

Optimizing MPI Communication Within Large Multicore Nodes with Kernel Assistance

S. Moreaud, B. Goglin, D. Goodell, R. Namyst

University of Bordeaux RUNTIME team, LaBRI – INRIA, France

Argonne National Laboratory



Context

Interprocess communication

 \rightarrow MPI dominates

Increasing number of cores



Increasing complexity of computing nodes

→ Need for efficient intranode communication Dedicated intranode communication model



Traditional intranode MPI



Double-Buffering

- Using shared memory buffers
- 2 copies required
- Pipelined strategy



Double-Buffering experiment



Dual Quad-Core Intel Xeon 2.33 GHz *Clovertown* Intel MPI Benchmark (*IMB*) *Pingpong* MPICH2-nemesis MPI implementation



IMB Pingpong with Double-Buffering (2.33GHz Clovertown)

6000 Double buffering: shared cache Double buffering: no shared cache 5000 4000 Throughput (MiB/s) 3000 2000 1000 0 4KiB 16KiB 64KiB256KiB 1MiB 4MiB 16MiB 1KiB Message size



6000

IMB Pingpong with Double-Buffering no cache re-usage (2.33GHz Clovertown)

DB: shared cache DB: no shared cache DB: shared cache DB: shared cache DB: no shared cache





Double-Buffering So what?

- Efficient for small messages
 - latency for 0-byte MPI message < 200ns
- X High CPU use
- X Cache pollution
- X Bandwidth consumption
 - \rightarrow Need large shared cache to be efficient

How to improve performance for large data transfers?



MPICH2-Nemesis

	Application	
	Ch3	
	Nemesis	
Shared Memory	Large Message Transfer (LMT))
64	shm-copy ?	User space

Kernel space $\frac{8}{8}$



MPICH2-Nemesis + KNEM





What's KNEM?

KNEM (Kernel Nemesis)

- Dedicated Linux kernel module
- Single memory copy
- Synchronous and asynchronous modes
- KNEM+I/OAT
 - Offloading copy on Intel Input/Output Acceleration Technology
 - Efficient copy in background
 - CPU and cache not involved
 - Performance does not depend on process location

Available in MPICH2 and soon in OpenMPI



KNEM: Point to point performances

(2.33GHz Clovertown)





KNEM: Point to point performances

(2.33GHz Clovertown)





KNEM: Point to point performances

(2.33GHz Clovertown)





What to expect?

Intranode communication performance vary with

- Strategy (double buffering, KNEM, KNEM+I/OAT)
- Placement, cache size
- Optimization by dynamically switching between strategies?

What is the impact of the location on larger architectures?

What about contention?

- Collective operations?
- Application?



Experimentation Platform

4 Intel-based architectures

- Dual Quad-Core Clovertown 2.33 GHz
- Dual Quad-Core Nehalem 2.66 GHz
- Quad Hexa-Core Dunnington 2.66 GHz
- 4 Quad Hexa-Core *Dunnington* 2.66 GHz NUMA nodes (without I/OAT)

Various number of cores (8, 24, 64)

Different shared caches



Experimentation Platform Relative placements

S	ystem(187GB)												
	Node#0(47GB)												
	Socket#1				Socket#0	cket#0 Socket#2							
	L3(16MB)				L3(16MB)				L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 P#0 Core#1 P#4	Core#2 P#8 Core#3 P#12	Core#4		Core#0 P#1 P#5	Core#2 P#9	Core#4		Core#0 P#2 Core#1 P#6	Core#2 P#10 Core#3 P#14	Core#4		
	Node#1(47GB)												
	Socket#4				Socket#5				Socket#6				
	L3(16MB)				L3(16MB)			L3(16MB)					
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 Core#1 P#28	Core#2 P#32 Core#3 P#36	Core#4 P#40		Core#0 Core#1 P#29	Core#2 P#33 Core#3 P#37	Core#4 P#41 P#45		Core#0 P#26 Core#1 P#30	Core#2 P#34 Core#3 P#38	Core#4 P#42		
	Node#2(47GB)												
	Socket#8				Socket#9				Socket#10				
	L3(16MB)				L3(16MB)				L3(16МВ)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		



