CACHE LINE AWARE OPTIMIZATIONS FOR CCNUMA SYSTEMS

24th ACM International Symposium on High-Performance Parallel and Distributed Computing

HPDC'15, Portland, 2015

Sabela Ramos (sramos@udc.es) GAC, Universidade da Coruña (Spain) Torsten Hoefler (htor@inf.ethz.ch) SPCL, ETH Zurich (Switzerland)

WHAT IS THE PROBLEM?

•The increase in

- Number of cores per processor
- Complexity of memory hierarchies

1 , ,													
		DRAM	l Bank										
					Core 0	Core 1	Core 2	Core 3		Core 0	Core 1	Core 2	
	Core 0	Core 1	Core 2	Core 3	u	u	u	u		u	u	u	I
DRAM Bank	u	u	u	u	12	12	12	12	QPI	12	12	12	
	1.2	12	12	12		Last Level	Cache (LLC)		1	Last Leve	l Cache (LLC)
Core 0 Core 1		Last Level C	Cache (LLC)	13	12	12	12	L2		L2	12	12	
- u - u	u	12 11	u	12	u	u	u	u		u	u	u	
L2	Core 4	Core 5	Core 6	Core 7	Core 4	Core 5	Core 6	Core 7		Core 4	Core 5	Core 6	

DRAM Bank 0 (NUMA)

DRAM Bank 1 (NUMA)

11 12

L2 L1 Core 7

Programmability is maintained through cache coherence

•Which hides peformance characteristics.

OUR PROPOSAL: CLA DESIGN

• GOAL: help programmers to be Cache-Aware



- 1. Detailed (but simple) performance model of the CC protocol
- 2. Methodology to translate algorithms into models
- 3. Select/Optimize/Design algorithms

OUR TESTBED

- Dual socket Intel Xeon Sandy Bridge E5-2660
- CC protocol: MESIF

DRAI	M Banl	JMA)		DRAI	M Ban	k 1 (NI	JMA)	
Core 0	Core 1	Core 2	Core 3		Core 0	Core 1	Core 2	Core 3
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
256 kiB L2	256 kIB L2	256 kiB L2	256 kiB L2	QPI	256 kiB L2	256 kiB L2	256 kIB L2	256 kiB L2
2	0MiB Last Le	evel Cache (L	LC)		2	0MiB Last Le	evel Cache (L	LC)
256 kiB L2	256 kiB L2	256 kiB L2	256 kiB L2		256 kiB L2	256 kiB L2	256 kiB L2	256 kiB L2
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
Core 4	Core 5	Core 6	Core 7		Core 4	Core 5	Core 6	Core 7

- Single-Line Transfers
- Multi-Line Transfers

DRAM		DRAI	M Banl	< 1 (N	JMA)			
Core 0	Core 1	Core 2	Core 3		Core 0	Core 1	Core 2	Core 3
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
256 kiB L2	256 kIB L2	256 kIB L2	256 kiB L2	QPI	256 kiB L2	256 kiB L2	256 kIB L2	256 kiB L2
20	0MiB Last Le	vel Cache (L	.LC)	-	2	LC)		
256 kiB L2	256 kiB L2	256 kiB L2	256 kiB L2		256 kiB L2	256 kiB L2	256 klB L2	256 kiB L2
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
Core 4	Core 5	Core 6	Core 7		Core 4	Core 5	Core 6	Core 7

- Single-Line Transfers
- Multi-Line Transfers

DRAN	ዛ Banl	k 0 (NI	JMA)		DRAI	M Banl	k 1 (NI	JMA)
Core 0	Core 1	Core 2	Core 3		Core 0	Core 1	Core 2	Core 3
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
256 kiB L2	256 kiB L2	256 kIB L2	256 kIB L2	QPI	256 kIB L2	256 kIB L2	256 kIB L2	256 kIB L2
20MiB Last Level Cache (LLC)					2	0MiB Last Le	vel Cache (L	LC)
256 kiB L2	256 kiB L2	256 kiB L2	256 kiB L2		256 kiB L2	256 kiB L2	256 kiB L2	256 kiB L2
32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
Core 4	Core 5	Core 6	Core 7		Core 4	Core 5	Core 6	Core 7

