informatica

White Paper

Informatica Ultra Messaging SMX Shared-Memory Transport

Breaking the 100-Nanosecond Latency Barrier with Benchmark-Proven Performance

This edition published June 2013

Table of Contents

Executive Summary

Informatica® Ultra Messaging® is software that significantly improves communication for electronic trading applications. Its high performance and architectural innovation have advanced business strategies and trading profitability throughout the capital markets industry.

Furthering its position as a leader, in May 2013, Informatica Ultra Messaging (formerly 29West) released a new shared-memory communication transport: Shared Memory Acceleration (SMX). SMX is optimized for high-speed electronic trading applications requiring the absolute lowest message latency and highest throughput. It breaks the 100-nanosecond latency barrier and achieves ultra-high throughput that scales linearly with increased message size and additional consumers.

This white paper begins with a summary of the newly released SMX shared-memory transport and its most common electronic trading use cases. It then describes the purpose, conditions, and methodology of benchmark tests conducted by Ultra Messaging engineers. Full test results of latency and throughput with C, Java, and C# applications are then reported in graphs and numeric tables, demonstrating the superior streaming messaging performance of SMX. The results show dramatic improvement in performance over the previous Ultra Messaging shared-memory benchmark completed in May 2010—specifically, a 16.5x decrease in latency and an 11x increase in aggregate throughput for like-kind test parameters.

With these tests, Informatica Ultra Messaging has demonstrated:

- Lowest latency messaging in the industry, as low as **39 nanoseconds** from sender to receiver
- **Sub-100-nanosecond** latency across C, Java, and C# in hyperthreading scenarios
- Aggregate throughput to multiple receivers of **320 million messages per second**
- **Linear scalability** across multiple CPU cores for both latency and throughput

The Informatica Ultra Messaging SMX benchmark sets a new standard, showing that Ultra Messaging is the fastest and highest performing messaging software in the industry.

The delivery of SMX as a core part of the Ultra Messaging software product family, along with the conducting of this benchmark, is a continuation of the significant innovation Informatica has shown in the use of shared-memory transports for inter-process communication (IPC) within the same physical server. These transports take advantage of parallel computing architectures supported by multi-core CPUs to accelerate the transfer of data from application to application. They also further enable the trading strategy of combining market data feed handlers with trading algorithms on the same host in order to respond to market changes as fast as possible.

Additionally, due to its unprecedented performance, SMX is the first shared-memory transport that can effectively be used for inter-thread communication (ITC) as well as IPC, and it provides a superior level of performance natively for all three major application development languages: C, Java, and C#.

SMX Transport Description and Electronic Trading Use Cases

SMX is new to the family of Informatica Ultra Messaging transports that trading firm customers can use for their application deployments. It is a complete redesign that supplements the existing IPC transport (LBT-IPC) and enables the software to take full advantage of the design and capabilities of modern x86 64-bit multi-core architecture CPUs. (SMX is only supported on 64-bit processors and applications.)

SMX is a zero-copy transport that enables direct reads and writes into the same, shared-memory space without contention and synchronization overhead. It supports single-threaded (lock-free), single-writer, and multiple-reader patterns with zero-copy APIs and reliable streaming (receiver-paced) delivery. SMX supports fully native C, Java, and C# implementations, is designed for many-core machines, and requires a dedicated CPU core for each SMX receiver thread (just as the existing LBT-IPC transport does).

In the Informatica Ultra Messaging family of transport protocols, SMX is the core low-latency transport. As shown in the benchmark tests that follow, it delivers sub-100-nanosecond latencies and extremely high aggregate throughput rates. For these reasons, Informatica believes SMX will be of particular interest in trading firm strategies where having the fastest and highest performing platform is critical for profitability.

These strategies include:

- High-frequency trading employed by proprietary trading firms and hedge funds
- Market makers on either side of a trade who commit to streaming executable prices that are posted electronically and can change instantaneously
- Arbitrage strategies that rely on speed to exploit market price discrepancies
- Algorithmic trading that relies on new market price updates being rapidly streamed from exchanges or consolidated market data feeds as quickly as possible
- Smart order routing that relies on speed to send orders as quickly as possible to liquidity venues that offer the best prices

Due to its unprecedented performance, Informatica Ultra Messaging SMX also changes the game for how messaging software can be utilized. Traditionally, messaging software has been used for communication between different application processes. Although any shared-memory transport can theoretically be used also for communication between threads in the same application (inter-thread communication or ITC), developers have historically not considered messaging as a tool for passing data between threads. This is because of the relatively high latency of messaging transports compared to traditional data passing techniques such as sharing data structures across threads or putting data on internal queues to be processed by other threads.

