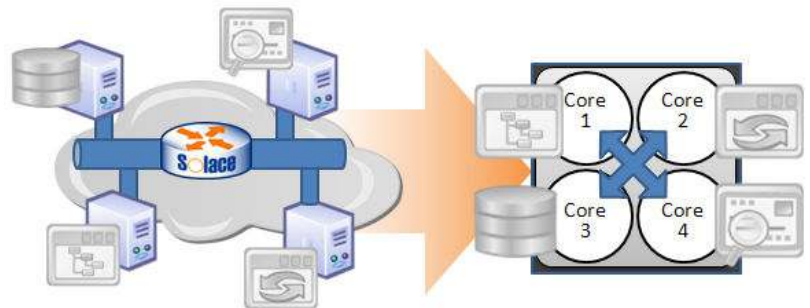
As processor speed are reaching their limit with current technology, processors with more and more cores are being produced and motherboards with multiple processors are becoming common, so you can now have 32 processors on a single motherboard. This allows multiple high performance applications to be run on a single server. These co-located applications can communicate with each other very quickly because cores on a given CPU or on the same motherboard share memory. This means "sending" applications can tell "receiving" applications to get data from memory, instead of physically sending it across the network.



This is usually done with a mechanism called Inter Process Communications (IPC), and it eliminates the latency and pitfalls of network connectivity. Done properly it can simplify deployments while enabling dramatic acceleration that's valuable wherever having the lowest possible latency represents competitive advantage, such as high-frequency trading on Wall Street usually in co-location facilities.

IPC Use Cases

Application deployments optimized for shared memory between multiple cores are far from a replacement for networked systems. Only a select few applications can leverage this configuration where either latency or throughput demands are so extreme that including a network becomes a hindrance. Two examples are high frequency trading in financial services and high-performance computing used to analyze data streams in the scientific community.

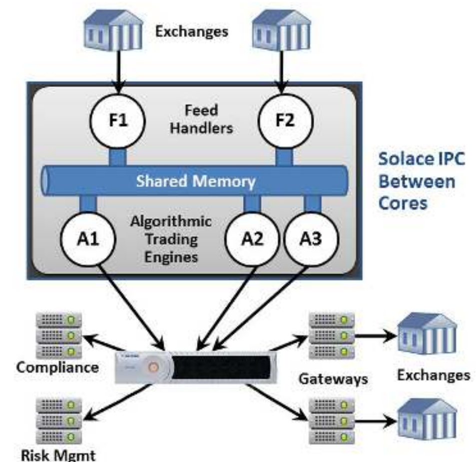


- **Algorithmic / High Frequency Trading:** The basic input to any trading system is market data feeds that consist of real-time pricing data for equities, options, foreign exchange or fixed income. There are usually separate engines for each trading strategy, and other components such as risk management and order routing applications. Architects use fast multi-core servers for algorithmic trading by configuring some of the cores to be feed handlers, and some of the cores to be trading engines. Performance is often further accelerated by renting co-location space directly at an exchange to minimize the distance and therefore the latency delay associated with receiving pricing data and entering orders.
- **High Performance Computing:** Many scientific applications such as oil and gas exploration analysis, scientific data analytics and weather forecasting perform highly-interdependent tasks resulting in distributed architectures that stress the throughput limits of the networks between them. Deploying a large-scale multi-core machine with shared memory allows the applications to interact more with greater ease and higher throughput, resulting in complex tasks being completed more quickly.

About Solace's IPC Solution

Solace has extended its Unified Messaging Platform to support IPC messaging in a shared memory environment.

- **Fastest IPC Available**: With average latency of around 700 nanoseconds, Solace's solution is the fastest available. In the interest of full disclosure and repeatability, the parameters and results of the tests that demonstrated this performance, as well as performance in a variety of configurations, are included in this paper.
- **Redeployment Without Redevelopment**: Applications that currently use Solace's API for ultra low latency messaging can be redeployed in a shared memory configuration by simply reconfiguring the communications settings, without any additional development work.
- **The Unified Messaging Platform Advantage**: Although trading applications aim to capture data and make decisions on a single machine, an entire trading platform usually doesn't fit on a single server so these applications inevitably have to interact with applications running on other systems. Since Solace's API and message routers support low latency, high fanout, guaranteed and WAN messaging in a distributed environment, applications can share data with co-located applications using IPC and with remote applications using other messaging types. Therefore the location of applications can easily be made a runtime deployment option. Solace's IPC solution can be used standalone for shared memory messaging or in conjunction with Solace Message Routers handling guaranteed and reliable distributed messaging.



