Information Management software

IBM® InfoSphere® Streams

Release 2.0 Performance Report

Roch Archambault
Richard King
Sujay Parekh
IBM Software Group

Contents

Executive Summary

In Release 2.0 of IBM® InfoSphere® Streams, IBM replaced the SPADE language with SPL. Using SPL programs, IBM measured a basic throughput of 440,000 tuples per second or 903 Mbit per second for 256-byte normal tuples and measured a round-trip latency time of 57 microseconds for 256-byte normal tuples. IBM made these measurements using TCP as the transport mechanism over 1 gigabit (Gb) Ethernet connections. For these, and all other measurements, IBM used Red Hat Release 5.5 Linux, 64-bit running on IBM HS-22 blades.

Using IBM® WebSphere® MQ Low Latency Messaging for tuple transport over InfiniBand (IB), IBM measured a basic throughput of 3,500,000 tuples per second or 7.2Gbit per second for 256-byte normal tuples. Also with that transport, IBM measured a round-trip latency time of 28 microseconds for 256-byte normal tuples.

Introduction

This document presents some performance measurements for IBM® InfoSphere® Streams Release 2.0. We will describe the tests carried out and the corresponding results.

IBM obtained all performance data contained in this publication in the specific operating environment and under the conditions described herein. Performance obtained in other environments may vary, so customers should conduct their own testing.

Test Environment

IBM obtained results using a cluster of IBM BladeCenter blades with specific hardware details as follows:

- IBM HS-22 Blades (model 7870-AC1) each with:
  - Two Intel® Xeon® X5570 CPU with:
    - 2.98 GHz clock speed
    - 4-cores
    - Cache: 32KB L1 (per core), 256KB L2 (per core), 8MB L3 (shared)
    - 6.4 GigaTransfer/sec Intel® QPI interconnect
  - 32GB RAM
  - 1Gbit Ethernet NIC on motherboard
  - Infiniband QDR NIC: Mellanox Technologies ConnectX VPI PCIe 2.0 MT25408
    - Up to 32 Gbit/sec effective throughput
    - 1.2 microsecond MPI ping latency
- Infiniband switch module: IBM BCH QDR HSSM, Voltaire 40Gb/s Switch module
- OS: RedHat Enterprise Linux 5.5 64-bit
- Infiniband driver:OFED v1.5.2

Any special software not included in the base RHEL 5.5 OS is as follows:

- netperf v2.4.5 (http://www.netperf.org/netperf/) - for measuring system level TCP performance
- perftestv1.2.4- for measuring system level Infiniband performance (ib_send_bw and related utilities)

The measurements correspond to the following versions of Streams:

- Streams v1.2.1 GA version
- Streams v2.0 GA version

For those throughput tests that use IBM® WebSphere® MQ Low Latency Messaging (LLM) for data transport (compiler option –F llm_rum_tcp or –F llm_rum_ib), the LLM configuration specifies that MinBatchingMicro = 400 and MaxBatchingMicro = 800. For latency tests, the configuration sets MinBatchingMicro = 25 and MaxBatchingMicro = 50.

Throughput Test

Test Description

IBM measures the maximum throughput achievable by Streams using a 2-operator SPL program: a source operator produces a stream that is consumed by a sink operator. The different variations (1 node, 2 nodes, all in a single thread, InfiniBand networking, etc.) are achieved with configuration options in the SPL (e.g. host placement constraints) and with SPL compiler options (e.g.-F llm_rum_ib). The sink operator performs the measurements; it records the arrival times of the first and last tuples and the number of tuples, together yielding the throughput.

The "Intranode" configuration we often refer to is one where the source and sink operators are placed on the same host; all communication is therefore within that one host. Conversely, the "Inter-node" configuration uses 2 hosts, one for the source, the other for the sink.

Intranode Configuration

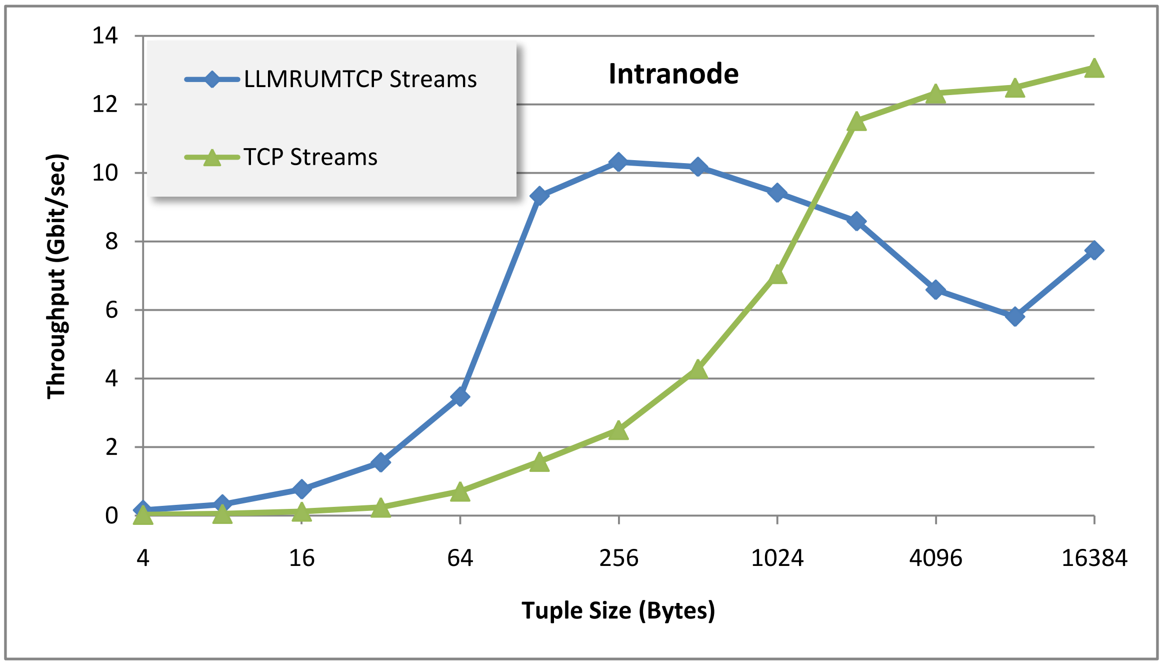

Figure 1 Comparison of intranode throughput with TCP based transports

In an intranode configuration, we compare the Streams Source-Sink application against similar sender-receiver applications that use the corresponding native APIs. From Figure 1 and Table 1 (columns 2, 3 and 4), we see that up to tuple sizes of 2KB, the default TCP transport only suffers small penalties relative to direct TCP access. Beyond 2KB, the default TCP transport encounters some bottleneck and the throughput flattens out, while direct TCP access continues to scale its throughput up to 8KB tuples. The nature of this bottleneck is unknown and is to be investigated further.

tuple size (bytes)	llmrumtcp	tcp streams	LLMRUMIB Streams
4	0.16	0.03	0.17
8	0.33	0.06	0.29

16	0.77	0.12	0.74
32	1.55	0.24	1.46
64	3.46	0.71	2.02
128	9.33	1.58	5.48
256	10.32	2.51	8.14
512	10.18	4.29	12.61
1024	9.42	7.06	13.39
2048	8.59	11.52	16.03
4096	6.59	12.32	15.80
8192	5.80	12.49	16.55
16384	7.74	13.07	17.39

Table 1 Throughput (Gbits/sec)

Interestingly, using TCP via the LLM transport (LLM_RUM_TCP) provides increased throughput even over raw TCP, at least up to 1KB tuples. This is due to some amount of batching that occurs in LLM, resulting in fewer system calls being made. For larger tuples, however, the LLM performance is worse than using the native TCP transport. The reason for this performance degradation is unknown and will be the subject of further study.

With InfiniBand, the batching introduced by LLM offers significant benefits relative to direct transfer

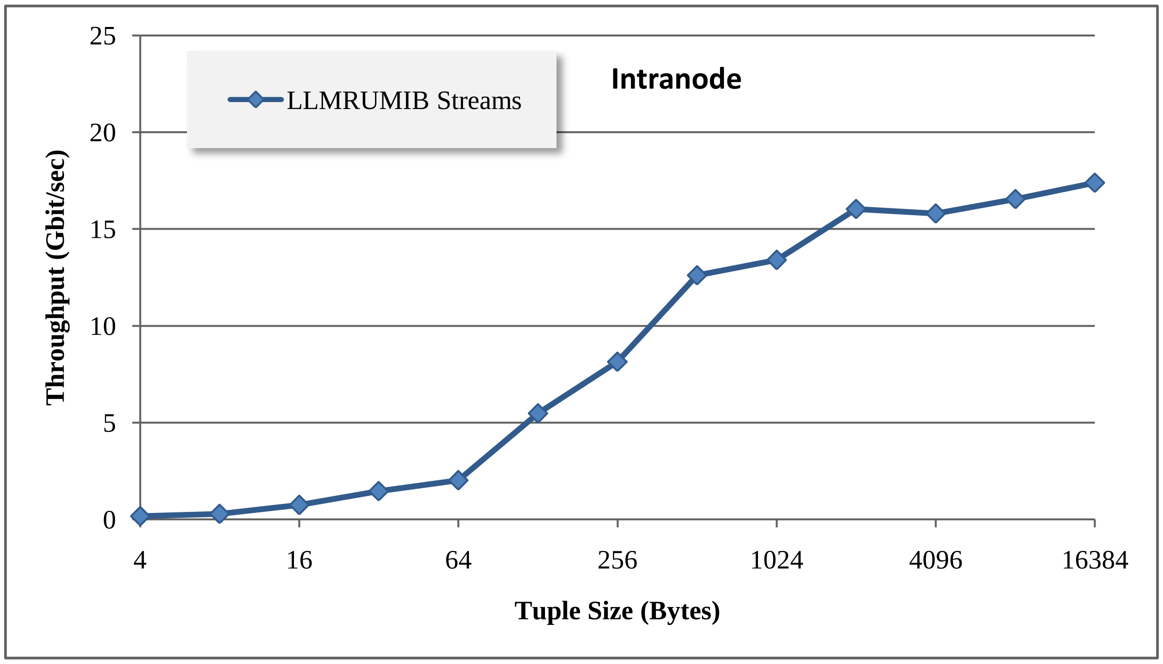

Figure 2 Comparison of intranode throughput with Infiniband transport

using native InfiniBand primitives. As shown in Figure 2 and Table 1 (columns 5 and 6), the throughput can be as much as 3 times more using LLM (for tuple size of 1K