- Single-Line Transfers
- Multi-Line Transfers

	DRAN	4 Banl	< 0 (NI	JMA)		DRAI	ሻ Banl	k 1 (NI	JMA)
L									
	Core 0	Core 1	Core 2	Core 3		Core 0	Core 1	Core 2	Core 3
	32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
	256 kiB L2	256 kIB L2	256 kIB L2	256 kiB L2	QPI	256 kiB L2	256 kIB L2	256 kIB L2	256 kIB L2
	20MiB Last Level Cache (LLC)						0MiB Last Le	vel Cache (L	LC)
	256 kiB L2	256 kiB L2	256 klB L2	256 kiB L2		256 kiB L2	256 klB L2	256 kiB L2	256 kiB L2
	32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1		32 kiB L1	32 kiB L1	32 kiB L1	32 kiB L1
	Core 4	Core 5	Core 6	Core 7		Core 4	Core 5	Core 6	Core 7

L: Local

- Single-Line Transfers
- Multi-Line Transfers

- L: Local
- R: Remote same socket



- Single-Line Transfers
- Multi-Line Transfers

- L: Local
- R: Remote same socket
- Q: Remote different sockets



- Single-Line Transfers
- Multi-Line Transfers

- L: Local
- R: Remote same socket
- Q: Remote different sockets
- I: From memory same socket



- Single-Line Transfers
- Multi-Line Transfers

- L: Local
- R: Remote same socket
- Q: Remote different sockets
- I: From memory same socket
- QI: From memory different sockets



Contention

- Several threads accessing the same line simultaneously
- Sandy Bridge does not suffer from contention

Congestion

- Several threads accessing different lines simultaneously
- The QPI link suffers from congestion ightarrow Regression model





1. PERFORMANCE MODEL Invalidation and Cache-line Stealing

- RFO of a shared line
- Cache-line stealing
 - Caused by
 - Polling
 - False-sharing





2. CLA Cla Pseudo-code

- Copy N lines: cl_copy (cl_t* src, cl_t* dest, int N)
- Wait (poll): cl_wait (cl_t* line, clv_t val, op_t comp=eq)
- Write: cl_write (cl_t* line, clv_t val)

• Add: cl_add (cl_t* line, clv_t val)

- Nodes: CLa operations
- Edges:

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

- Nodes: CLa operations
- Edges:

Edge 1: within the same thread



- Nodes: CLa operations
- Edges:

Edge 1: within the same thread



- Nodes: CLa operations
- Edges:

Edge 1: within the same thread



- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads

Thread 0: S01: cl_write(a,5)



- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads



- Nodes: CLa operations
- Edges:

Edge 2: dependency between threads





- Nodes: CLa operations
- Edges:

- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:

- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:



- Nodes: CLa operations
- Edges:

Edge 1: within the same thread

Edge 2: dependency between threads

Edge 3: sequential restriction between threads

Edge 4: line-stealing caused by non-related operations

• Set of rules to obtain the T_{\min}



	Funct	${f ion}$ OneLineBroadcast(int me, cl_t * my)	$data, tree_t$
	tree)		
	if	tree.parent $!= -1$ then	
[S1]		<pre>cl_wait(tree.pflag[tree.parent],1);</pre>	//one-to-many
[S2]		cl_copy(tree.data[tree.parent],mydata	<i>u</i> ,1);
	if	tree.children > 0 then	
[83]		<pre>cl_copy(mydata,tree.data[me],1);</pre>	
[S4]	1 1	<pre>cl_write(tree.pflag[me],1);</pre>	//one-to-many
[S5]		<pre>cl_wait(tree.sflag[me],tree.children);</pre>	//many-to-one
	if	tree.parent $!= -1$ then	
[S6]		<pre>cl_add(tree.sflag[tree.parent],1);</pre>	//many-to-one
	ei	nd	
	end		



	Funct	$cion$ OneLineBroadcast(int me, cl_t * mg	ydata, tree_t
	tree)		
	if	tree.parent $!= -1$ then	
[S1]		<pre>cl_wait(tree.pflag[tree.parent],1);</pre>	//one-to-many
[S2]		cl_copy(tree.data[tree.parent],mydate	a,1);
	if	tree.children > 0 then	
[S3]		<pre>cl_copy(mydata,tree.data[me].1);</pre>	
[S4]		cl_write(tree.pflag[me],1);	//one-to-many
[S5]		<pre>cl_wait(tree.sflag[me],tree.children);</pre>	//many-to-one
	if	tree.parent $!= -1$ then	
[S6]		<pre>cl_add(tree.sflag[tree.parent],1);</pre>	//many-to-one
	e	nd	
	\mathbf{end}		