Deployment Options

The memory management of CPUs varies greatly, as does the architecture of applications. Two factors affect the latency of shared memory messaging: the sharing of memory caches and the configuration of the application.

Memory Caching

A cache is very high speed memory located directly on the die of a CPU. Memory caches allow very fast access, but require lots of power and can't be made very dense, so you can only put a small amount of memory on the chip itself. Over time, layers of caches have been added, with small level 1 caches being the fastest, then bigger but slower level 2 caches, and now some processors even have level 3 caches.

- **Caching in Multi-core CPUs and Multi-chip Servers:** In multi-core processors each core frequently has its own cache, so when one core changes a memory location, other cores with that location in their caches must update theirs or they could read old data. This has led to "cache coherency" protocols that keep all the caches synchronized. Some CPUs use a discrete cache for each core, and on some the slower caches are shared among cores. The Intel Xeon E5450 CPU we used has a separate level 1 cache for each core, while two cores share one level 2 cache and the other 2 cores share a different level 2 cache. On Intel's Nehalem chip, some cores share a level 3 cache, but L1 and L2 are separate. The configuration that enables the most value is also the most complicated to architect for: a motherboard running multiple processors, each with multiple cores. Today's servers are available with up to four quad-core processors, for a total of 16 cores. In such a server, processors do not share a common cache, so it takes longer for applications to access the memory of a different chip than the one it's running on. So now you have to optimize what core is accessing which part of the external memory as well, and doing IPC between cores that share a cache will be faster than between cores that do not , for example cores on different chips.

- **Hyper-threading:** Hyper-threading is an Intel-proprietary technology used to improve parallelization of computations performed on CPUs. A processor with hyper-threading enabled is treated by the operating system as two processors instead of one, which means that only one processor is physically present but the operating system sees two virtual processors, and shares the workload between them. In a hyper-threaded configuration each application runs a little bit slower, but overall the system can perform more operations per second. That makes hyper-threading ideal for things like Web servers, and not as important for applications where latency is critical.

Blocked vs. Spinning Applications

There are two ways applications can be configured to wait for work to be done or messages to be processed:

- **Blocked:** Applications can be "blocked" by the operating system which means they are on standby, not utilizing any CPU resources until called upon to perform some operation. When an application blocks waiting for work (e.g. waiting on a semaphore or issuing a select() call) then some other application needs to perform an operating system call to activate it. That causes it to make a context switch into the OS, which needs to schedule the blocked application in a core, deschedule the application currently executing in that core, reload state for the blocked application, then execute it. This so called "scheduling latency" takes time away from processing the critical event.
- **Spinning:** Applications can also be configured as "spinning" which means they are constantly running in a core even when not performing any useful work, polling one or more memory locations for work to do or a message to service. This lets the application execute that event or message immediately without incurring the latency of context switching and scheduling. This means you need as many cores as you have critical application threads, and until recently the lack of sufficient cores per motherboard made this architecture difficult to achieve.

Performance Testing Results

This section describes performance tests run on Cisco Unified Computing System hardware, along with methodology and configurations of each test. Anyone with similar hardware and Solace IPC software should expect similar results.

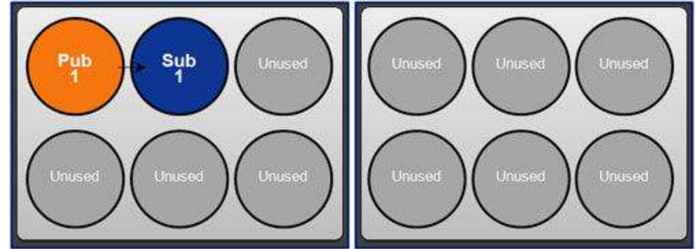
- **Hardware:** Cisco C200-M2 Server with 2 Intel Xeon 5670 (Westmere) 2.93 GHz processors with 48 GB RAM (12 x 4 1333 MHz DIMMS) and six cores each. Publishers and subscribers were locked onto cores using the Linux "taskset" command, and hyper-threading was turned off so each application had dedicated use of a core.
- **OS:** Red Hat Enterprise Linux 5.4 release with kernel-2.6.18-164.el5.
- **Testing Tool:** A Solace tool called SDKPerf was used to generate traffic, and to collect latency measurements on a subset of the messages in order to minimize the impact of sampling the system clock.