Inter-node Configuration

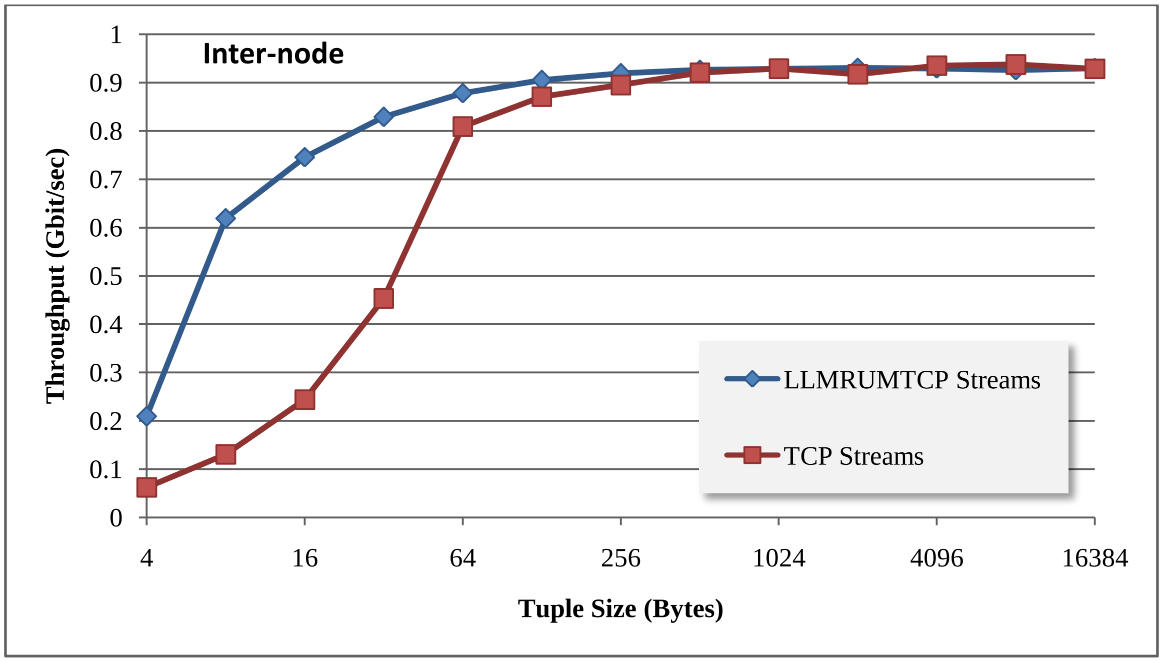

Figure 3 Comparison of inter-node throughput with TCP transport

tuple size	llmrumtcp	tcp streams	llmrumib
4	0.21	0.06	0.13
8	0.62	0.13	0.31
16	0.75	0.24	0.61
32	0.83	0.45	1.21
64	0.88	0.81	2.44
128	0.90	0.87	5.17
256	0.92	0.89	7.79
512	0.93	0.92	13.75
1024	0.93	0.93	12.96
2048	0.93	0.92	19.20
4096	0.93	0.94	19.04
8192	0.93	0.94	17.11
16384	0.93	0.93	20.42

Table 2 Internode Throughput (Gbits/sec)

We now consider the same application as before, but the sender and receiver are placed on different hosts (see Figure 3 and Table 2 columns 2, 3 and 4). In this case, the inter-node TCP throughput is limited by the 1Gbit/sec capacity of the Ethernet. Once again, we see that using the default TCP transport incurs a modest overhead for small tuples (up to 256B) and almost no overhead for larger tuples. Using TCP via the LLM option provides the best of both worlds: much higher throughput for small tuples (up to 32 bytes) and in the limit it is identical to the native TCP transport.

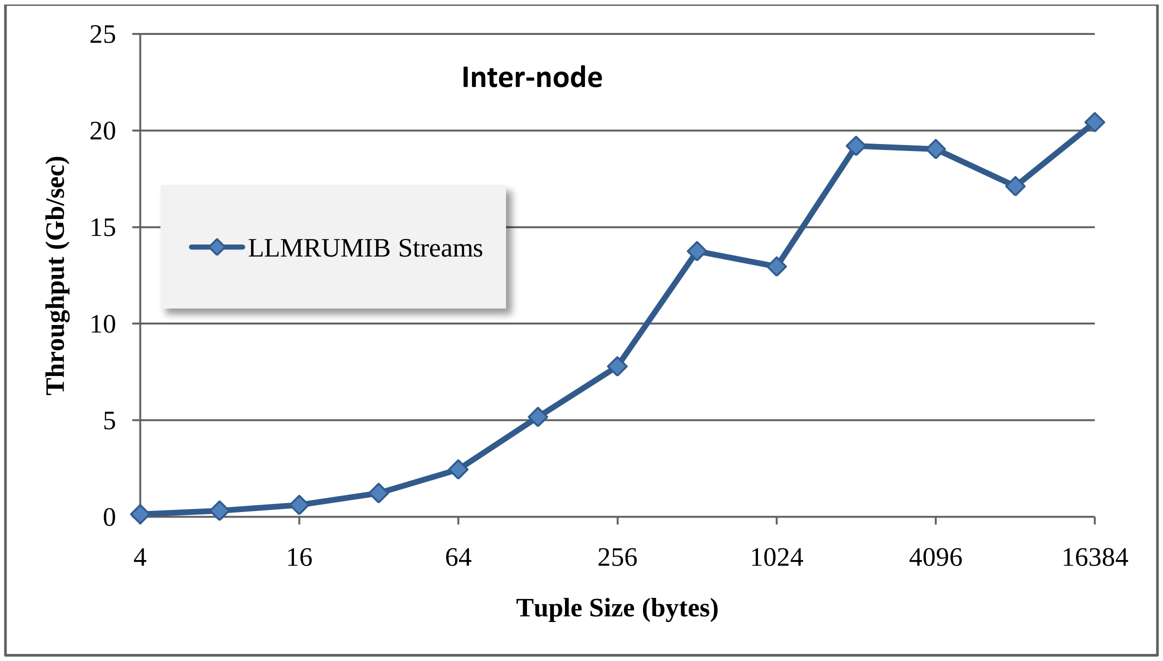

Figure 4 Comparison of inter-node throughput with Infiniband transport

Infiniband in the inter-node case has a much higher theoretical capacity relative to TCP: 32 Gbit/sec effective throughput. In Figure 4 and Table 2 (columns 5 and 6), we see that the throughput achieved with a single sender-receiver pair seems to reach a limit around 20-22 Gbit/sec with larger tuples. Similar to LLM with TCP, using LLM with IB helps achieve good throughput with smaller tuples (up to 1KB).

Benefit of Façade Tuples

Using the "-k" or "—prefer-façade-tuples" option of the SPL compiler allows the generated code to to treat tuples as fixed-size data buffers that don't require serialization and deserialization. With Intranode TCP, façade tuples provide a slight advantage, as seen in Figure 5 and Table 3 columns 2 through 5. (Note: we attribute the areas where Normal exceeds Façade to measurement noise.)