Experimentation Platform Relative placements

27	/stem(18/GB)													
	Node#0(47GB)													
	Socket#1				Socket#0					Socket#2				
	L3(16MB)				L3(16MB)					L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KB
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB	,	L1(32KB) L1(3	32KB) L1(32KB)	L1(32KB)	L1(32KB) L1(32	кв)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)
	Core#0 P#0 Core#1 P#4	Core#2 P#8 Core#3 P#12	Core#4 P#16 Core#5 P#20		Core#0 Core	e#1 Core#2	Core#3 P#13	Core#4	#5 21	Core#0 P#2	Core#1 P#6	Core#2	Core#3 P#14	Core#4
	Node#1(47GB)													
	Socket#4				Socket#5					Socket#6				
	L3(16MB)				L3(16MB)					L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KE
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB	,	L1(32KB) L1(3	32KB) L1(32KB)	L1(32KB)	L1(32KB) L1(32	кв)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)
	Core#0 P#24 Core#1 P#28	Core#2 P#32 Core#3 P#36	Core#4 P#40 Core#5 P#44		Core#0 P#25	e#1 Core#2	Core#3 P#37	Core#4 P#41	#5 +5	Core#0 P#26	Core#1 P#30	Core#2 P#34	Core#3 P#38	Core#4
	Node#2(47GB)													
	Socket#8				Socket#9					Socket#10				
	L3(16MB)				L3(16MB)					L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KB)		L2(3072KB



11/2260)

Experimentation Platform Relative placements

- 33	ystem(1876B)												
	Node#0(47GB)												
	Socket#1				Socket#0			זר	Socket#2				
	L3(16MB)				L3(16MB)				L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 P#0 P#4	Core#2 P#8 Core#3 P#12	Core#4 P#16 Core#5 P#20		Core#0 P#1 Core#1 P#5	Core#2 P#9 P#13	Core#4		Core#0 P#2 Core#1 P#6	Core#2 P#10 Core#3 P#14	Core#4		
	Node#1(47GB)												
	Socket#4			ו	Socket#5			זר	Socket#6				
	L3(16MB)				L3(16MB)			L3(16MB)					
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 P#24	Core#2 P#32 Core#3 P#36	Core#4 P#40 Core#5 P#44		Core#0 P#25 Core#1 P#29	Core#2 P #33 Core#3 P#37	Core#4		Core#0 P#26 Core#1 P#30	Core#2 P#34 Core#3 P#38	Core#4		
	Node#2(47GB)												
	Socket#8				Socket#9				Socket#10				
	L3(16MB)				L3(16MB)			L3(16MB)					
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		



L1(32KB)

Experimentation Platform Relative placements

| [1/(32KB)] [1/(

L.1(32KB)

_	ystem(1876B)												
	Node#0(47GB)												
	Socket#1				Socket#0			ן ן ר	Socket#2				
	L3(16MB)				L3(16MB)	L3(16MB)				L3(16MB)			
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 P#0 Core#1 P#4	Core#2 P#8 Core#3 P#12	Core#4 P#16 Core#5 P#20		Core#0 P#1 Core#1 P#5	Core#2 P#9 Core#3 P#13	Core#4 P#17 Core#5 P#21		Core#0 P#2 Core#1 P#6	Core#2 P#10 Core#3 P#14	Core#4		
				_				_					
	Node#1(47GB)												
	Socket#4				Socket#5			ו	Socket#6				
	L3(16MB)				L3(16MB)				L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)		
	Core#0 P#24 Core#1 P#28	Core#2 P#32 Core#3 P#36	Core#4 P#40		Core#0 P#25 Core#1 P#29	Core#2 Core#3 P#37	Core#4 P#41 Core#5 P#45		Core#0 P#26 Core#1 P#30	Core#2 P#34 Core#3 P#38	Core#4		
	Node#2(47GB)												
	Socket#8				Socket#9			ו	Socket#10				
	L3(16MB)				L3(16MB)				L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KE		

1 (32KB)

L1(32KB)

(סאככ/ בו



Experimentation Platform Relative placements

S	ystem(187GB)														
	Node#0(47GB)														
	Socket#1][Socket#0					s	ocket#2				
	L3(16MB)				L3(16MB)						L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)			L2(3072KB)		L2(3072KB)		L2(3072K
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)		L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)
	Core#0 P#0 Core#1 P#4	Core#2 P#8 Core#3 P#12	Core#4 P#16 P#20		Core#0	Core#2 P#9	Core#3	Core#4	Core#5 P#21		Core#0 P#2	Core#1 P#6	Core#2	Core#3	Core#4
										_					
	Node#1(4/GB)									_					
	Socket#4				Socket#5]	Sa	ocket#6				
	L3(16MB)				L3(16MB)						L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)			L2(3072KB)		L2(3072KB)		L2(3072K
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)		L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)
	Core#0 Core#1 P#28	Core#2 P#32 Core#3 P#36	Core#4 P#40 P#44		Core#0 P#25 Core#1 P#29	Core#2	Core#3 P#37	Core#4	Core#5 P#45		Core#0 P#26	Core#1	Core#2	Core#3	Core#4
	Node#2(47GB)														
	Socket#8				Socket#9					Se	ocket#10				
	L3(16MB)				L3(16MB)						L3(16MB)				
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)		L2(3072KB)			L2(3072KB)		L2(3072KB)		L2(3072K
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	Г	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)	L1(32KB)



Experimentation Platform Relative placements

L1(32KB)

