Evaluation and Improvements of Programming Models for the Intel SCC Many-core Processor

Carsten Clauss, Stefan Lankes,
Pablo Reble, Thomas Bemmerl

International Workshop on New Algorithms and Programming Models for the Many-core Era (APMM 2011)
As part of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011)

LEHRSTUHL FÜR BETRIEBSSYSTEME
Univ.-Prof. Dr. habil. Thomas Bemmerl

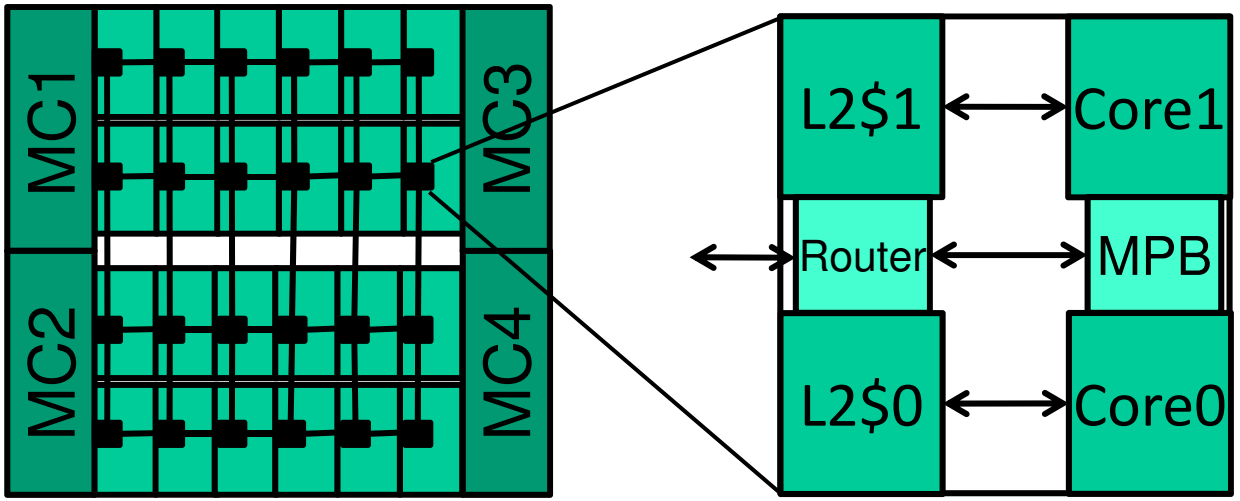
RWTH AACHEN UNIVERSITY

Overview

- The Intel SCC Many-core Processor
- Shared-Memory Models of the SCC
- Message-Passing on the SCC
- Selected Performance Results
- Outlook and Conclusions

The Intel SCC Many-core Processor

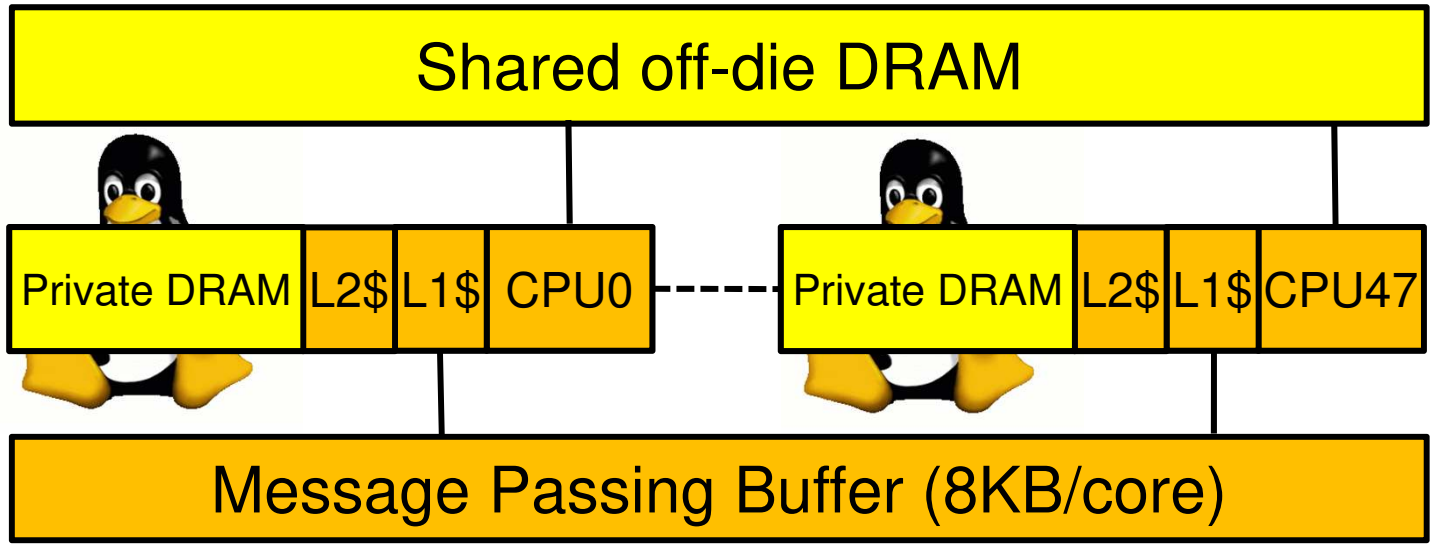
- **Intel Single-Chip Cloud Computer**
  → a *Concept Vehicle* for Many-core Software Research
- **48 Pentium-I Cores arranged in a 6x4 on-die Mesh**
  → 2 Cores and 1 Router per *Tile*