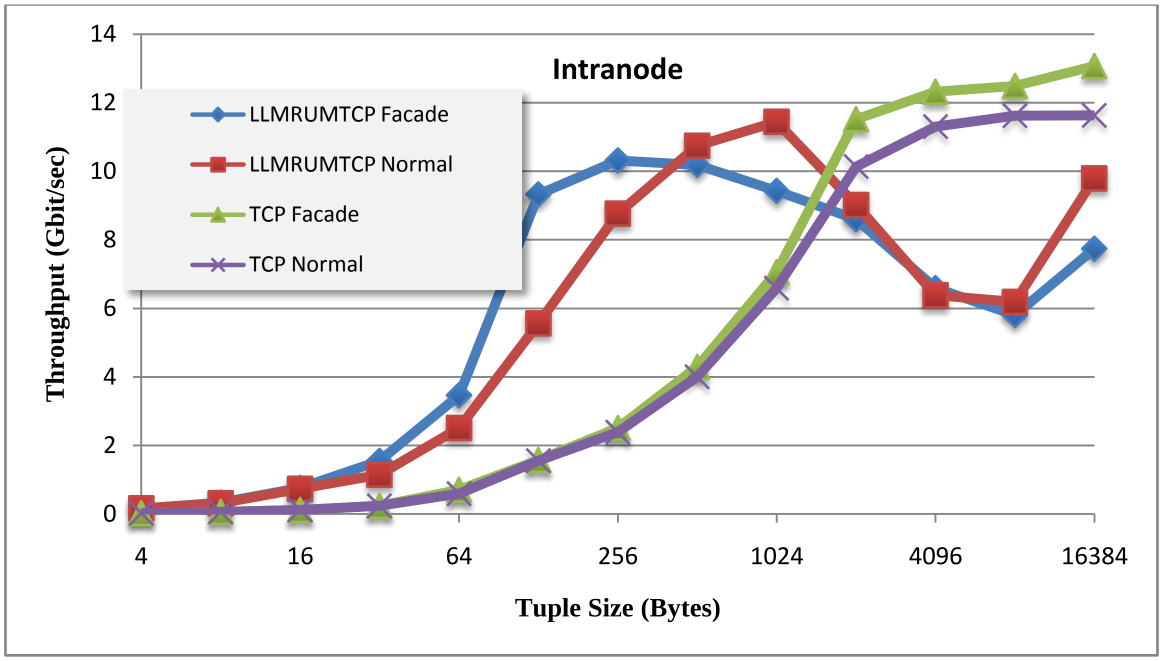

Figure 5 Comparison of Facade Tuples with Regular Tuples

Intranode communication using Infiniband also benefits from using façade tuples, as we see in Figure 6 and Table 3 columns 6 and 7. Larger tuples do better with façade tuples because of the elimination of lengthy copying during serialization and deserialization.

tuple size (bytes)	LLMRUMTCP Façade	LLMRUMTCP Normal	TCP Façade	TCP Normal	LLMRUMIB Façade	LLMRUMIB Normal
4	0.16	0.16	0.03	0.03	0.17	0.17
8	0.33	0.31	0.06	0.06	0.29	0.32
16	0.77	0.74	0.12	0.12	0.74	0.54
32	1.55	1.15	0.24	0.24	1.46	1.18
64	3.46	2.51	0.71	0.59	2.02	2.53
128	9.33	5.56	1.58	1.55	5.48	3.61
256	10.32	8.77	2.51	2.39	8.14	7.39
512	10.18	10.74	4.29	4.01	12.61	10.99
1024	9.42	11.44	7.06	6.59	13.39	11.88
2048	8.59	9.01	11.52	10.14	16.03	12.91
4096	6.59	6.38	12.32	11.30	15.80	10.91
8192	5.80	6.19	12.49	11.62	16.55	14.05
16384	7.74	9.79	13.07	11.63	17.39	17.12

Table 3 Facade vs Normal Throughput (Gbits/sec)

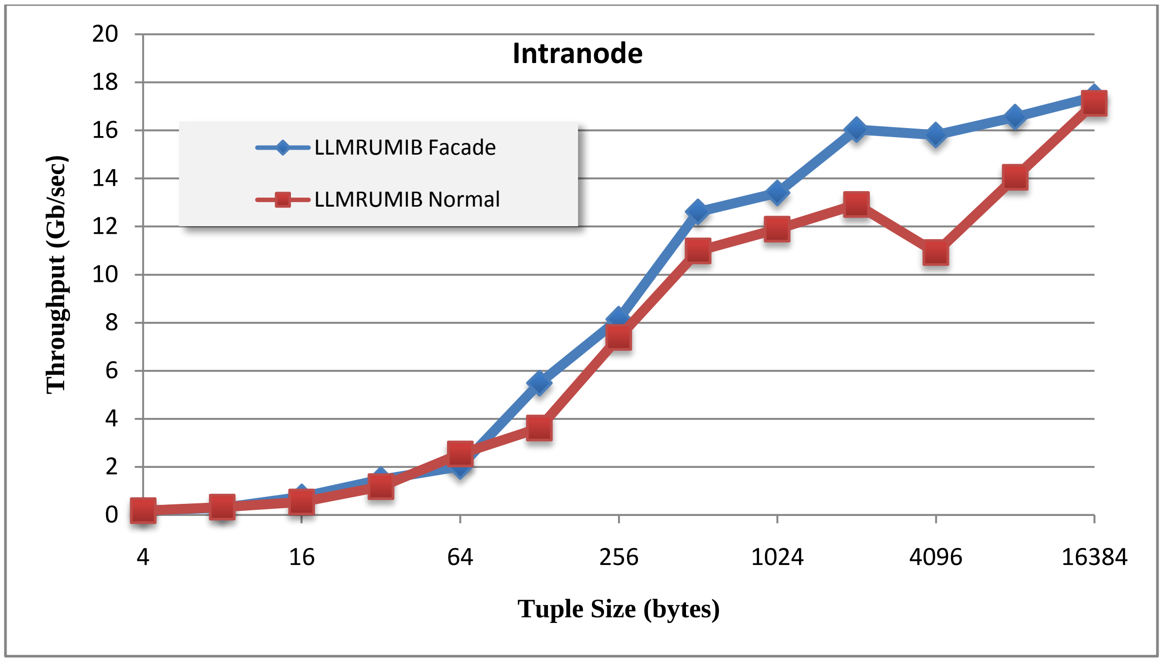

Figure 6 Intranode throughput using IB

For Inter-node communication using 1GbE (via either plain TCP transport or LLMRUMTCP), façade tuples provide no significant benefit, as we see in Figure 7 and Table 4 columns 2 through 5.

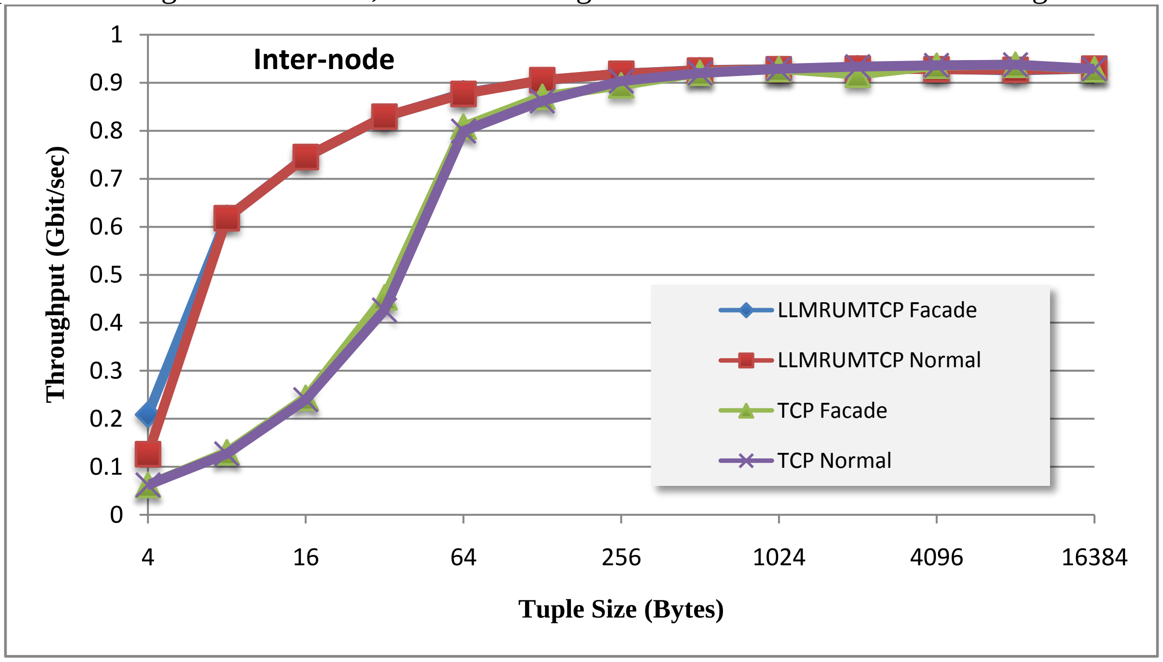

Figure 7 Comparison of plain TCP throughput with LLM-TCP

tuple size (bytes)	LLMRUMTCP Façade	LLMRUMTCP Normal	TCP Façade	TCP Normal	LLMRUMIB Façade	LLMRUMIB Normal
4	0.21	0.13	0.06	0.06	0.13	0.18
8	0.62	0.62	0.13	0.13	0.31	0.27

16	0.75	0.75	0.24	0.24	0.61	0.54
32	0.83	0.83	0.45	0.43	1.21	1.08
64	0.88	0.88	0.81	0.80	2.44	2.21
128	0.90	0.91	0.87	0.86	5.17	3.65
256	0.92	0.92	0.89	0.90	7.79	7.18
512	0.93	0.93	0.92	0.92	13.75	11.82
1024	0.93	0.93	0.93	0.93	12.96	12.13
2048	0.93	0.93	0.92	0.93	19.20	13.61
4096	0.93	0.93	0.94	0.94	19.04	11.39
8192	0.93	0.93	0.94	0.94	17.11	16.95
16384	0.93	0.93	0.93	0.93	20.42	19.73

Table 4 Internode Facade vs Normal (Gbits/sec)

The higher throughputs of Inter-node communication using Infiniband are sufficient to significantly load the host processors (especially on the sending host). Thus, with a processing bottleneck limiting the throughput, the savings provided by façade tuples once again becomes apparent. Figure 8 and Table 4 (columns 6 and 7) once again shows us an advantage with façade tuples when the tuples are larger.