23	ystem(187GB)										
	Node#0(47GB)										
	Socket#1				Socket#0][Socket#2		
	L3(16MB)				L3(16MB)		L3(16MB)	L3(16MB)			
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072K
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)
	Core#0 P#0 P#4	Core#2 P#8 Core#3 P#12	Core#4 P#16 Core#5 P#20		Core#0 P#1 Core#1 P#5	Core#2 P#9 Core#3 P#13	Core#4 P#17 P#21		Core#0 P#2 Core#1 P#6	Core#2 P#10 Core#3 P#14	Core#4
	Node#1(47GB)							_			
	Socket#4				Socket#5][Socket#6		
	L3(16MB)				L3(16MB)				L3(16MB)		
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072K
	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB) L1(32KB)		L1(32KB) L1(32KB)	L1(32KB) L1(32KB)	L1(32KB)
	Core#0 P#24	Core#2 P#32 Core#3 P#36	Core#4 P#40		Core#0 P#25 Core#1 P#29	Core#2 P#33 Core#3 P#37	Core#4 P#41 P#45		Core#0 P#26 Core#1 P#30	Core#2 P#34 Core#3 P#38	Core#4
				_				_			
	Node#2(47GB)										
	Socket#8				Socket#9][Socket#10		
	L3(16MB)				L3(16MB)				L3(16MB)		
	L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072KB)		L2(3072KB)	L2(3072KB)	L2(3072K

(ave)

(avcc) (



Impact of process placement IMB Pingpong (NUMA Dunnington)





Impact of process placement

- Shared memory double copy faster than KNEM for small messages
- KNEM better for medium and large messages without shared cache (> 16 KiB)
- I/OAT interesting starting at 16 KiB, except on Nehalem
- Switching between strategies point-to-point model based
- What about collective operations?



Collectives: N-to-One Pattern IMB Reduce (2.66 GHz Nehalem, np=8)





Collectives: One-to-N Pattern IMB Scatter (2.66 GHz Dunnington, np=16)





Collectives: N-to-N Pattern IMB Alltoall (2.33 GHz Clovertown, np=8)

3000 <u>۴۰۰۰*</u>····¥····¥ 2500 Aggregated throughput (MiB/s) 2000 1500 1000 500 0 256B 1KiB 4KiB 16KiB 1MiB 64KiB 256KiB 4MiB Message size



Collectives: N-to-N Pattern IMB Alltoall (NUMA Dunnington, np=64)





Collective operations

All-to-N, One-to-N Pattern

- Double buffering still faster for small messages
- KNEM more interesting for larger messages (>128KiB)
- Behavior and threshold vary slightly with the hosts

N-to-N Pattern

- Alltoall: biggest dependency of placement and topology
 - KNEM improve performance on all architectures
 - \rightarrow from 50% to 100%
 - \rightarrow significantly reducing contention



NAS Parallel Benchmark

Machine	Benchmark	Nemesis	KNEM	Speedup
	ft.B.8	14.60s	13.90s	+5.0%
Hannibal9	ft.C.8	63.77s	60.31s	+5.7%
Παιπιμαιο	is.B.8	0.70s	0.60s	+16.6%
	is.C.8	2.81s	2.41s	+16.6%
	ft.B.8	40.43s	36.02s	+12.2%
Dillo	ft.C.8	175.46s	158.36s	+10.8%
DIIIO	is.B.8	2.42s	1.89s	+12.2%
	is.C.8	10.04s	8.02s	+25.2%
	ft.B.16	24.14s	21.70s	+11.2%
Idkopp24	ft.C.16	97.65s	85.73s	+13.9%
IUKUIIIZ4	is.B.16	1.29s	0.99s	+30.3%
	is.C.16	5.88s	4.43s	+32.7%
	ft.C.64	31.91s	28.82s	+10.7%
Portha06	ft.D.64	727.37s	645.36s	+12.7%
Derthago	is.C.64	3.17s	2.29s	+38.4%
	is.D.64	65.00s	52.72s	+23.3%



Conclusion

Analysis of intranode MPI communication

- Double Buffering, KNEM kernel-assisted single copy
- Different multicore machines
- \rightarrow Double-Buffering more sensitive to placement
- \rightarrow KNEM interesting for large message
- \rightarrow Complex behavior of collective operations
 - Dynamically choosing the best strategy is difficult
- \rightarrow I/OAT only useful for obsolete architectures
- \rightarrow Large performance improvement with KNEM on NAS



Future Works

Improving the KNEM interface

- send/receive-oriented \rightarrow multiple accesses to a single buffer
- receiver-directed data transfer relaxed \rightarrow All-to-one

Improving dynamic adaptation to strategy

- Inside collective operation algorithms
- Processes location, message size...



Questions?

stephanie.moreaud@labri.fr (goodell@mcs.anl.gov)

http://runtime.bordeaux.inria.fr/knem http://www.mcs.anl.gov/research/projects/mpich2