- **On-die *Message-Passing Buffers*** (MBP) / 16kByte per Tile
  → accessible as distributed on-die *Shared-Memory*
- **4 on-die Memory Controllers** (MC1-4)
  → max. 64GByte DDR3 off-die main memory

Shared-Memory Models of the SCC

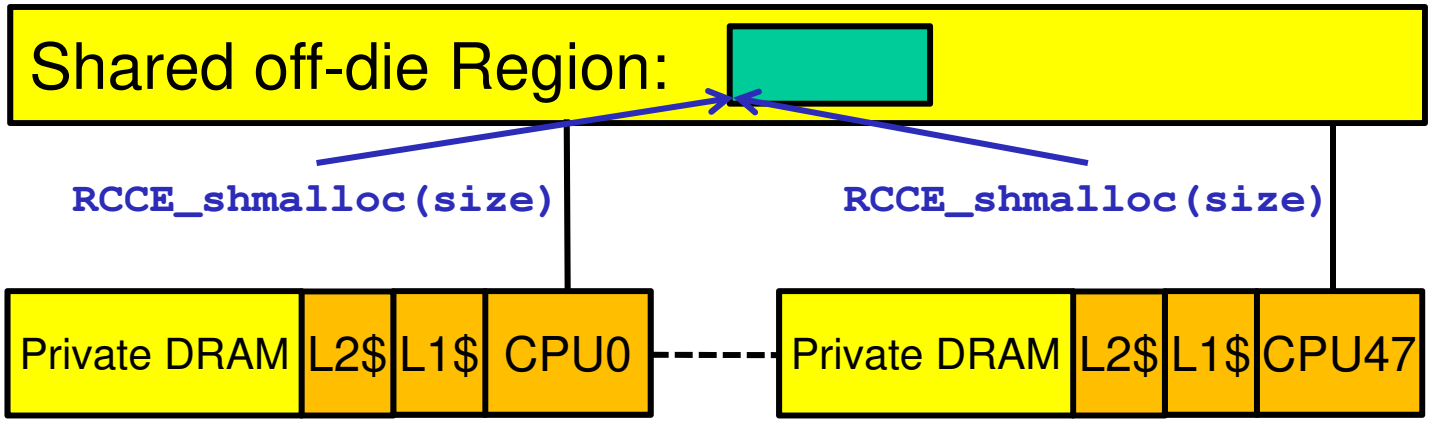
- **Strictly No Cache Coherency**
  → Cluster-on-Chip Architecture
- **Private off-die DRAM Regions (one per Core)**
  → Caches *enabled*! One *Linux* instance per Core!