A series of tests were run to determine performance boundaries while simulating configurations and functionality required in real world use cases: Single-stream tests measuring latency and throughput at various messages sizes, and multi-stream tests of simultaneous fan-out and fan-in at various message sizes

Test	Publishers: Subscribers	Byte Ranges	Number of Cores	Lowest Latency	Highest Message Rate	Highest Total Throughput
Latency & Message Rates	1:1	16-1,024	2	388	2.97M msg/sec	N/A
Fan In / Fan Out	6X6	16-16,384	12	N/A	46.8 M msg/sec	154.5 Gb/sec (at 8K msg size)

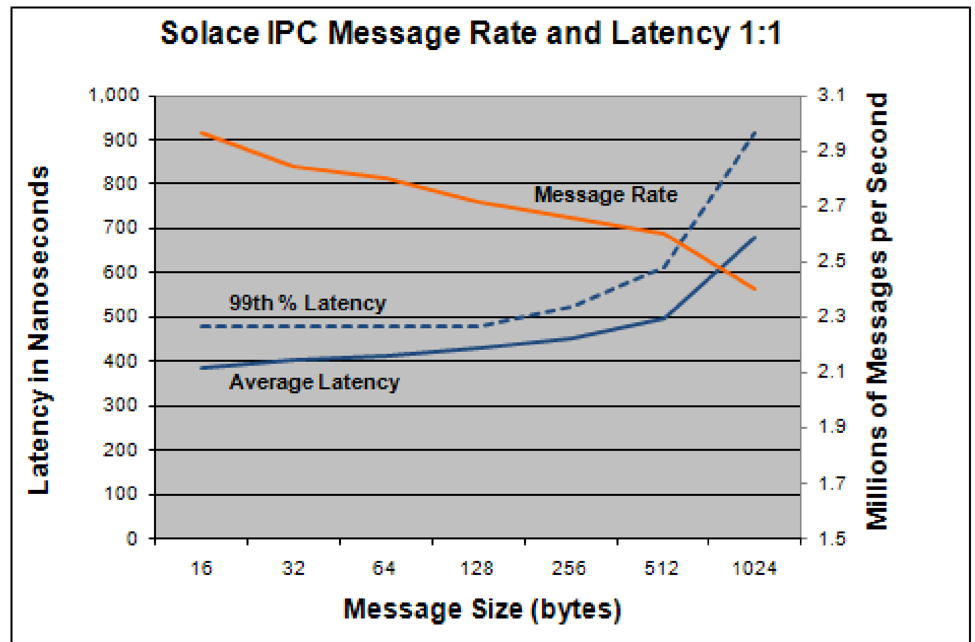
Latency and Throughput in 1:1

The first test shows latency and message rates for one core publishing to a second core on the same CPU. Each test captured the average and 99th percentile latency for a range of message sizes as well as total throughput in number of messages. All tests were performed on an optimally configured Cisco Unified Computing System server running the Solace API using the Solace IPC transport for multi-core, shared memory messaging.



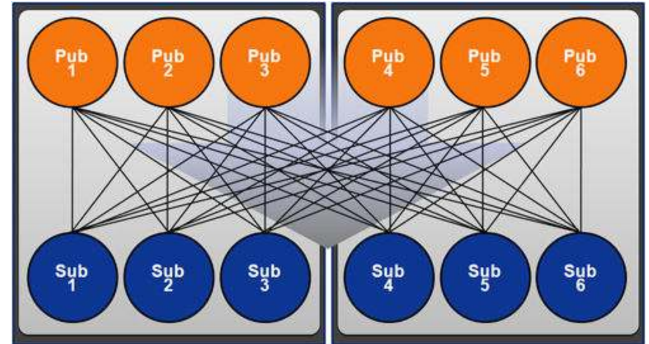
Testing was done with 1 publisher and 1 subscriber for message sizes ranging from 16 to 1024 bytes for 5 minutes. Only 2 of the available 12 cores were used since latency at volume was the requirement, not total throughput. We expect latency to stay uniform when running parallel publisher/subscriber pairs bound to different CPU pairs.

