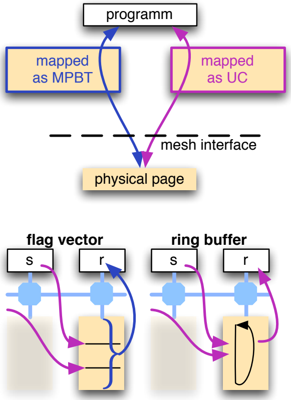
Brandenburg
University of Technology
Cottbus



06 July 2011

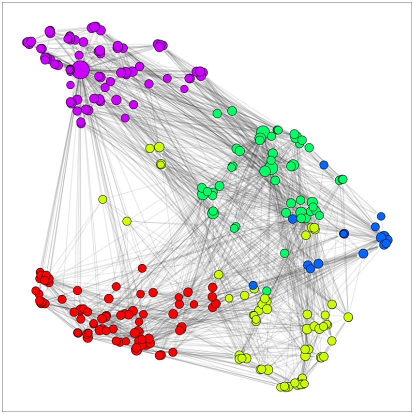
LOW LATENCY COMMUNICATION ON THE INTEL SCC

Randolf Rotta

Brandenburgische Technische Universität Cottbus

APPLICATION BACKGROUND: NETWORK ANALYSIS

Infrastructure for graph algorithms: clustering, layout, . . .



(*Neural Network of worm C.Elegans
Modularity Clustering, LinLog Layout*)

```
for next few nodes $v$:
  compute $\Delta Q(v)$ [parallel]
collect results
choose best, update
repeat
```

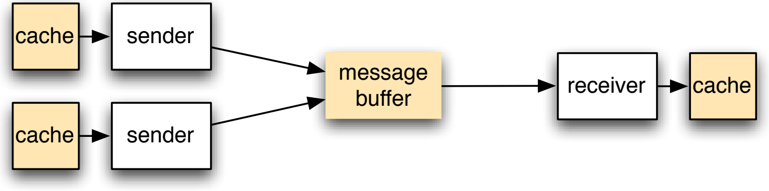
- fine-grained parallelism,
  highly irregular
  $\Rightarrow$ many small tasks
- large shared data,
  SW memory consistency
  $\Rightarrow$ coordination required

Brandenburg
University of Technology
Cottbus

REQUIRES LOW LATENCY MESSAGING

. . . for a high number of small messages (≈1–3 cache lines)

- any-source polling ⇒ more flexible middleware
- low overhead ⇒ enables fine-grained coordination

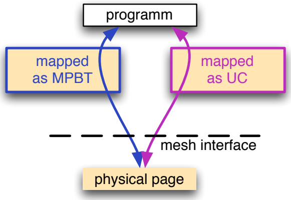


What we achieved so far. . .

- understanding SCC's capabilities
- cost models, benchmarks
- design space for messaging protocols

1. Abusing the SCC

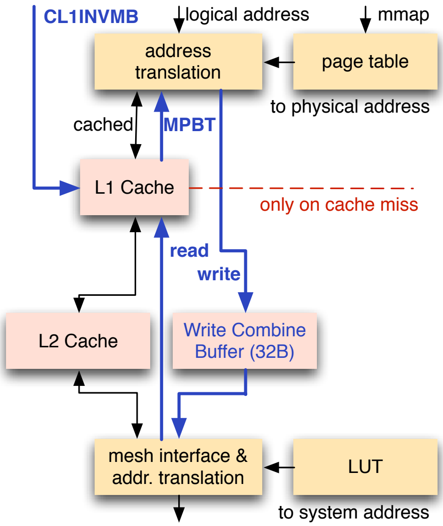
2. Design Issues of Messaging Protocols



READING AND WRITING CACHE LINES (32 BYTE)

. . . using the Message Passing Buffer Type (MPBT)

- logical $\mapsto$ physical addr. mapping by Page Table sets access type
- reads are cached in L1 Cache
- writes go to Write Combine Buffer but *only on cache miss!*
- `CL1INVMB` drops all MPBT lines