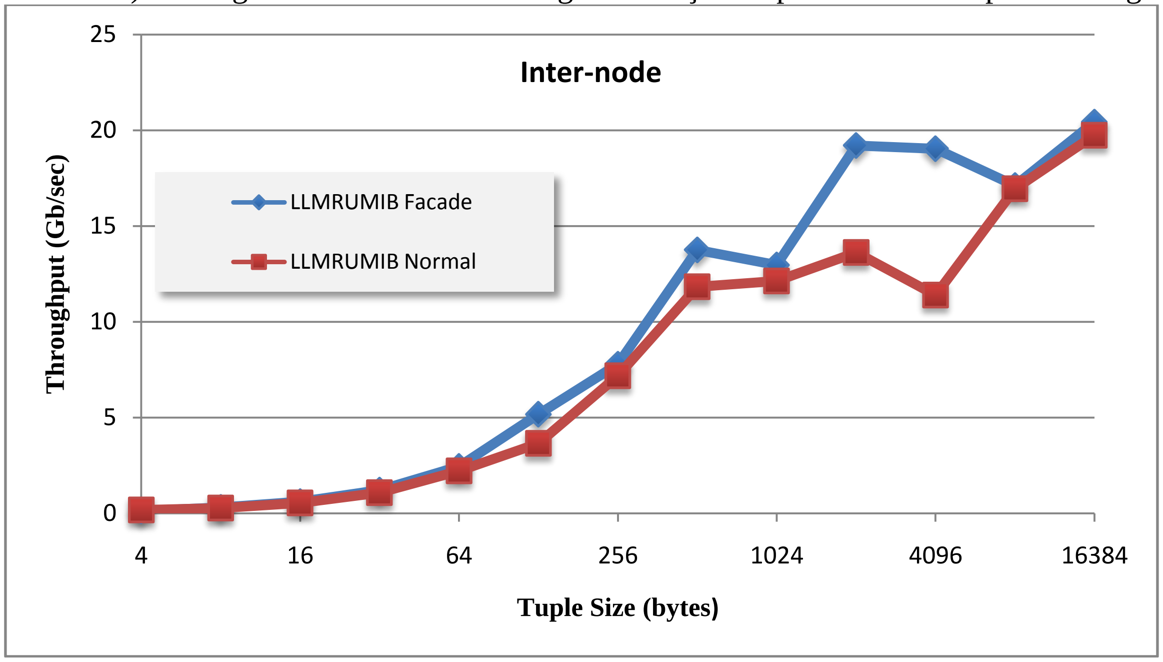

Figure 8 Inter-node throughput of InfiniBand

Overhead of Threaded Ports

Fusing the source and sink operators into a single thread yields the lowest possible communication costs (a function call). When the application would benefit from use of another core, the sink can put a threaded input port (a new feature in Streams v2.0) in its configuration. This provides it with a separate thread for its processing, at the cost of having the source enqueue a tuple in the sending thread, and having the sink dequeue it in the receiving thread. See, for example, https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/W4671426578cc_49bd_b585_3b9911c491b8/page/Streams%20Performance%20Topics for information on performance bottlenecks; the techniques discussed there can help one decide if the use of threaded ports would be helpful in a particular application.

One could, instead, get use of another core by communicating intranode using LLM over InfiniBand. Note, however, that this would require InfiniBand hardware on the node, even if it is only intranode communication that requires this speed, plus tuning of LLM to get the best performance out of it. See, for example, http://www.ibm.com/developerworks/data/library/techarticle/dm-1006performancetuning/index.html for suggestions on how to go about tuning LLM.

Figure 9 and Table 5 show the overhead of using threaded ports for pipeline parallelism. The overhead of threaded ports is slightly greater than the overhead of using LLM over InfiniBand between PEs for tuples of 512 bytes and smaller. However, for larger tuples, threaded ports have the advantage over LLM over InfiniBand. With threaded ports, the communicating operators run in the same address space. No data copies occur, only the enqueueing and dequeueing of pointers, giving an advantage over LLM, which must copy data into and out of communication buffers to pass data between the processes that contain the PEs. This leads to as much as 20 times the throughput via threaded ports as that achieved between PEs. Given this and the other considerations already mentioned, we strongly recommend the use of threaded ports to gain parallelism between operators.

Another way of comparing these transports is to say that fused operators can pass 20 million tuples per second, while operators separated by a threaded port can pass 2.5 million tuples per second. LLM over InfiniBand also runs at about 2.5 million tuples per second as long as the tuples are 512 bytes or smaller, but is limited in bandwidth for larger tuples, while fused operators and those using threaded ports are not. Given that our processors run at 2.98GHz, we can figure that in return for adding about 170 processor cycles to enqueue and dequeue the tuple, we can pipeline the processing of it across 2 different cores. For a string of operators that are fused together (for example by COLA, as discussed in the next section), or for a large custom operator that can be rewritten as a pipeline of operators, this can relieve a bottleneck that limits the performance of an entire application.

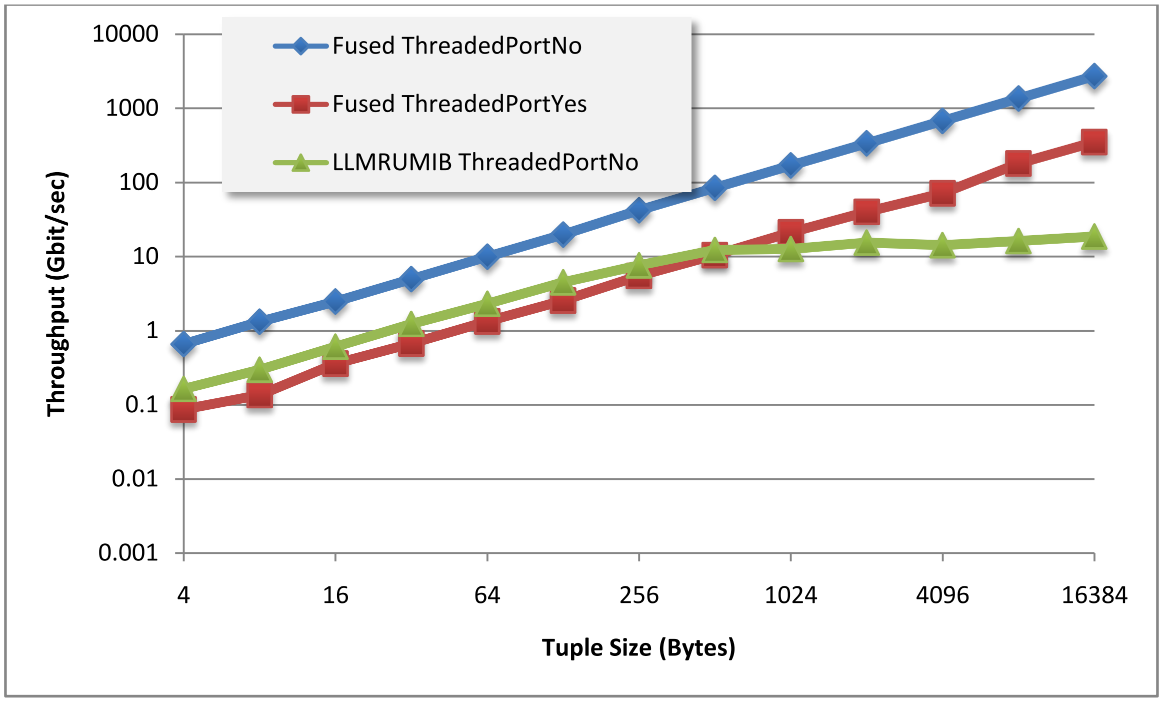

Figure 9 Comparison of threaded port throughput with Infiniband used intranode

tuple size (bytes)	Fused	Threaded Port	LLMRUMIB
4	0.66	0.09	0.16
8	1.32	0.14	0.30
16	2.47	0.36	0.61
32	4.94	0.67	1.23
64	9.91	1.33	2.30
128	19.81	2.54	4.48
256	42.08	5.47	7.62
512	84.42	10.46	12.29
1024	169.49	21.19	12.59
2048	337.85	39.95	15.44
4096	677.63	71.21	14.28
8192	1355.20	179.81	16.16
16384	2711.61	349.17	18.67

Table 5 Threaded Port Throughput (Gbits/sec)

Optimized Fusion using COLA

COLA is the new fusion optimizer for the SPL compiler. Based on statistics gathered on previous runs of an application (compiler options –P 10 –S 0.1, for example), COLA computes which operators to fuse together into a single thread (compiler option –p FOPT). Its goal is to maximize the application throughput given the available resources. To see how good a job COLA does at this, we ran an application based on the VWAP calculation (see http://www.sciencedirect.com/science/article/pii/S074373151000167X#s000045 for details) that we configured the application to include a total of 332 operators. We measured the throughput of the application when it was run 4 different ways: unfused (each operator running as a separate PE using only TCP transport), cola 1 (as fused by COLA based on the stats collected during the unfused run), cola 2 (as fused by COLA based on the stats collected during the cola 1 run), and manual (fusion specified by the programmer based on some trial & error). We also tried these variations on different numbers of hosts.