- **Shared / Global off-die DRAM Region**
  → Caches *disabled* per default! → e.g. for global shared data
- **Shared on-die MPB Regions**
  → Cached in L1, L2 Bypass / Fast L1 Invalidation for MPB-Data

Shared-Memory Models of the SCC

- **The RCCE Communication Library from Intel**
  → An Application Programming Interface (API) for the SCC
- **Using Shared off-die DRAM Region:**
  `void* RCCE_shmalloc(int size)`



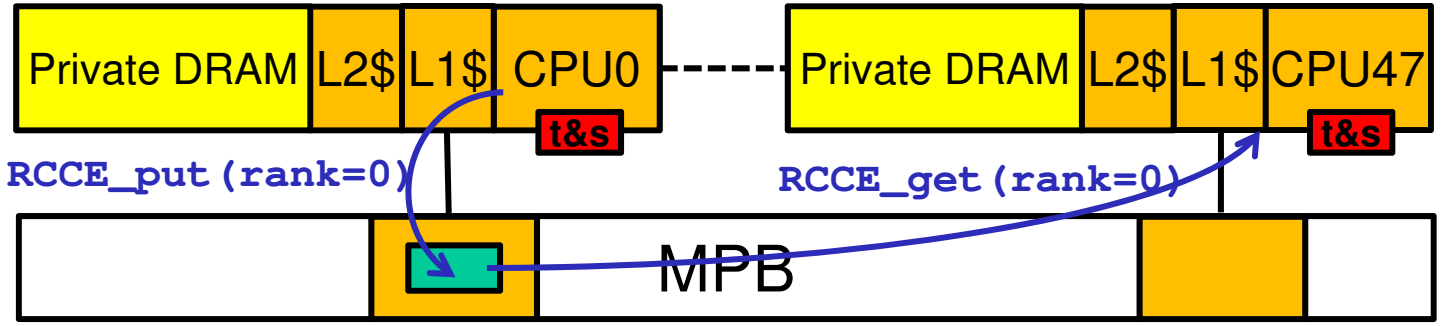
`shmalloc()` is a *collective* function call that returns pointers to a new off-die shared region (*caches disabled per default!*)

- **Caches *can* be enabled on demand**
  → but the SCC does not provide any Cache Coherency!

Shared-Memory Models of the SCC

- **Using Shared on-die MPB Regions:**

```
RCCE_put(void* mpb, void* src, int size, int rank)
RCCE_get(void* dst, void* mpb, int size, int rank)
```



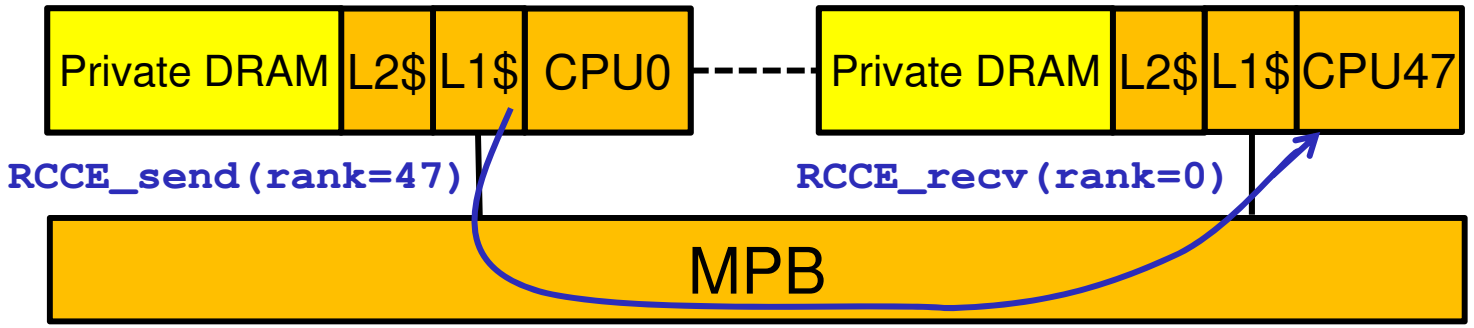
- ***Local* and *Remote* MPB Regions**
  → a core *rank* (or ID) is used for addressing a certain region

- **Synchronization e.g. via 48 *Test-and-Set Registers:***

```
RCCE_acquire_lock(int rank)
RCCE_release_lock(int rank)
```

Message-Passing on the SCC

- **RCCE's Two Sided Communication Interface:**
  → *blocking* send and receive functions