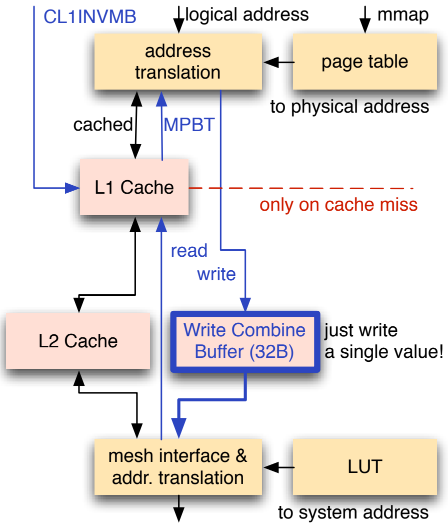
ABUSING THE WRITE COMBINE BUFFER

. . . to write single values (1,2,4 byte)

- transfer triggered, when. . .
  written to all bytes,
  write to other MPBT line,
  write to any other memory (?)

⇒ write value, then do
dummy write to somewhere else

⇒ WCB sends single value
does not overwrite other data

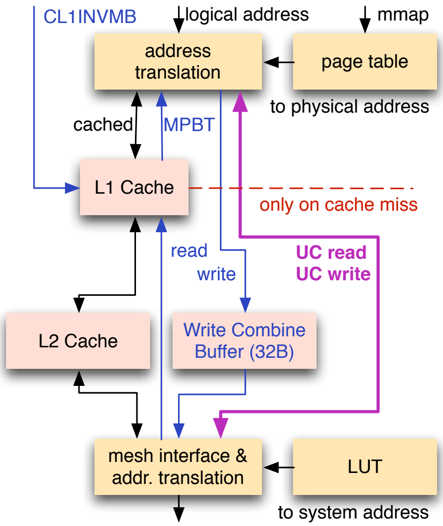


READING AND WRITING SINGLE VALUES (1,2,4 BYTE)

. . . using the Uncached Memory Type (UC)

- read/write not cached at all
  only on cache miss!
- goes directly to the mesh,
  not simulated by CC hardware

⇒ no false-sharing problems

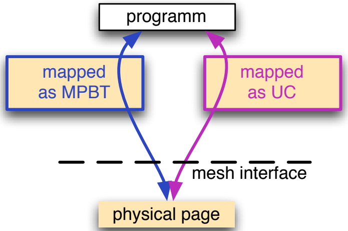


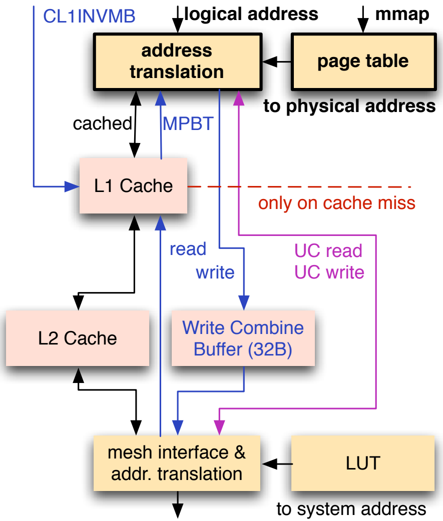
ABUSING THE ADDRESS TRANSLATION

. . . by mapping physical pages twice to different logical addresses

- as UC using `/dev/rckncm`
  as MPBT using `/dev/rckmpb`
- `CL1INVMB` before read/write to UC!

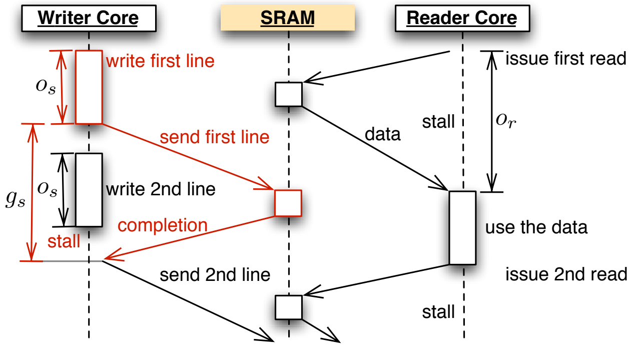
⇒ can mix UC and MPBT access to the same data