But Informatica Ultra Messaging SMX has shifted this paradigm, becoming the first messaging transport to offer developers a compelling solution for both ITC and IPC. This closes the latency gap between processes and threads and has the potential to significantly reshape application and system design.

Furthermore, SMX is the only shared-memory transport to deliver this high level of performance natively and consistently for all three major application development languages—C, Java, and C#—with interoperability among them. This is important first because high-performance systems are often developed in multiple languages (e.g., market data feed handlers written in C; algo engines written in Java). It's significant also because the high performance and seamless interoperability that SMX provides across all three languages allow developers to use the best features of each language without sacrificing performance.

SMX Performance Benchmark

Purpose and Performance Test Conditions

The purpose of this SMX performance benchmark is to demonstrate in a reproducible and transparent way the extremely low-messaging latency and high-messaging throughput achievable using the new shared-memory SMX transport on a single server. The benchmark also demonstrates the advantages of hyperthreading in Intel processors, which have improved tremendously in recent years, and how SMX takes full advantage of this capability.

Three tests were performed for each of the three major application development languages—C, Java, and C#: two tests measured average latency from one sender to one receiver across a range of message sizes; one test measured aggregate throughput (messages per second) from one sender to an increasing number of receivers.

- **Average latency (nanoseconds) relative to message size: core to hyperthread**
    - Range of latencies is from 39 nanoseconds for 16-byte message sizes in C to 135 nanoseconds for 512-byte message sizes in C#
    - Range of R2 coefficient of determination values (indicating degree of linear regression fit to data) is 0.9613 for Java to 0.9974 for C# (a value of 1 equals perfect linear fit)
- **Average latency (nanoseconds) relative to message size: core to sibling core**
    - Range of latencies is from 103 nanoseconds for 16-byte message sizes in C to 173 nanoseconds for 512-byte message sizes in Java
    - Range of R2 values is 0.9761 for Java to 0.9916 for C
- **Aggregate throughput (millions of messages/second): 1 sender to 1-19 receivers (C & Java); 1 sender to 1-7 receivers (C#)**
    - Highest throughput for C (with batched message delivery) is 320 million messages/second at 16 receivers
    - Highest throughput for Java (with batched message delivery) is 286.1 million messages/second at 18 receivers
    - Highest throughput for C# is 137.3 million messages/second at 7 receivers (identical results for batched and non-batched message delivery)

In these tests, senders and receivers were mapped to individual "logical" cores (meaning a single hyperthread within a given core). Hence, the average latency measurements for core-to-hyperthread scenarios represent SMX communication between two different application threads in a single core. Similarly, the average latency for core-to-sibling-core scenarios reflects SMX communication between two different application threads in separate cores. For technology stack configurations, see Appendix B.

Performance Test Methodology

Testing was performed to determine the maximum performance of the technology stack for two simple benchmarks: lowest latency (within a single core and across multiple cores on a single CPU socket) and highest aggregate throughput from one sender to multiple receivers on a single CPU (with and without batched message delivery).

Latency

For all three development languages, the latency tests were measured as the round-trip time (RTT) divided by 2 of sending messages from a single sending process to a single receiving process and back again (Ping-Pong Test). Latencies are shown in nanoseconds and were calculated as the average latency relative to message size for both hyperthreading communication (between two threads in a single core) and communication between a thread in one core and a thread in a sibling core on the same socket/CPU.

Latency measurements were performed using the Ultra Messaging *lbmlatping* and *lbmlatpong* applications. The ping application generates messages and sends them to the pong application, which simply reflects the messages it receives back to the sending application. The ping side uses the same settings as the pong side to ensure that both sides of the round-trip are equal. For the C version of the test applications, two time stamps were taken for each message, one from just before sending a message from the ping application and another immediately upon receiving that message back from the pong application. For the Java and C# versions of the test applications, a time source with a resolution high enough to measure individual message latencies in nanoseconds was not available natively from the run time, so time stamps were taken before sending the first message in a small group of messages and upon receiving the last reflected message in the group. The average latency for messages in each group was then computed by dividing the total time per group by the number of messages in the group. These round-trip times were then divided by 2 to get the approximate one-way latency.