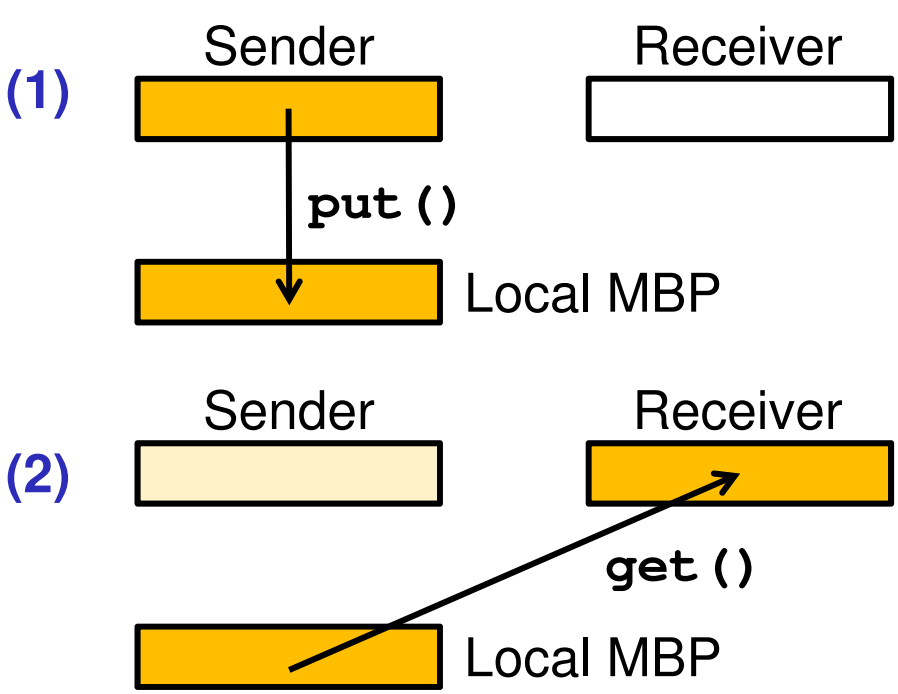
- **Our Improvements: iRCCE**
  → e.g. *non-locking* send and receive functions

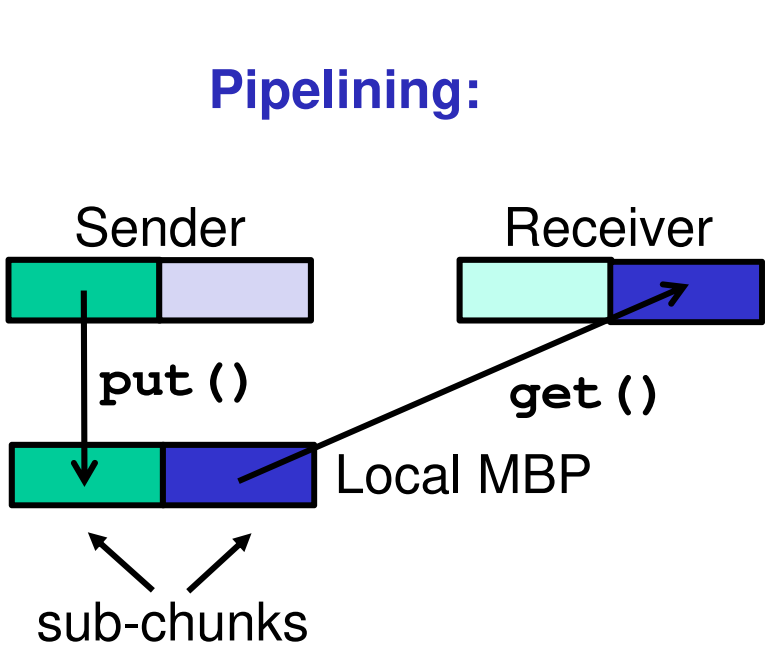
```
iRCCE_isend(void* buf, int size, int rank, iRCCE_sreq *req)
iRCCE_irecv(void* buf, int size, int rank, iRCCE_rreq *req)

iRCCE_isend_test(iRCCE_sreq *req, int* flag)
iRCCE_isend_wait(iRCCE_sreq *req);
...
```

Message-Passing on the SCC

Further Communication Improvements of iRCCE:

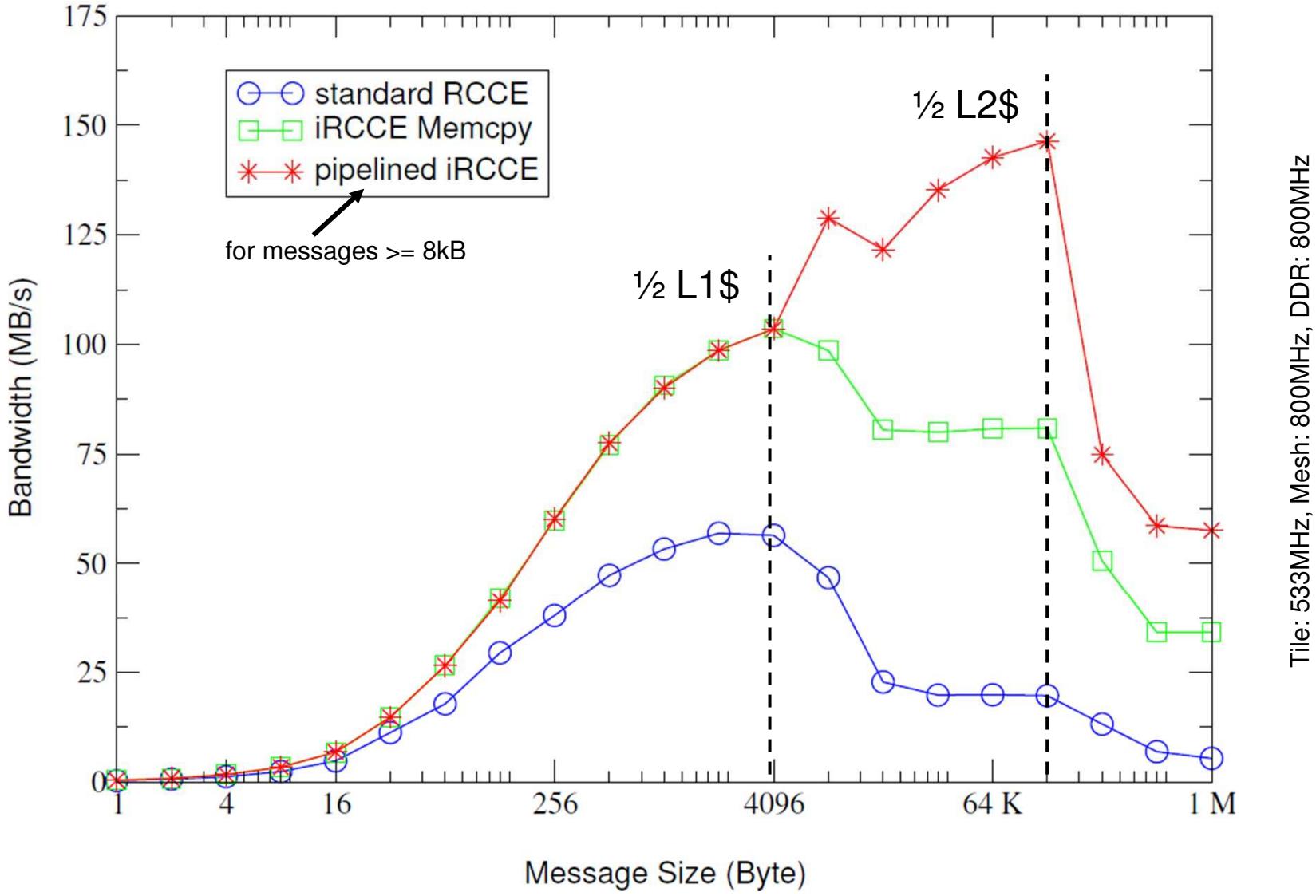
- **Usage of Pentium-optimized `memcpy()` routines**
  → assembler-coded *Prefetching* and *Loop-Unrolling*
- **Applying of Pipelining for large messages**
  → divide local MPB into two sub-chunks and process in parallel





Message-Passing on the SCC

Ping-Pong Measurements



Tile: 533MHz, Mesh: 800MHz, DDR: 800MHz

MPI-based Message-Passing on the SCC

- **MPI is the most popular Message-Passing Interface**
  → why not porting an MPI implementation to the SCC?