MEASURING THE WRITE ACCESS COSTS

Send overhead overlaps; UC write faster than MPBT

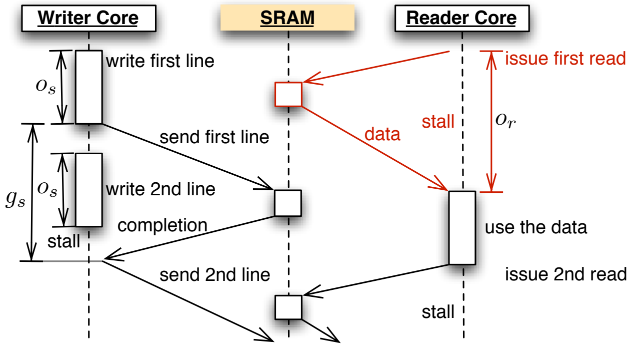


	1 byte (UC)	4 byte (UC)	32 byte (MPBT)
o_s	5	5	28
g_s	48–81	48–81	75–105
o_r	53–86	53–86	58–88

Brandenburg
University of Technology
Cottbus

MEASURING THE READ ACCESS COSTS

Read is blocking; MPBT read as fast as UC

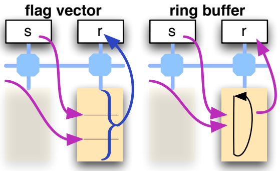


	1 byte (UC)	4 byte (UC)	32 byte (MPBT)
o_s	5	5	28
g_s	48–81	48–81	75–105
o_r	53–86	53–86	58–88

Brandenburg
University of Technology
Cottbus

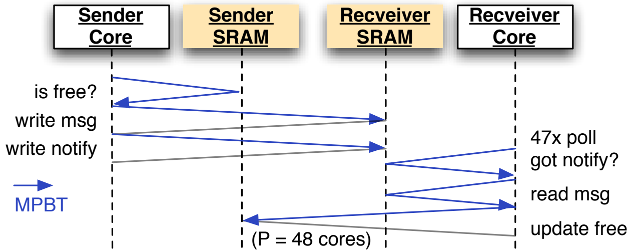
1. Abusing the SCC

2. **Design Issues of Messaging Protocols**



IDENTIFYING BOTTLENECKS OF COMMON PROTOCOLS

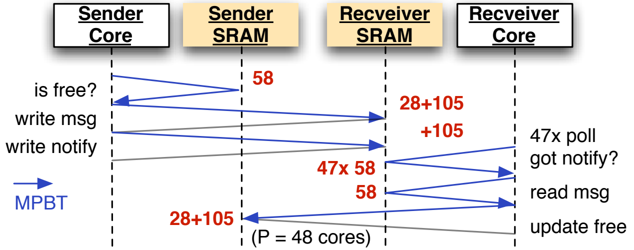
Example: A very traditional protocol from systems with Cache Coherence



- separate point-to-point channel for each pair of cores
  message placed in fixed slot at receiver-side
- receiver-side flag to notify about new message
  sender-side flag to acknowledge received message
  counter trick to omit flag reset
- polling has to read 47 cache lines!

IDENTIFYING BOTTLENECKS: POLLING

. . . applying the cost model



- Round-Trip Time: $>$6426 cycles (mesh costs)
- polling takes the most time: $>$5452 cycles (85%)!
  grows linearly: $\mathcal{O}(58 * P)$ cycles
- most memory unused (fixed p2p allocation)
  grows quadratically: $\mathcal{O}(32 * 3 * P^2)$ bytes

Brandenburg
University of Technology
Cottbus

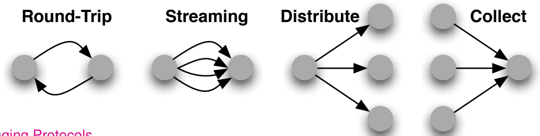
DISSECTING PROTOCOLS INTO A DESIGN SPACE

1. At which level of the system?
2. Sender-side vs. receiver-side placement?
3. Message slot allocation
4. Notification of the receiver
5. Acknowledgement: freeing resources
6. How to wait, when queue is full?

(4–6: flow control)

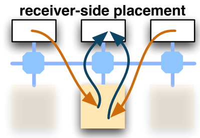
Communication Patterns: How to compare Protocols?