Aggregate Throughput

Throughput is measured as a rate of millions of messages delivered per second and shown in aggregate as the number of receivers scaled from 1 to 19. This represents an application, which is on 1 thread in a single core, publishing messages that are subscribed to by the other 19 threads on a 10-core CPU socket (see Appendix B for hardware configuration). The throughput graphs show how the aggregate messaging rate scales linearly for 16-byte messages (with and without batched message delivery) as the additional receivers incrementally subscribe to the message stream.

Throughput measurements were performed using the *lbmlatsrc* and *lbmlatrcv* applications. *lbmlatsrc* generates a series of messages, as fast as it can, and publishes them to an SMX shared memory segment. Meanwhile, *lbmlatrcv* reads messages from the shared memory segment, also going as fast as it can.

The purpose of throughput testing with a shared-memory transport is essentially to demonstrate a 1:1 linear growth in message rate as CPU cores are added. This amounts to measuring the throughput of a 1:1 fan-out, and then adding receiving applications until the test system runs out of cores, as well as measuring the relative aggregate increase in messages delivered.

Performance Test Results

The test results are extremely impressive. They show an order of magnitude of improvement over the previous Ultra Messaging shared memory performance benchmark completed in May of 2010—specifically a 16.5x decrease in latency and an 11x increase in aggregate throughput for like-kind test parameters. The following figures supply graphs and descriptions of each test result by development language. For the numeric data of these test results, see Appendix A.

Results for C Latency and Throughput

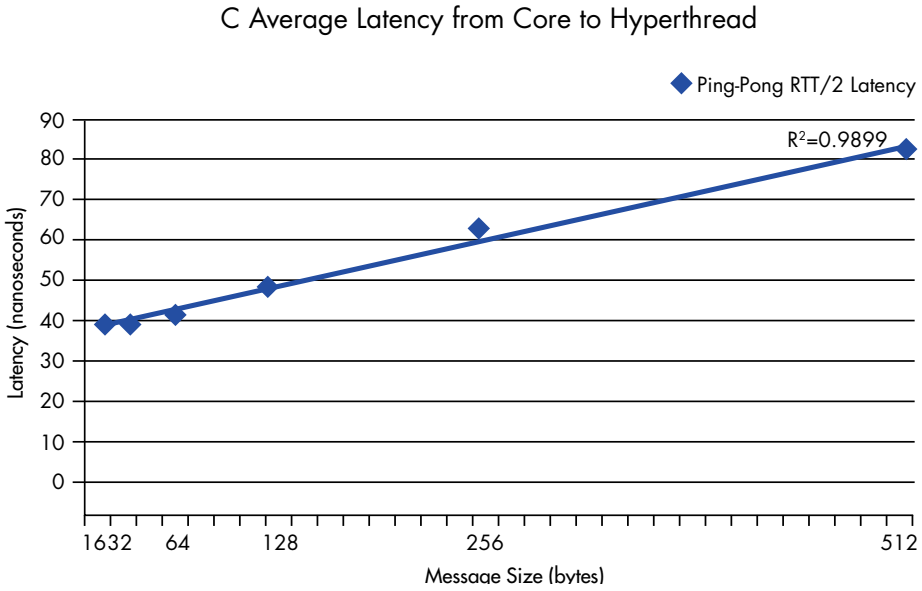

Figure 1. Average latency for messaging on C applications scales linearly from 39 nanoseconds for 16-byte message sizes to 81 nanoseconds for 512-byte message sizes for hyperthreaded communication between processes on a single core.