The results, seen in Figure 10, indicate that COLA doesn't always do a better job than humans, but that it often does. It's especially interesting to note that the second round of COLA optimization further improves performance over the first round. COLA's calculations are based on an approximate model of both the communication costs between operators and the topology of the system (since it is oblivious of the number of cores on each of the hosts), and these approximations limit its accuracy. When the result of the first round of COLA optimization actually runs, the newer stats give COLA a chance to apply itself to a more accurate measure of the consequences of its fusion choices. That second round is seems worthwhile in our tests and we believe it to be so in general. Performing more iterations might possibly be advantageous, but the results in http://www.springerlink.com/content/aw817m13m4536001/fulltext.pdf indicate that additional iterations provide no significant further improvement.

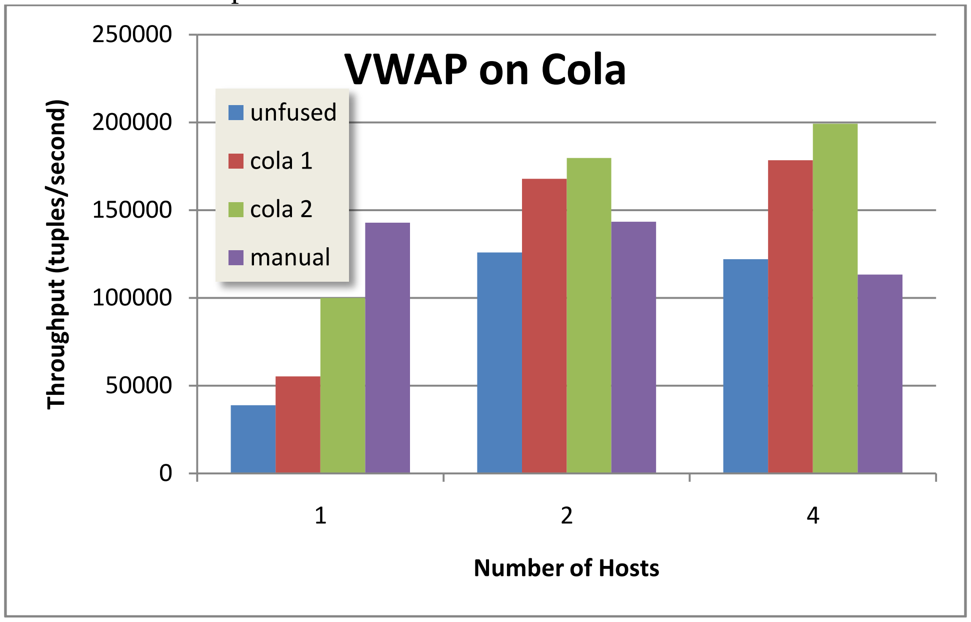

Figure 10 VWAP Throughput as Fusion Varies

Fan-out within a Single Host

It turns out that there is another reason to use LLM over TCP instead of the plain TCP transport. When an operator sends to multiple operators, all of which are on the same host, including the sender, the throughput drops. Note that we observe this only when communication is intranode, not when communication is between different nodes.

That is, with plain TCP as the transport between the operators, we get the result seen in Figure 11 and Table 6. For small (22-byte to 108-byte) tuples, the throughput drops dramatically (by about a factor of 10) when going from 1 to 2 receivers, and staying at that low rate as more receivers are added. Our own investigations, as well as those by Linux experts elsewhere in IBM, indicate that this is a reaction of

TCP to the fast arrival of acknowledgements to small packets when used intra-node. As a result of these quick acknowledgements, TCP never bothers to batch the packets. This (somehow) interacts with the time spent cycling between the different receivers. Various attempts were made to work around this problem, including fiddling with TCP parameters and user-level batching. The only general-purpose solution is to switch to using LLM over TCP as the transport. As we can see, again in Figure 11, the throughput when using LLM over TCP is always better than plain TCP transport, and does not suffer from the fan-out problems of plain TCP. We attribute this to the batching being performed by LLM, which keeps TCP from seeing small packets in the first place.

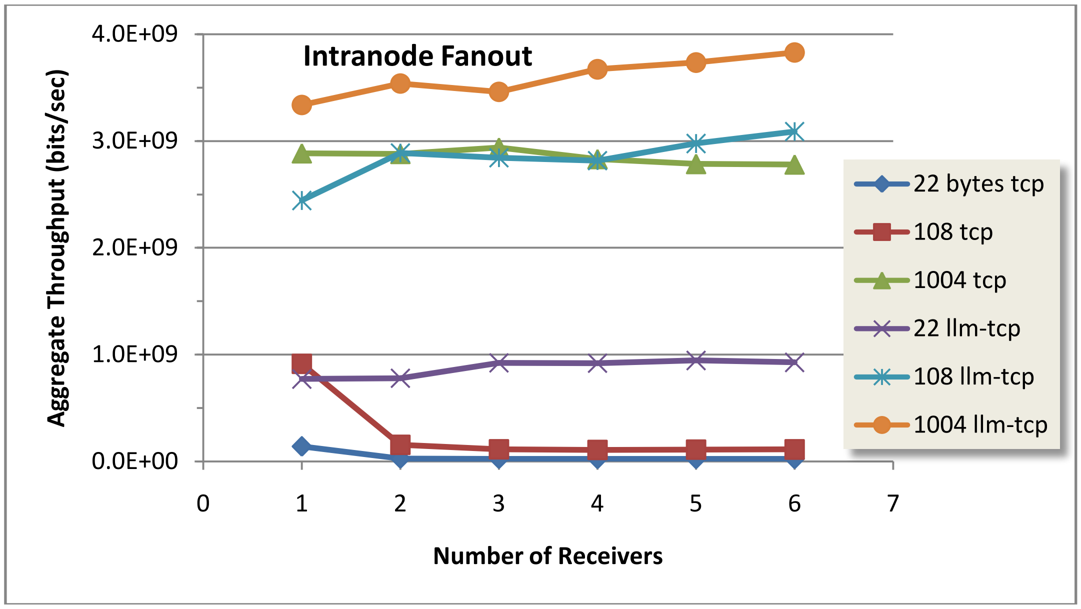

Figure 11 Intra-node Fan-out with Different Tuple Sizes and Transports

Receivers	22 bytes TCP	108 bytes TCP	1004 bytes TCP	22 bytes LLM-TCP	108 bytes LLM-TCP	1004 bytes LLM-TCP
1	0.14	0.91	2.89	0.77	2.44	3.34
2	0.03	0.15	2.88	0.78	2.89	3.54
3	0.02	0.11	2.94	0.92	2.84	3.46
4	0.02	0.11	2.83	0.92	2.82	3.67
5	0.02	0.11	2.78	0.94	2.98	3.73
6	0.02	0.11	2.78	0.93	3.09	3.83

Table 6 Fan-out Throughput (Gbits/sec)

Comparison to Previous Release

To verify that Streams changes between versions 1.2 and 2.0 were improvements to its performance, we made a variety of comparison runs. In Figure 12 and Table 7 columns 2 through 5 we compare the v1.2 and v2.0 performance within a single node using TCP (and LLM over TCP) as the transport between source and sink. The plain TCP results show no significant change, but LLM is significantly different. Since Streams changed from LLM version 2.3 to version 2.5, this is not a surprise. Happily, when using

TCP, LLM version 2.5 seems to perform better, overall, than version 2.3. There's a crossover for tuples around 1KB, but that may well be measurement noise.