Results and Observations



- As one would expect, smaller messages exhibited lower latency and higher message rates.
- For 16 byte messages, the Solace IPC API published 2.97 million messages per second with an average latency of 388 nanoseconds and a 99th percentile latency of 480 nanoseconds.
- At 128 bytes, eight times the payload size of the first tests and reasonably close to normalized market data update size, performance was remarkably similar. Message rates were 2.71 million messages per second with an average latency of 431 nanoseconds and a 99th percentile latency of 480 nanoseconds.
- Increasing payload to 1024 bytes, a full 64 times as large as the initial test, still produced throughput of 2.40 million messages per second with average latency of 679 nanoseconds and 99th percentile latency of 916 nanoseconds.
- In each test above, even all latency results were under 1 microsecond up to the 99th percentile.

Message Throughput in Fan-out / Fan-in Scenarios 6X6

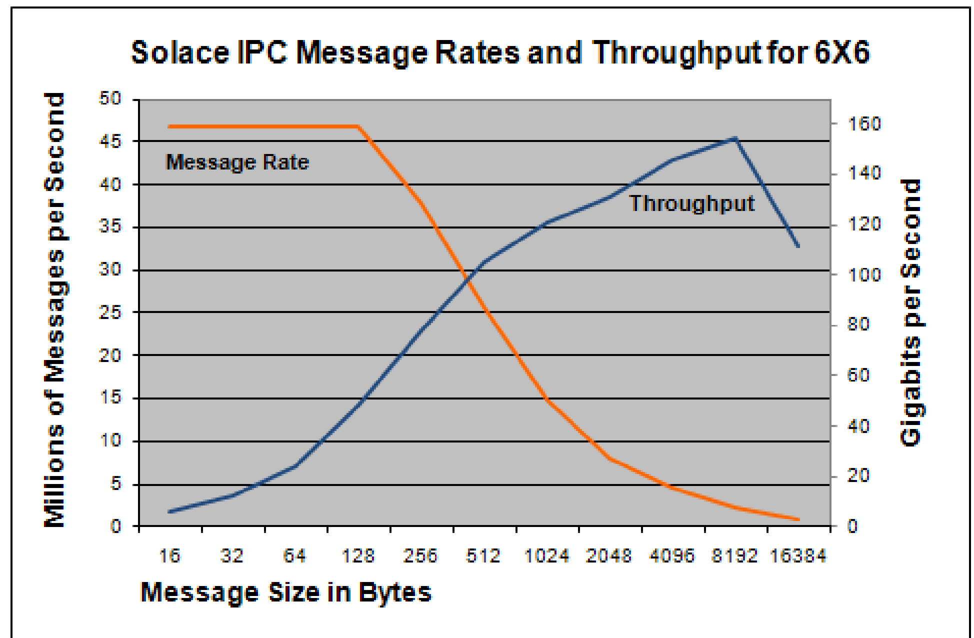


For the message throughput test, the cores were allocated evenly to simulate a use case that performs both fan-out and fan-in. Each publisher sent messages to six different subscribers (a fan-out configuration) and simultaneously each subscriber received messages from six different publishers (a fan-in configuration).

As described in the use case section, a common configuration for algorithmic trading is for multiple feeds to publish simultaneously to multiple algorithmic trading engines in a full mesh configuration. The tests measured total number of messages and bandwidth in gigabits per second for a range of message sizes between 16 and 16384 bytes on a 12-core Cisco Unified Computing System server running the Solace API and the Solace IPC transport for shared memory messaging.

Testing was done with 6 publisher applications and 6 subscriber applications (one per core) for message sizes ranging from 16 bytes to 16384 bytes (16K) for a duration of 5 minutes. All 12 cores were used to maximize single server throughput. Each publisher sent messages to all 6 subscribers and each subscriber received messages from all 6 publishers.

Results



- The maximum total message rate between publishers and subscribers is consistent at between 46.7 and 46.8 million messages per second, for all message sizes from 16 to 128 bytes. This result is so consistent because memory copies have approximately the same CPU cost for message sizes up to about 128 bytes.
- All rates above 128 message rates take longer to copy each message into memory buffers and the results show steady reduction in total messages delivered and corresponding increase in total data processed.
- The peak data rate is achieved when message sizes reach 8096 bytes with a total of 155.47 gigabits per second.