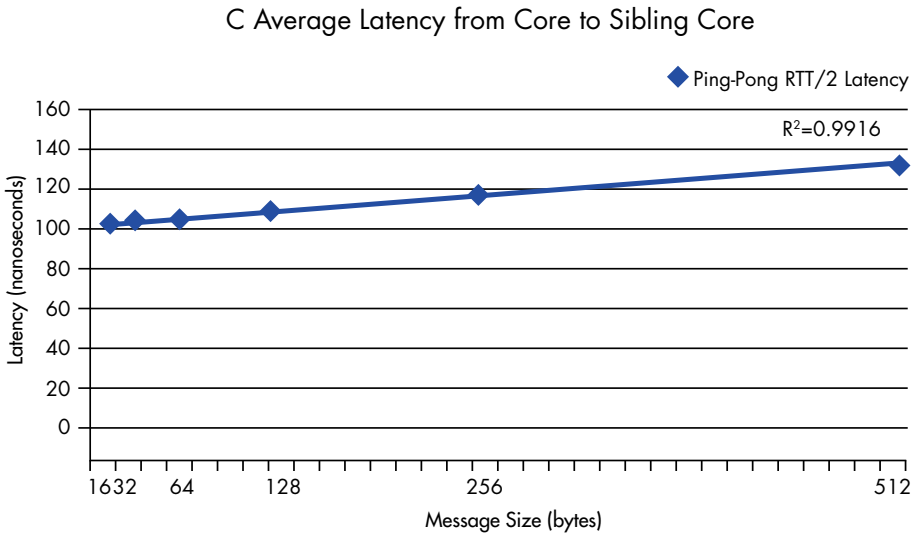

Figure 2. Average latency for messaging on C applications scales linearly from 103 nanoseconds for 16-byte message sizes to 135 nanoseconds for 512-byte message sizes for communication between processes on different cores on a single CPU socket.

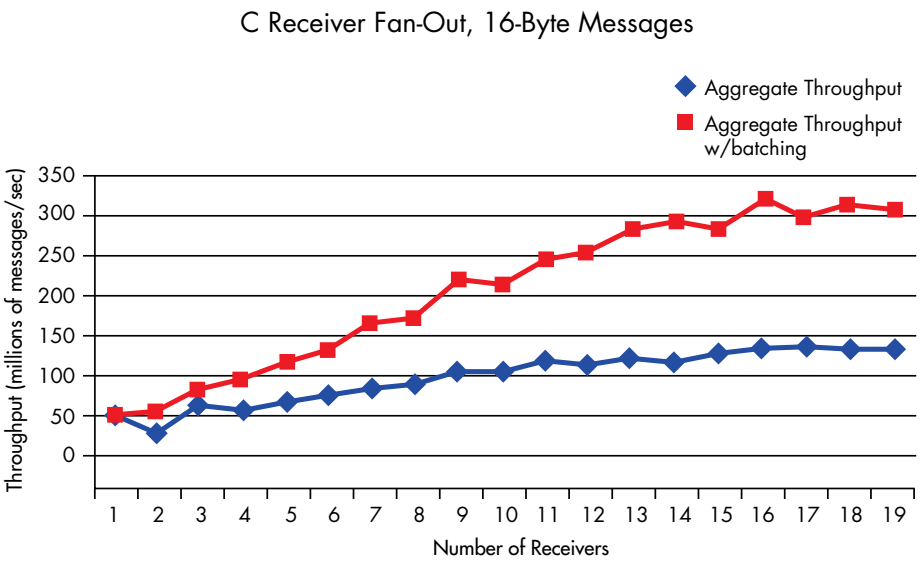

Figure 3. Highest aggregate throughput (with batched message delivery) for C application receiver fan-out is achieved at 16 subscribing receivers with a rate of almost 320 million messages per second.

Results for Java Latency and Throughput

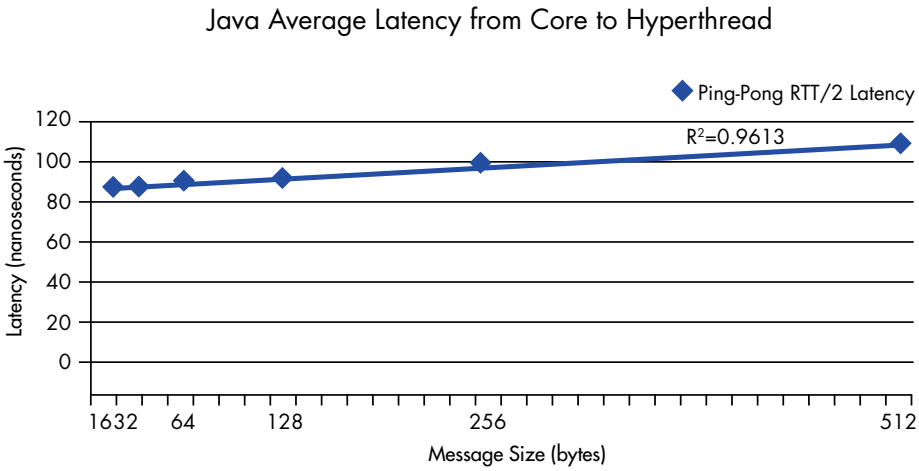

Figure 4. Average latency for messaging on Java applications scales linearly from 87 nanoseconds for 16-byte message sizes to 106 nanoseconds for 512-byte message sizes for hyperthreaded communication between processes on a single core.