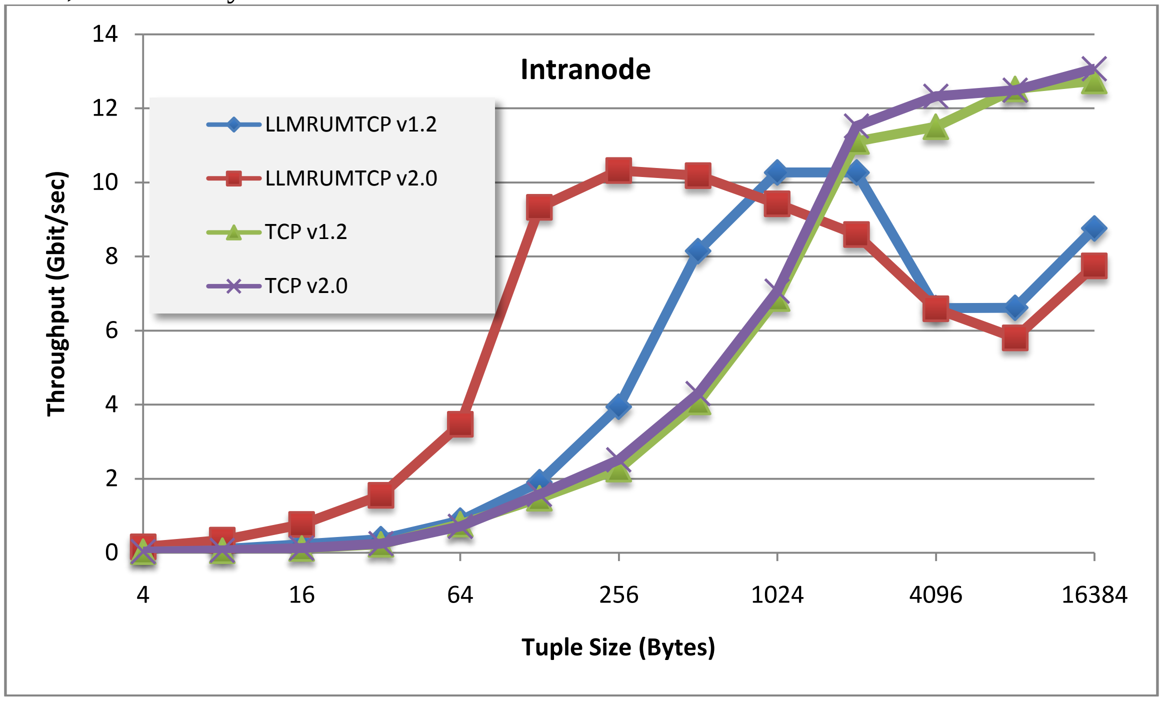

Figure 12 Comparison between Streams v1.2 and v2.0 using TCP intranode

tuple size (bytes)	LLMRUMTCP v1.2	LLMRUMTCP v2.0	TCP v1.2	TCP v2.0	LLMRUMIB v1.2	LLMRUMIB v2.0
4	0.05	0.16	0.03	0.03	0.04	0.17
8	0.09	0.33	0.06	0.06	0.09	0.29
16	0.23	0.77	0.11	0.12	0.27	0.74
32	0.36	1.55	0.23	0.24	0.44	1.46
64	0.86	3.46	0.77	0.71	0.68	2.02
128	1.90	9.33	1.47	1.58	1.58	5.48
256	3.94	10.32	2.26	2.51	2.56	8.14
512	8.14	10.18	4.07	4.29	5.56	12.61
1024	10.26	9.42	6.87	7.06	8.01	13.39
2048	10.27	8.59	11.11	11.52	10.44	16.03
4096	6.61	6.59	11.50	12.32	13.48	15.80
8192	6.61	5.80	12.52	12.49	16.73	16.55
16384	8.76	7.74	12.74	13.07	17.65	17.39

Table 7 Intranode Throughput Comparison (Gbits/sec)

In Figure 13 and Table 7 columns 6 and 7 we have the results when still staying with a single node, but switching to InfiniBand transport. The advantage of Streams version 2.0 over version 1.2 is clearer here, over an even wider range of tuple sizes. Since we used the same LLM configurations, we attribute this to the use of LLM 2.5 instead of LLM 2.3.

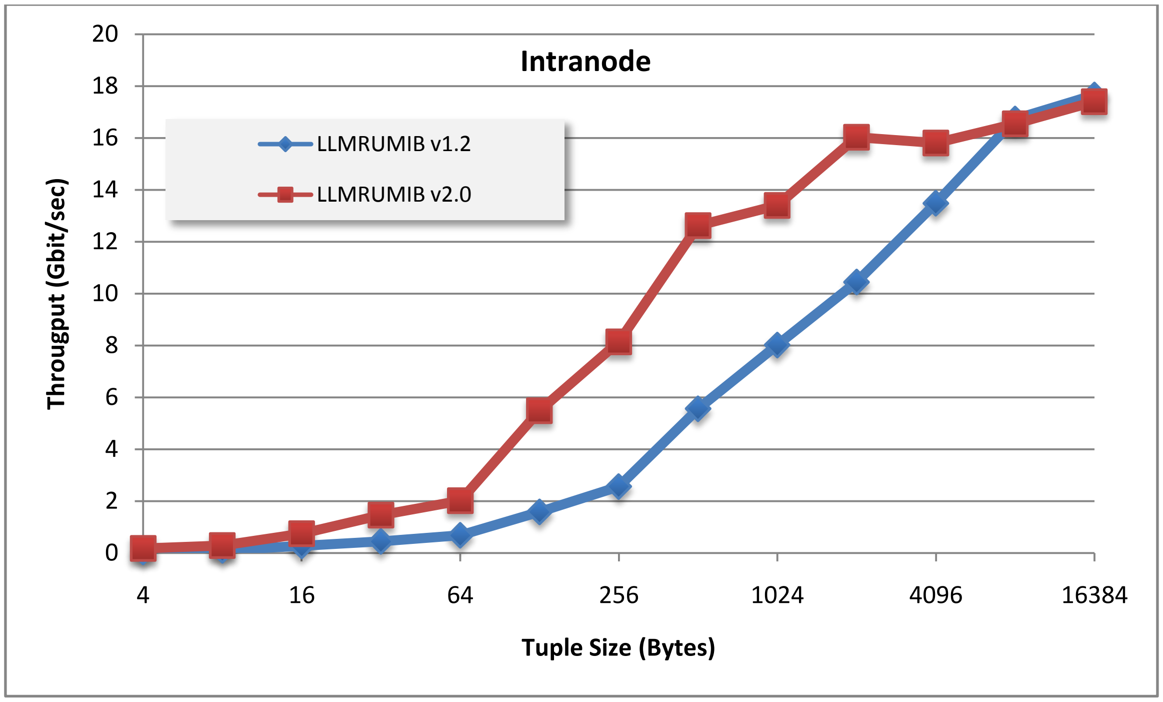

Figure 13 Comparison between Streams v1.2 and v2.0 using InfiniBand intranode

Inter-node throughput comparisons using TCP can be seen in Figure 14 and Table 8 columns 2 through 5. The only significant difference between Streams v1.2 and v2.0 is for LLM over TCP with 4-byte and 8-byte tuples.

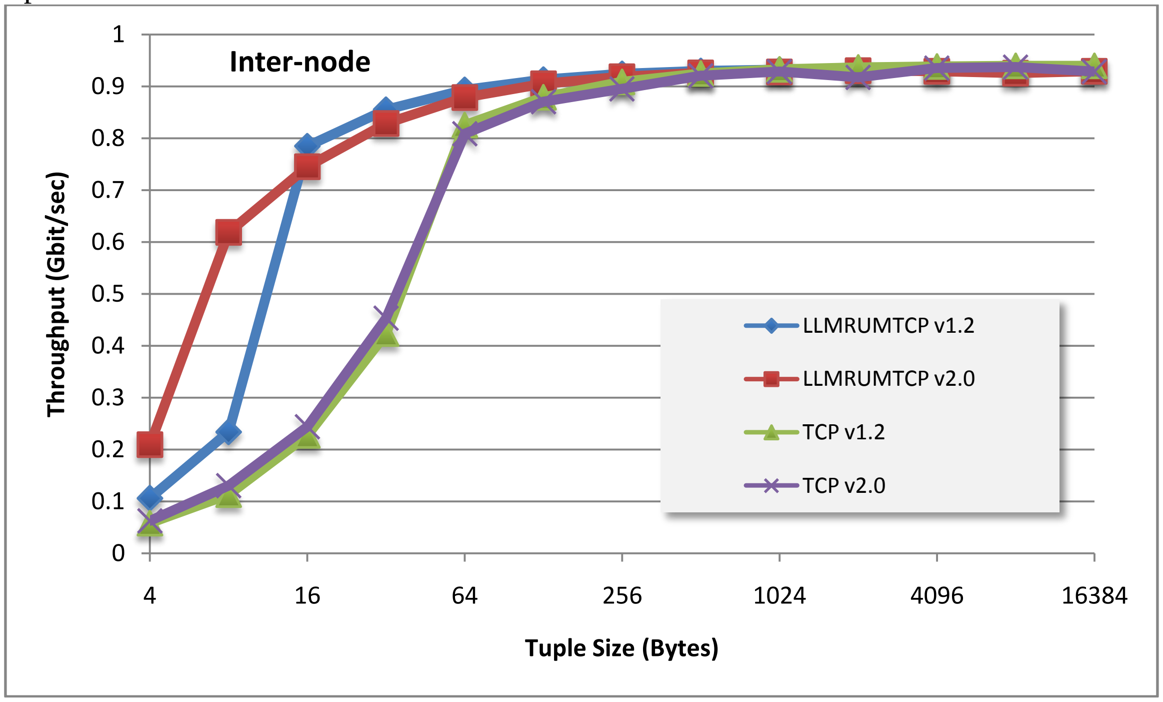

Figure 14 Comparison of Streams v1.2 with v2.0 using TCP between 2 nodes

tuple size (bytes)	LLMRUMTCP v1.2	LLMRUMTCP v2.0	TCP v1.2	TCP v2.0	LLMRUMIB v1.2	LLMRUMIB v2.0
4	0.11	0.21	0.06	0.06	0.13	0.13
8	0.23	0.62	0.11	0.13	0.21	0.31
16	0.78	0.75	0.23	0.24	0.45	0.61
32	0.86	0.83	0.42	0.45	1.09	1.21
64	0.89	0.88	0.83	0.81	2.19	2.44
128	0.91	0.90	0.88	0.87	3.58	5.17
256	0.92	0.92	0.91	0.89	6.36	7.79
512	0.93	0.93	0.92	0.92	12.08	13.75
1024	0.93	0.93	0.93	0.93	11.34	12.96
2048	0.93	0.93	0.94	0.92	16.02	19.20
4096	0.93	0.93	0.94	0.94	21.35	19.04
8192	0.93	0.93	0.94	0.94	21.69	17.11
16384	0.93	0.93	0.94	0.93	22.54	20.42

Table 8 Inter-node Throughput Comparison (Gbits/sec)

The inter-node comparison using InfiniBand in Figure 15 and Table 8 columns 6 and 7 gives the overall advantage to Streams v2.0 over v1.2. The lead is held by v2.0 up to 4KB tuples, where v1.2 takes a bit of a lead. The stumble by v2.0 at 8KB is the largest of the problems. Again, this may be noise, although repeated runs always seem to show some problem at 8KB. Since within Streams itself the handling of the tuples is uniform across the entire range of tuple sizes, this is, if anything, an LLM issue. As of yet, this is unconfirmed.