]	Function OneLineBroadcast(int me, cl_t * mydata, tree_t
t	tree)
	if tree.parent $!= -1$ then
[S1]	<pre>cl_wait(tree.pflag[tree.parent],1); //one-to-many</pre>
[S2]	<pre>cl_copy(tree.data[tree.parent],mydata,1);</pre>
	if tree.children > 0 then
[83]	cl_copy(mydata,tree.data[me],1);
[S4]	cl_write(tree.pflag[me],1): //one-to-many
[85]	cl_wait(tree.sflag[me],tree.children); //many-to-one
	if tree.parent != -1 then
[S6]	cl_add(tree.sflag[tree.parent],1); //many-to-one
	end
e	end





Thread 0

\mathbf{F}	Yunction OneLineBroadcast(<i>int me, cl_t * mydata, tree_t ree</i>)	Thread 1	53
	if tree.parent != -1 then		
[S1]	cl_wait(tree.pflag[tree.parent], 1); //one-to-many	Parent = 0	I I
[82]	if tree.children > 0 then	#children = 0	V
[S3]	cl_copy(vydata,tree.data[me],1);		C A
[S4]	cl_write(tree.pflag/me],1); //one-to-many		54
[S5]	<pre>cl_wait(tree.sflag/me],tree.children); //many-to-one</pre>		
	if tree.parent $!= -1$ then		i
[S6]	cl_add(tree.sflag[tree.parent],1); //many-to-one		V
	end		
e	nd		(55)
		_	





S6

	Func	tion OneLineBroadcast(int me, cl_t st my	data, tree_t
	tree)		
	if	f tree.parent != -1 then	
[S1]		<pre>cl_wait(tree.pflag[tree.parent],1);</pre>	//one-to-many
[S2]		cl_copy(tree.data[tree.parent],mydata	<i>i</i> , 1);
	if	tree.children > 0 then	
[S3]		<pre>cl_copy(mydata,tree.data[me],1);</pre>	
[S4]		<pre>cl_write(tree.pflag[me],1);</pre>	//one-to-many
[S5]		<pre>cl_wait(tree.sflag[me],tree.children);</pre>	//many-to-one
	if	tree.paren/!= -1 then	
[S6]		<pre>cl_add(tree.sflag[tree.parent],1);</pre>	//many-to-one
	e	nd	
	\mathbf{end}		



	Function OneLineBroadcast(int me, cl_t * mydata, tree_t	
	tree)	
	if tree.parent $!= -1$ then	Three
[S1]	<pre>cl_wait(tree.pflag[tree.parent],1); //one-to-many</pre>	
[S2]	cl_copy(tree.data[tree.parent],mydata,1);	Parent = 0
	if tree.children > 0 then	#children = 0
[S3]	cl_copy(mydata,tree.data/me],1);	
[S4]	cl_write(tree.pflag/me],1); //one-to-many	
[S5]	cl_wait(<i>tree.sflag[me]</i> , <i>tree.children</i>); //many-to-one	S1
	if tree.parent $!= -1$ then	
[S6]	cl_add(tree.sflag[tree.parent],1); //many-to-one	, i i
	end	
	end	S2



PERFORMANCE RESULTS



- Speedup of 14x vs. MPI
- Speedup of 1.8x vs. HMPI

CONCLUSIONS AND DISCUSSION

- Cache-coherency helps programmability
- BUT it complicates performance-centric programming
- The CLa methodology simplifies the analysis of algorithms under heavy thread interaction conditions that affect performance:
 - Contention and congestion
 - Polling
 - Cache-line stealing
- We compared our algorithms (communication and synchronization) with MPI, OpenMP and HMPI obtaining high speedups.

CACHE LINE AWARE OPTIMIZATIONS FOR CCNUMA SYSTEMS

24th ACM International Symposium on High-Performance Parallel and Distributed Computing

HPDC'15, Portland, 2015

Sabela Ramos (sramos@udc.es) GAC, Universidade da Coruña (Spain) Torsten Hoefler (htor@inf.ethz.ch) SPCL, ETH Zurich (Switzerland)