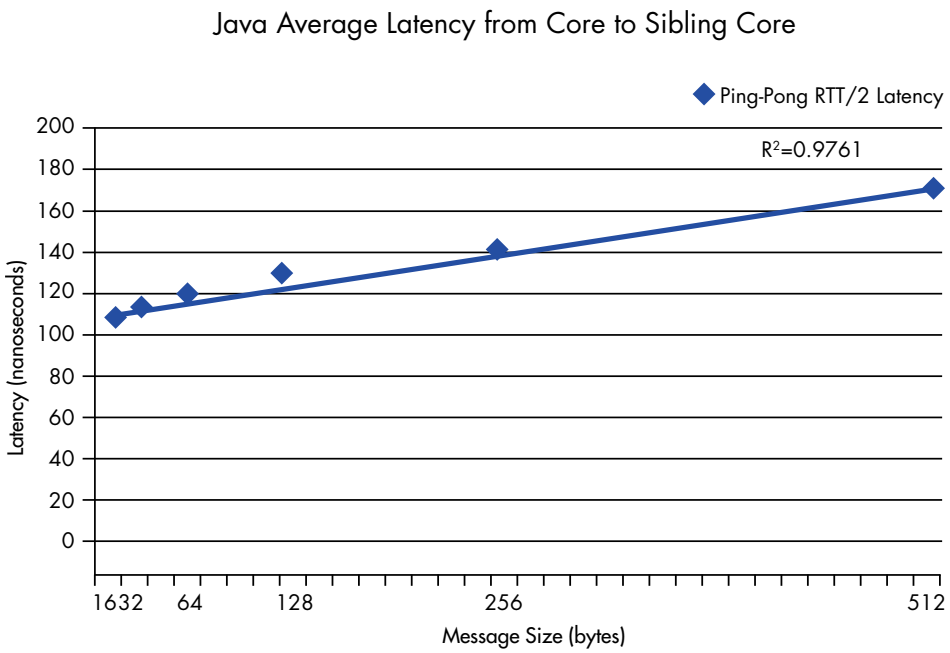

Figure 5. Average latency for messaging on Java applications scales linearly from 108 nanoseconds for 16-byte message sizes to 173 nanoseconds for 512-byte message sizes for communication between processes on different cores on a single CPU socket.

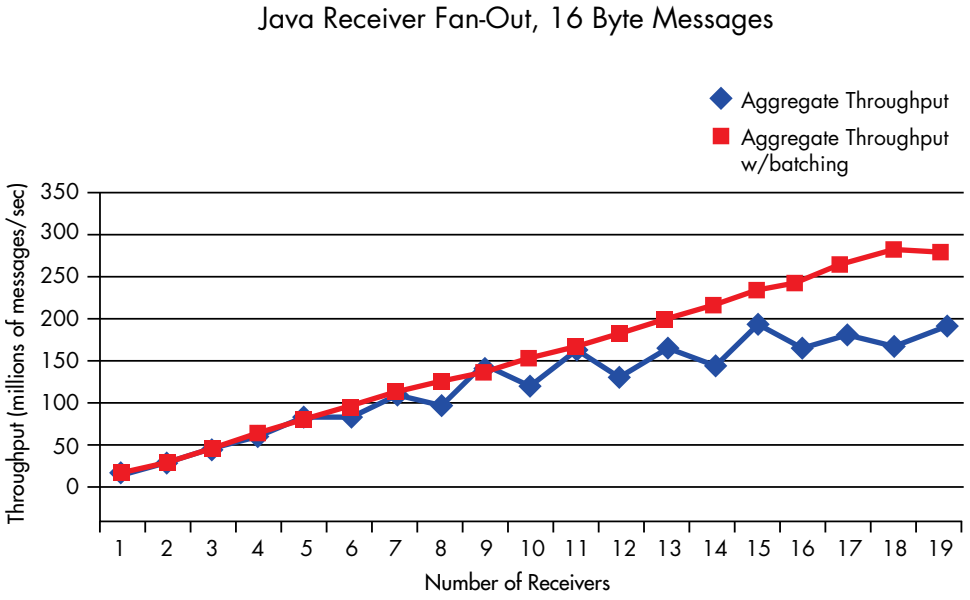

Figure 6. Highest aggregate throughput (with batched message delivery) for Java application receiver fan-out is achieved at 18 subscribing receivers with a rate of 286.1 million messages per second.