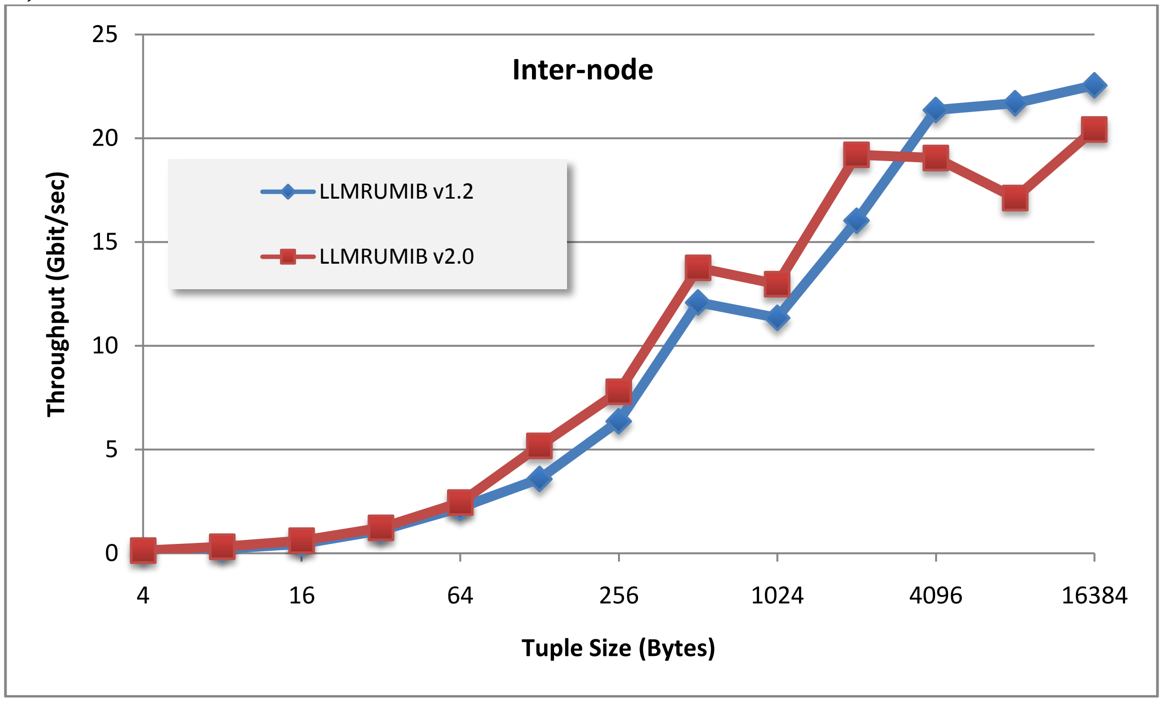

Figure 15 Comparison of Streams v1.2 with v2.0 using InfiniBand between 2 nodes

Latency Test

Test Description

Clock skew is always an issue with latency measurements. We control that by always doing round-trip measurements: from a source operator, to one that receives tuples and passes them along, to the final sink. The source records the current time in each tuple, and the sink compares that with the arrival time there at the sink to compute the round-trip latency. When measuring the latency between 2 different nodes, clock skew is avoided by always putting the source and sink on the same node, with the relay operator being on the other. That way, the timestamp recorded by the source is compared to a timestamp read by the sink from the very same clock. We also avoid use of the high-performance cycle counter as a clock since these can be different on different cores within a single node. This avoids all of the problems that arise when trying to keep the clocks on 2 different nodes in sync.

Various queueing and batching effects can come into play as the workload varies, and these can have a significant impact on latency. To see if that's the case, we measure latency while having the source offer tuples at both a rate of 1,000 tuples per second and also, separately, at a rate of 5,000 tuples per second.

Latency Results

When the source, relay, and sink are all on the same node, there's little to distinguish the latencies seen, regardless of the Streams transport chosen, as we see in Figure 16. LLM works off of timers, so the lack of variation among the different flavors and flow rates for LLM is unsurprising; it's just doing what it does based on the configuration that we specify, and that's what we get. It's difficult to say what's required for TCP to have reduced latency at higher throughputs. It might have something to do with timers in the kernel.

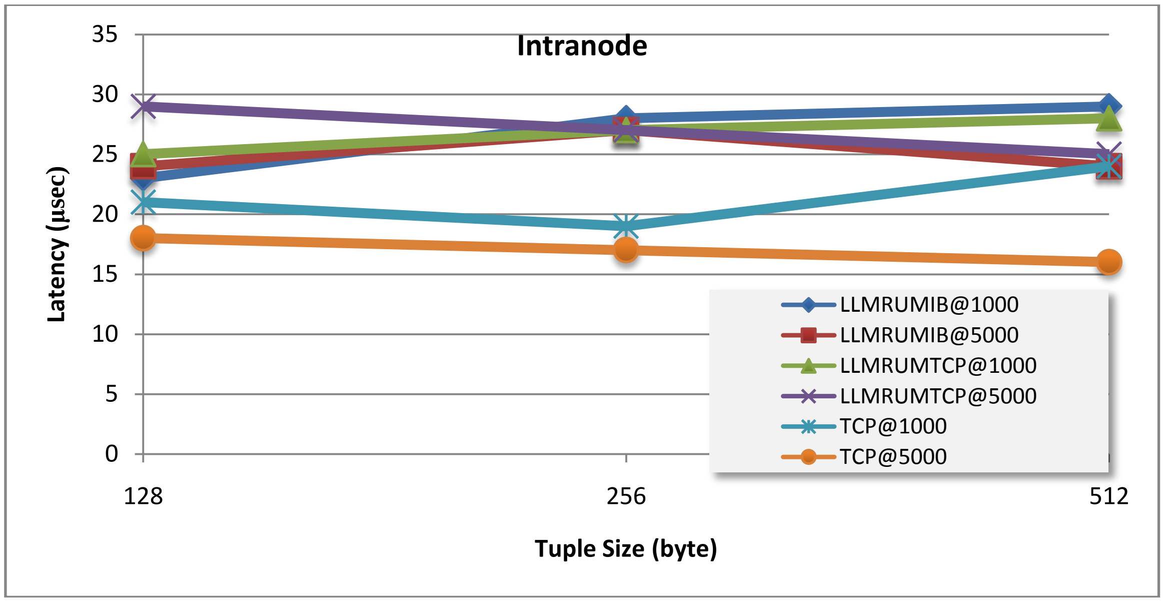

Figure 16 Latency within a single node using different transports

When we look at the inter-node results (where the source and sink are on one node, while the relay is on another) shown in Figure 17, we see that LLM over InfiniBand has much lower latencies. Unlike the intranode measurements, here we have a clear upward trend as the tuple size grows. The time on the wire really does change when sending more data.

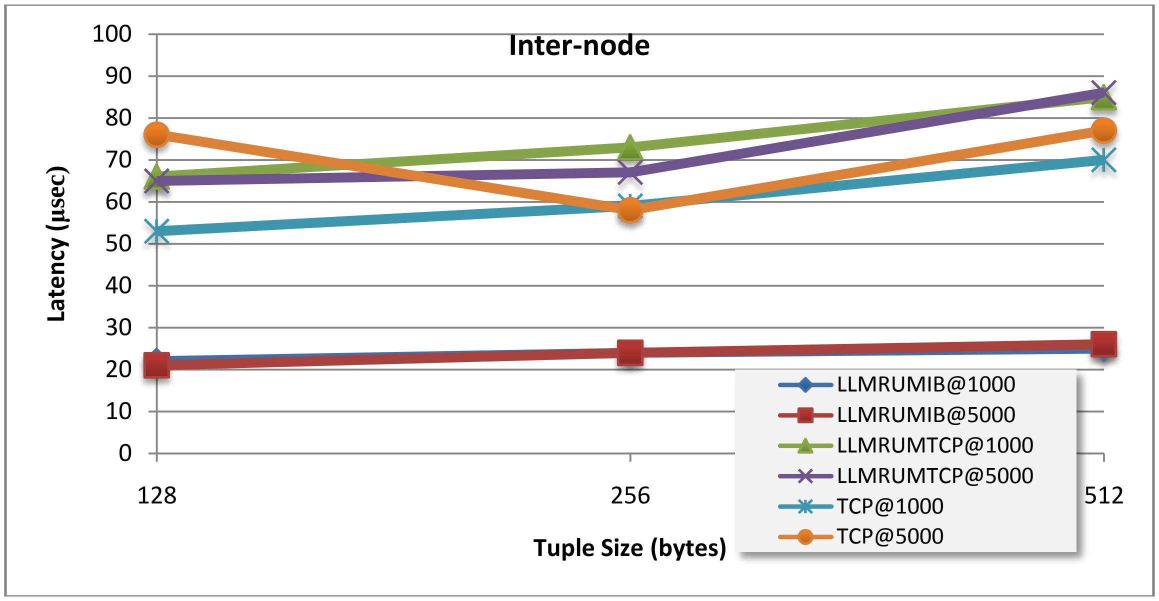

Figure 17 Latency between 2 nodes using different transports

Comparison to Previous Release

Intranode latency with plain TCP and with LLM over IB was unchanged from Streams v1.2 to v2.0, as we can see in Figure 18. The one difference is an improvement (reduction, that is) in latency with LLM over TCP. Intranode throughput numbers for these tuple sizes also showed a big improvement, and it's quite possible that these are related to each other, and to the change in LLM version that we speculated about earlier.