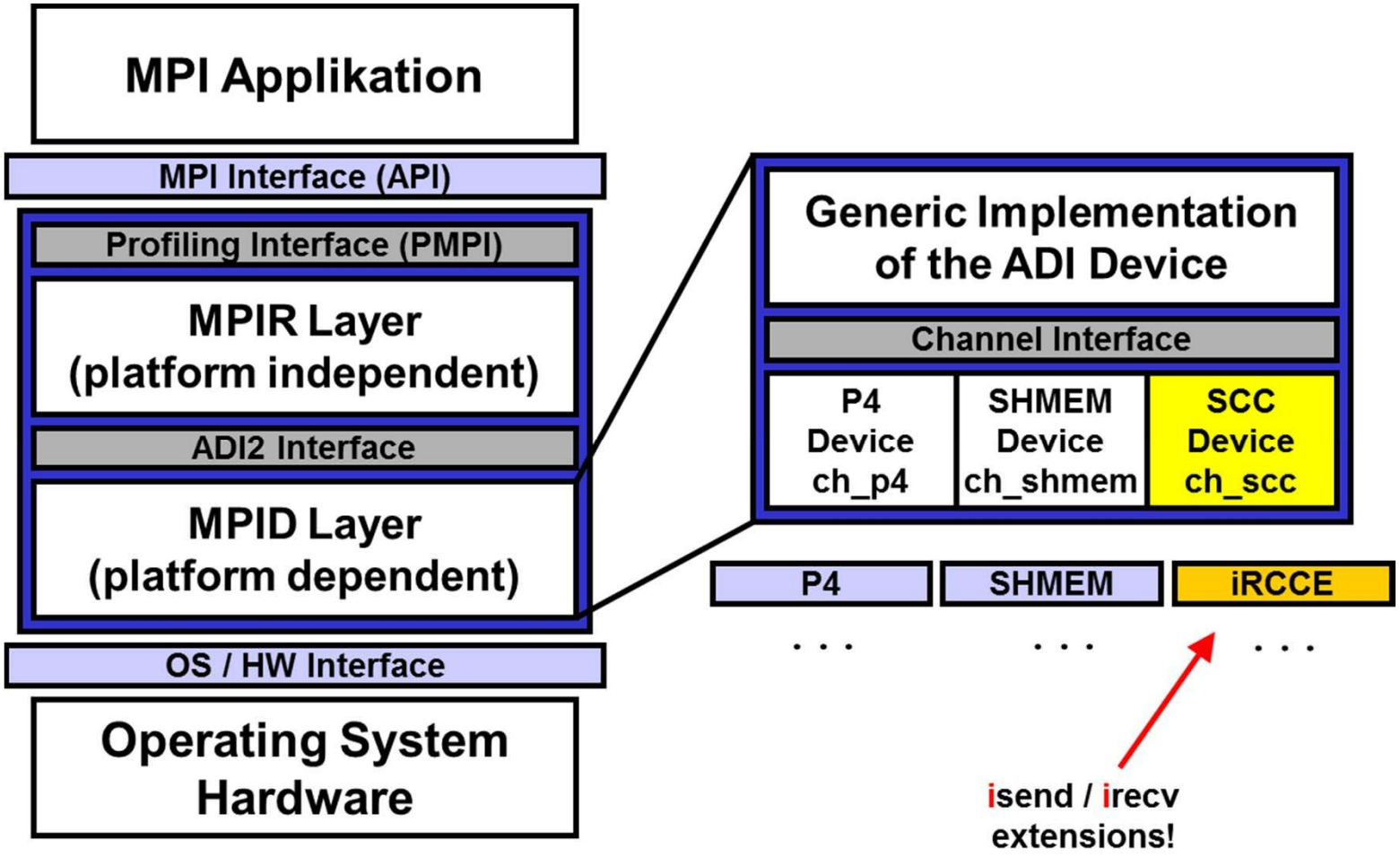
- **Why not just using a *TCP-based* MPI implementation?**
  → the SCC's TCP driver only utilizes *slow off-die* memory ☹

- **The usage of the *fast on-die MPB* would speed-up the core-to-core communication**
  → why not implementing an SCC-customized MPI library?