Results for C# Latency and Throughput

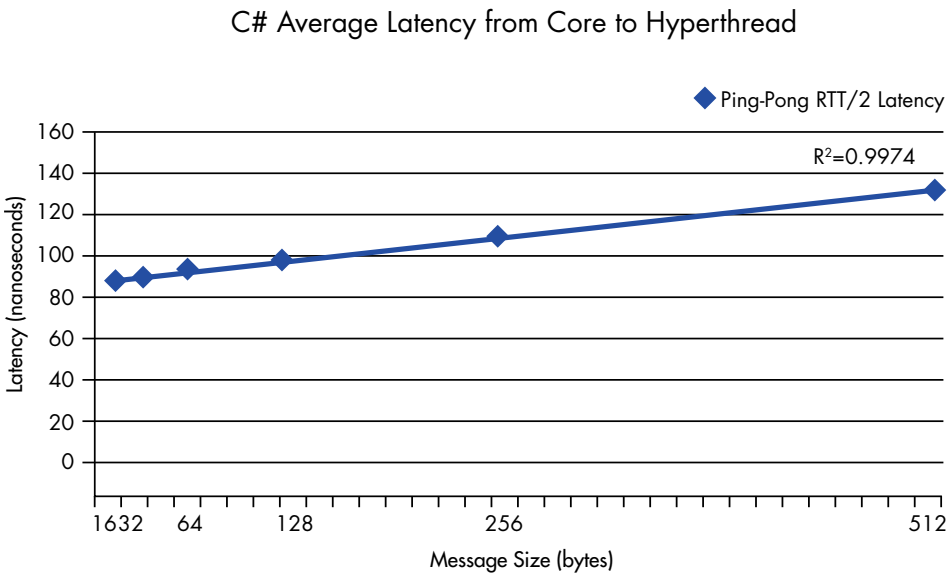

Figure 7. Average latency for messaging on C# applications scales linearly from 86 nanoseconds for 16-byte message sizes to 135 nanoseconds for 512-byte message sizes for hyperthreaded communication between processes on a single core.

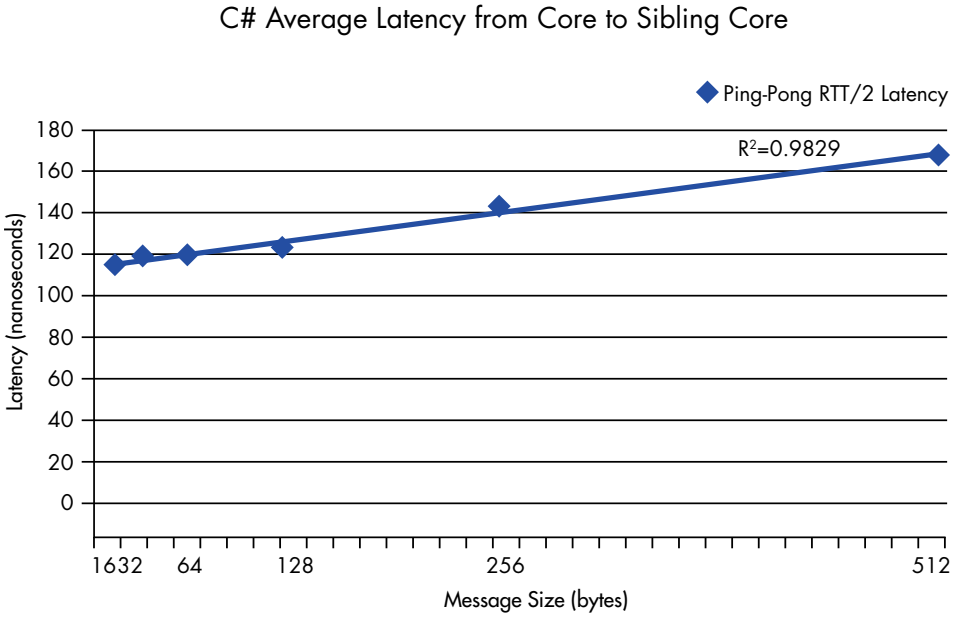

Figure 8. Average latency for messaging on C# applications scales linearly from 114 nanoseconds for 16-byte message sizes to 167 nanoseconds for 512-byte message sizes for communication between processes on different cores on a single CPU socket.