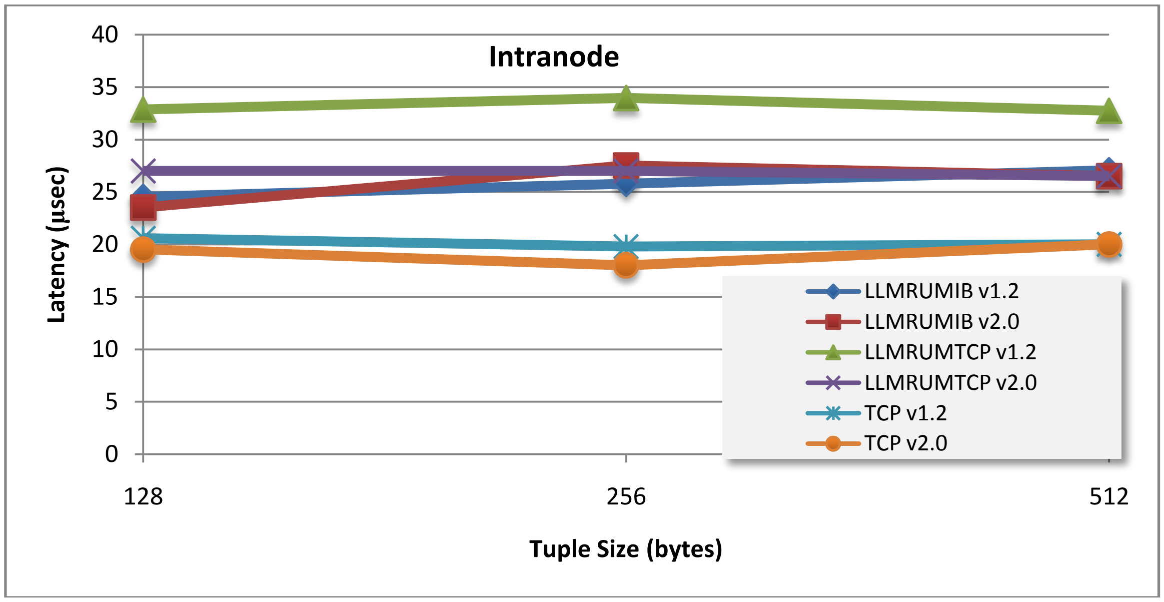

Figure 18 Comparison of Streams v1.2 and v2.0 latencies within a single node

Figure 19 shows the result of measuring inter-node latency at 5,000 tuples per second. Latencies for LLM over InfiniBand latencies were unchanged. However, in repeated runs, both plain TCP and LLM over TCP improved significantly with Streams v2.0. It is our conjecture that this is due to differences in how Streams v1.2 and Streams v2.0 interact with the Nehalem processors in the HS22 blades.

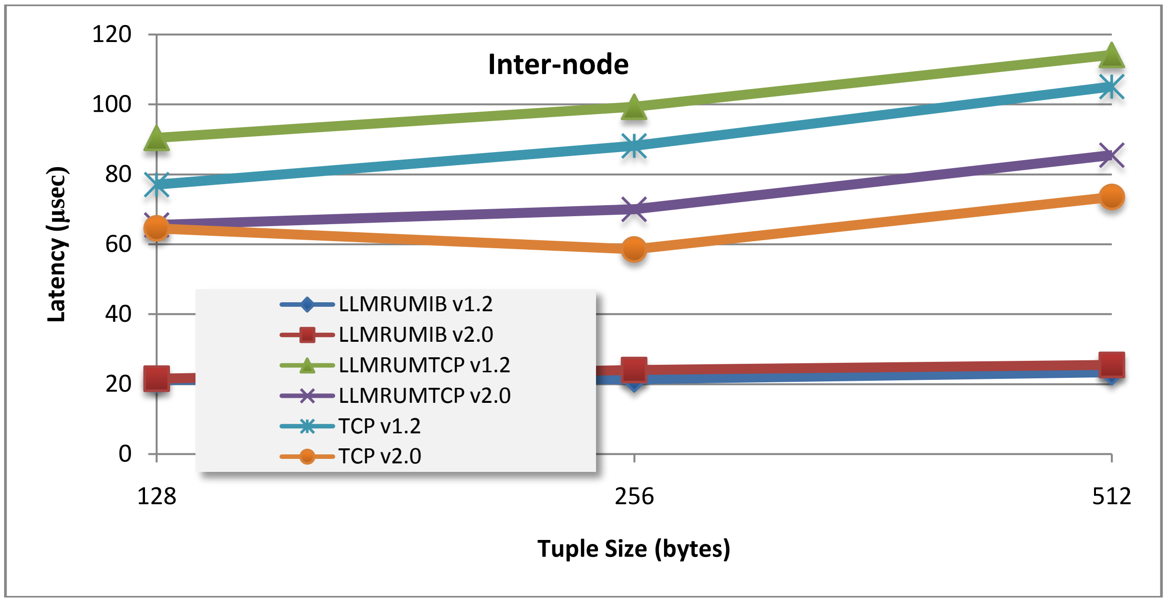

Figure 19 Comparison of Streams v1.2 and v2.0 latencies between 2 nodes

Conclusions

We have now looked at all of the various ways of passing data from one operator to another using Streams v2.0. This ranges from the fastest (fused into a single thread) to the slowest (plain TCP over 1Gb Ethernet between hosts). Given complicated differences in system use (like when using threaded ports to bring additional cores into play within a group of fused operators), the choice of which transport to use is not a simple either-or proposition.

Fused operators can pass data at up to 20 million tuples per second, regardless of tuple size. Placing a threaded port (a new feature in Streams v2.0) between the operators limits throughput to 2.5 million tuples per second, but allows a pipeline of processing to take advantage of multiple processing cores. LLM over InfiniBand can, within a node, offer approximiately the same throughput as threaded ports for smaller tuples (512 bytes and smaller) but is bandwidth limited for larger tuples. LLM over TCP reaches throughputs of 10Gbit/sec, but its throughput drops off for tuples larger than 1KB. The plain TCP transport offered by Streams achieves 12Gbit/sec for tuples 2KB and larger, but is much slower than LLM over TCP for smaller tuples.

Between nodes, Streams using LLM over QDR InfiniBand reached 20Gbit/sec, and was well over 1Gbit/sec even for relatively small tuples. The other transports, LLM over TCP and plain TCP, were limited to the 1Gbit/sec provided by our Ethernet. LLM over TCP was much better for smaller tuples, providing over 600 Mbit/sec even for 4-byte tuples.

Low latency can be achieved with Streams using any of the transports for intranode communication. Round-trip (2 hop) times between operators within a node were all between 15 and 30 microseconds. When communicating between 2 nodes, the round-trip time was 20 microseconds when Streams used LLM over InfiniBand. The other transports (LLM over TCP and plain TCP) provided round-trip times between 50 and 80 microseconds, depending mostly on the tuple size.

The new fusion optimizer for the SPL compiler (COLA), automatically chooses which operators to fuse into a single thread. Test results from runs on a single host indicate that COLA's choices do not always improve on the hand tuning that humans can provide, but are a reasonable improvement over no fusion. However, COLA proved to be superior on all of our tests on multiple hosts. With very large SPL programs, where hand tuning is impractical or even impossible, COLA provides good fusion choices.

Aside from the features newly added to Streams v2.0, performance comparisons were made between corresponding Spade programs from Streams v1.2 and SPL programs from Streams v2.0. Streams v2.0 improves on Streams v1.2 in both throughput and latency. The throughput improved for smaller tuples both within a node and between nodes, when using LLM transports. Latency improved for communication within a single node when using LLM, and also improved when communicating over 1GbE using either plain TCP transport or LLM over TCP.

The significance of some of the results presented in this document is not obvious. Providing a guide to the programmer based on these and other results is beyond the scope of this document. A separate document is currently planned to serve that role.

For more information

To learn more about IBM InfoSphere Streams and associated products to build them, visit:

http://www.ibm.com/software/data/infosphere/streams/

IBM