- **Currently, two of such SCC-MPI implementations exist:**
  → RCKMPI (by Intel)
  → SCC-MPICH (from our Institute)

MPI-based Message-Passing on the SCC

- **The Layered Design of SCC-MPICH:**



MPI-based Message-Passing on the SCC

- **SCC-MPICH provides four Protocols**
  → the choice depends on the message length

- **SHORT Protocol**
  → for short payload and signaling messages

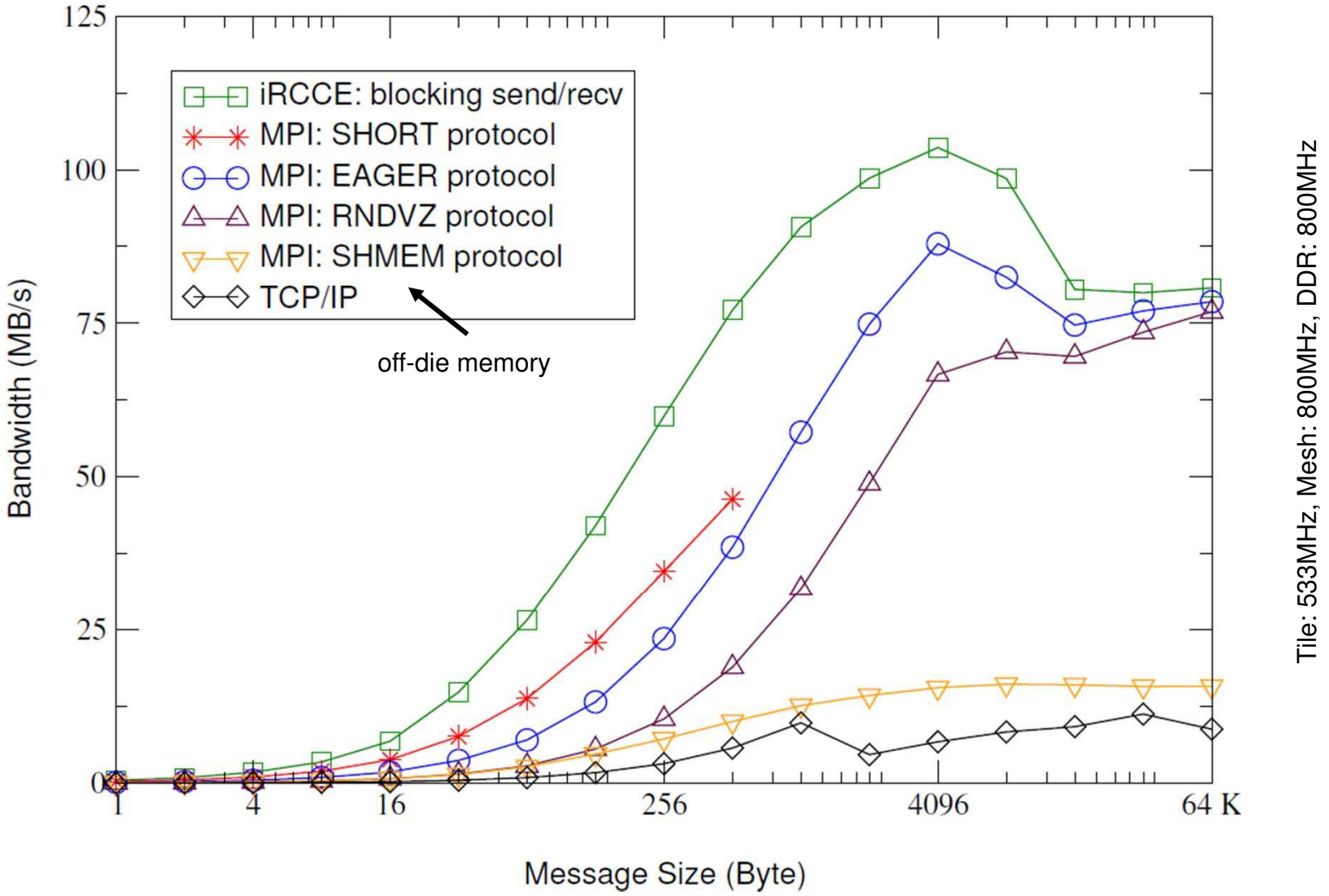
- **EAGER Protocol**
  → for mid-size messages (expected and unexpected)

- **RENDEZVOUS Protocol**
  → for synchronous as well as for large messages

- **SHM-EAGER Protocol**
  → second EAGER that utilizes the shared off-die memory