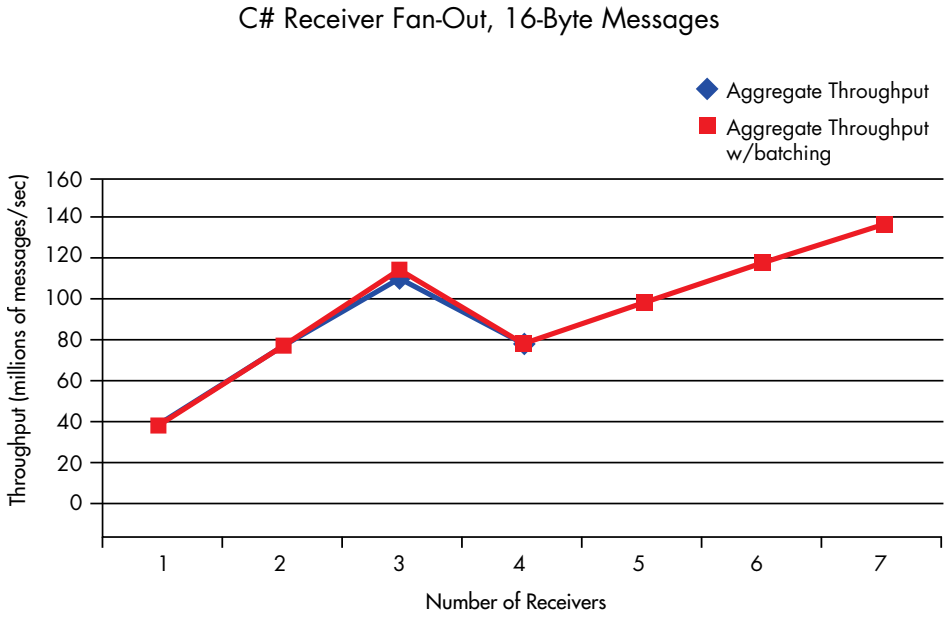

Figure 9. Highest aggregate throughput for C# application receiver fan-out is achieved at 7 subscribing receivers with a rate of 137.3 million messages per second (same results achieved with and without batched message delivery).

Conclusion

Informatica Ultra Messaging software has advanced trading strategies throughout the capital markets industry by significantly improving communication for electronic trading applications. With the release of the Ultra Messaging SMX shared-memory transport and this performance benchmark report, Informatica further strengthens its commitment to providing customers with the highest performing messaging software.

As demonstrated in performance test data, SMX has broken the 100-nanosecond latency barrier on commodity hardware and proven to be the fastest messaging software available in the industry—with latencies as low as 39 nanoseconds. Based on this benchmark, Informatica SMX provides a compelling solution to increase profitability when engaged in trading strategies that require the absolute lowest latency and highest throughput. SMX also increases developer productivity by enabling the use of commercial messaging for both IPC and ITC communication, and it facilitates seamless interoperability among C, Java, and C# applications in shared-memory environments.

ABOUT INFORMATICA ULTRA MESSAGING

Informatica Ultra Messaging (formerly 29West) is the highest performance, lowest latency, and most functional and efficient messaging software available. With more than 200 front-office electronic trading deployments globally, it is the de facto capital markets industry standard for high-performance messaging. Of the top 10 investment banks, 9 rely on it; in addition, 8 of the top 10 FX dealers use it, as do proprietary trading firms, hedge funds, and leading electronic exchanges. After switching to the Ultra Messaging API, customers have reported up to 10x performance improvements in latency reduction and message throughput increase, dramatic reductions in infrastructure costs, and uninterrupted business operations.

ABOUT INFORMATICA

Informatica Corporation (NASDAQ: INFA) is the world's number one independent provider of data integration software. Organizations around the world rely on Informatica for maximizing return on data to drive their top business imperatives. Worldwide, over 4,630 enterprises depend on Informatica to fully leverage their information assets residing on-premise, in the Cloud and across social networks.