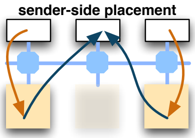
- LogP: Latency, Overhead, Gap, #Processors
- completion time, message throughput, data throughput, scaleability



(2) PLACEMENT AND (3) ALLOCATION

Reasoning based on the cost model...





Which side to put?

- local write+remote read = remote write+local read

$\Rightarrow$ does not matter (beware the symmetry)

Which slot to use?

- heap-like allocation:
  works just on sender-side,
  requires explicit ack from receiver
- implicit allocation on receiver-side:
  exploits the notification, may save the ack

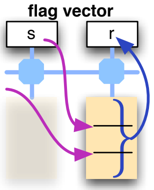
$\Rightarrow$ use the memory, independent of P

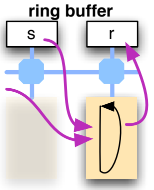
Brandenburg
University of Technology
Cottbus

(4) NOTIFICATION: YOU GOT MAIL!

. . . and how to find it by abusing the WCB or UC

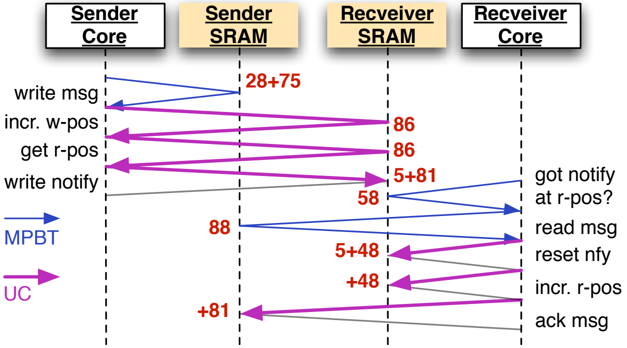
vector	48 flags packed in 2 lines, each 1 byte
	$\Rightarrow$ scan in $\approx$1100 cycles (116 from mesh)
	check groups of 4 bytes first
	$\Rightarrow$ scan in $\approx$220 cycles
$\Rightarrow$	requires P flags, overhead grows with P
ring	w-pos: 1 atomic counter per receiver
	r-pos: updated by receiver
$\Rightarrow$	flags + overhead independent of P, but a lot coordination overhead!
. . .	other ideas !?!





AN IMPROVED PROTOCOL?

Notification: Ring buffer; Allocation: heap with idle-ack-scan

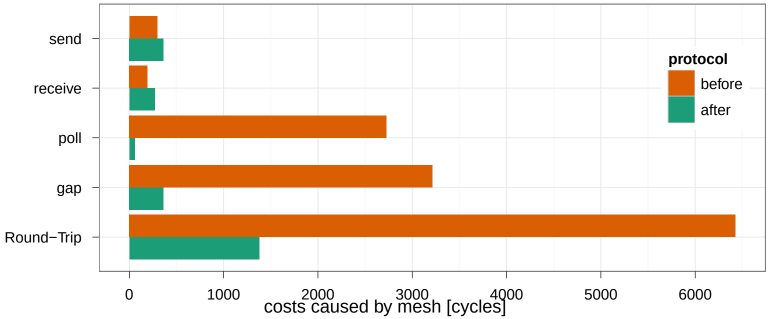


- Round-Trip Time: >1378 cycles (mesh costs), independent of #Processors, shared by all senders
- 380 cycles for actual data transfer, 72% for flow control

CONCLUSIONS: STILL SPACE FOR IMPROVEMENTS

- on-chip Message Passing Buffers (SRAM) are very useful
- real non-cached memory operations mix well with caching
- lack of cache coherence simplifies protocol design
- Flow control still is the bottleneck!

  atomic counters do not fully solve this problem



CONCLUSIONS: STILL SPACE FOR IMPROVEMENTS

- on-chip Message Passing Buffers (SRAM) are very useful
- real non-cached memory operations mix well with caching
- lack of cache coherence simplifies protocol design
- Flow control still is the bottleneck!
  atomic counters do not fully solve this problem

Future Work

- implement more protocols, predict and measure performance
- extend FPGA or hardware:
  special messaging support, asynchronous write (fire&forget)
  $\Rightarrow$ How much improvement possible?
- Cache control instead of Write Combine Buffer?
  $\Rightarrow$ will not change much. . .