Appendix A: Numeric Tables of Test Results

Figure 1. C Average Latency from Core to Hyperthread

Message Size (bytes)	Average Latency (nanoseconds)
16	39
32	39
64	42
128	48
256	63
512	81

Figure 2. C Average Latency from Core to Sibling Core

Message Size (bytes)	Average Latency (nanoseconds)
16	103
32	104
64	104
128	111
256	120
512	135

Figure 3. C Receiver Fan-Out, 16-Byte Messages

Number of Receivers	Aggregate Throughput (no batching) (millions of messages/sec)	Aggregate Throughput (with batching) (millions of messages/sec)
1	42.02	43.80
2	32.86	56.94
3	58.90	85.75
4	53.21	94.90
5	66.93	121.73
6	72.41	136.51
7	87.35	165.83
8	92.29	176.13
9	105.68	216.57
10	107.58	210.90
11	115.24	248.72
12	113.47	256.50
13	122.36	284.57
14	117.61	292.09
15	129.22	288.70
16	132.23	319.86
17	135.11	299.43
18	132.32	307.68
19	133.92	305.34

Figure 4. Java Average Latency from Core to Hyperthread

Message Size (bytes)	Average Latency (nanoseconds)
16	87
32	89
64	91
128	93
256	100
512	106

Figure 5. Java Average Latency from Core to Sibling Core

Message Size (bytes)	Average Latency (nanoseconds)
16	87
32	89
64	91
128	93
256	100
512	106

Figure 6. Java Receiver Fan-Out, 16-Byte Messages

Number of Receivers	Aggregate Throughput (no batching) (millions of messages/sec)	Aggregate Throughput (with batching) (millions of messages/sec)
1	15.50	15.65
2	27.50	31.11
3	46.50	45.78
4	53.17	62.76
5	77.59	75.57
6	76.79	93.05
7	112.18	111.48
8	99.98	125.64
9	140.38	136.61
10	122.30	155.61
11	163.00	167.08
12	135.16	184.31
13	164.71	201.83
14	145.26	213.83
15	195.63	231.43
16	159.90	245.92
17	176.67	263.15
18	164.95	286.10
19	194.69	285.12

Figure 7. C# Average Latency from Core to Hyperthread

Message Size (bytes)	Average Latency (nanoseconds)
16	86
32	88
64	93
128	99
256	110
512	135

Figure 8. C# Average Latency from Core to Sibling Core

Message Size (bytes)	Average Latency (nanoseconds)
16	114
32	119
64	119
128	124
256	145
512	167

Figure 9. C# Receiver Fan-Out, 16-Byte Messages

Number of Receivers	Aggregate Throughput (no batching) (millions of messages/sec)	Aggregate Throughput (with batching) (millions of messages/sec)
1	38.46	38.46
2	76.92	76.92
3	115.38	96.77
4	78.43	78.43
5	98.04	98.04
6	117.65	117.65
7	137.25	137.25

Appendix B: Technology Stack Test Configturations

Hardware and Operating System Configurations

C and Java Latency Tests

The test system consisted of one machine with one socket, four cores, and eight hyperthreads.

CPU/Processor	Intel Xeon E5-1620 @ 3.60GHz
RAM	8 GB
O/S	CentOS 6.4

C and Java Throughput Tests

The test system consisted of one machine with two sockets, each with a 10-core CPU package with hyperthreading (total of 20 threads in each CPU package).

CPU/Processor	Intel Ivy Bridge @ 2.80GHz, 10-core
RAM	32 GB
O/S	RHEL 6.2

C# Latency and Throughput Tests

The test system consisted of one machine with one socket, four cores, and eight hyperthreads.

CPU/Processor	Intel Xeon E5-1620 @ 3.60GHz
RAM	8 GB
O/S	Windows 7 Professional SP1

BIOS Configurations—All Tests and Machines

C3-Report	Enabled
C6-Report	Enabled
Hyperthreading	Enabled
Turbo Mode	Enabled
Intel SpeedStep	Enabled

Software Versions and Configurations

Informatica Ultra Messaging

Version 6.1 of Ultra Messaging Streaming Edition
All LBT-SMX transport-specific configurations set to their default values
UM Test Applications: *lbmlatping/lbmlatpong* (latency); *lbmlatsrc/lbmlatrcv* (throughput)

Java

Version 1.7.0-21
Java SE Runtime Environment (build 1.7.0-21-b11)
Java HotSpot 64-bit Server VM (build 23.21-b01; mixed mode)

Put potential to work.™

Worldwide Headquarters, 100 Cardinal Way, Redwood City, CA 94063, USA Phone: 650.385.5000 Fax: 650.385.5500
Toll-free in the US: 1.800.653.3871 informatica.com linkedin.com/company/informatica twitter.com/InformaticaCorp

IN09_0613_02476