Message-Passing on the SCC

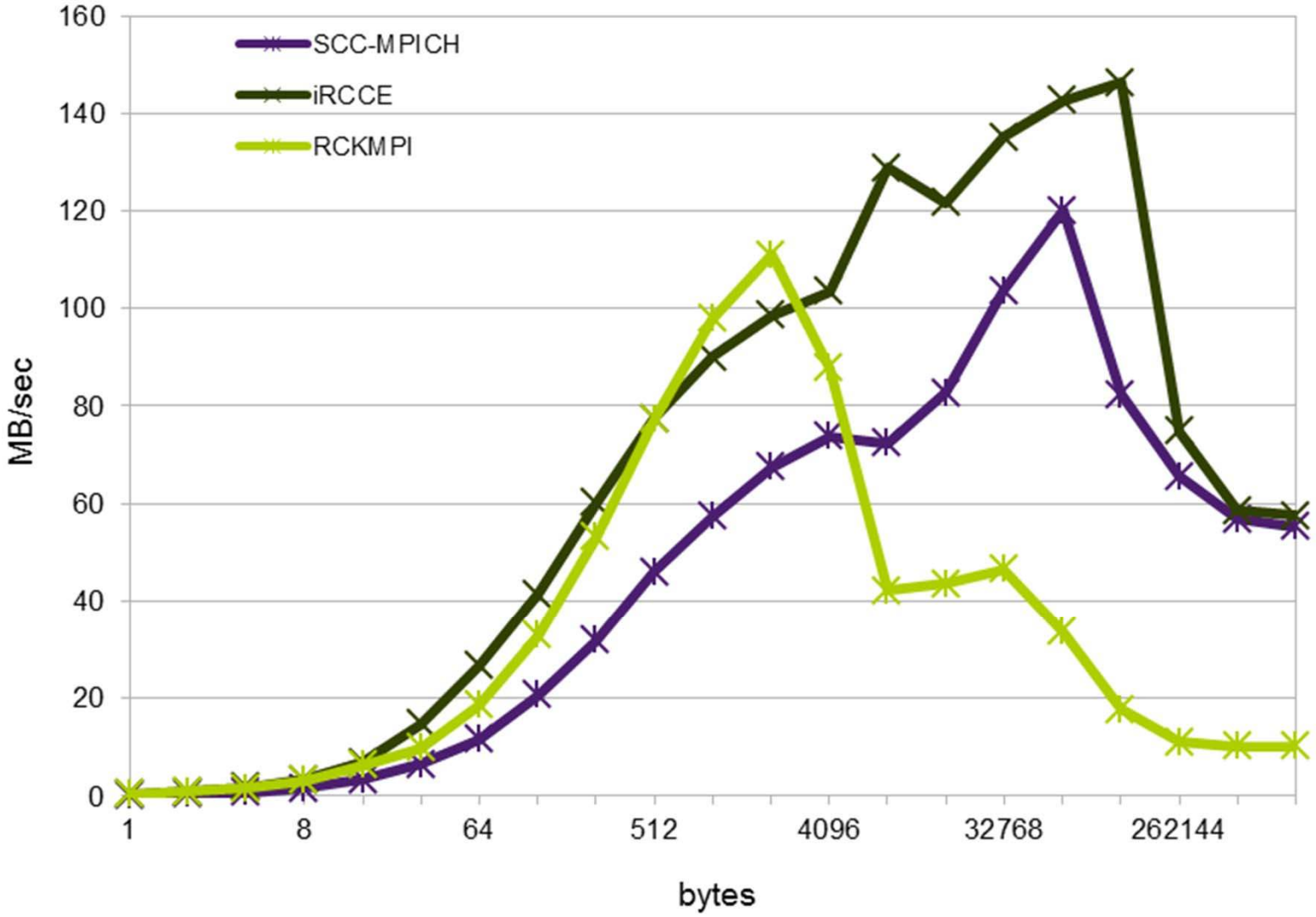
Ping-Pong Measurements



Tile: 533MHz, Mesh: 800MHz, DDR: 800MHz

Message-Passing on the SCC

SCC-MPICH vs. iRCCE vs. RCKMPI



Selected Performance Results

- **Selected Performance Results**
  → comparing different Programming Models for the SCC

- **Benchmark Scenario?**
  → we have used a simple **Jacobi Solver**

- **Dense System of Equations (*Ax=b*)**
  → use the following Iterative Rule to solve:

$$x_i^{m+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^m \right)$$

Selected Performance Results

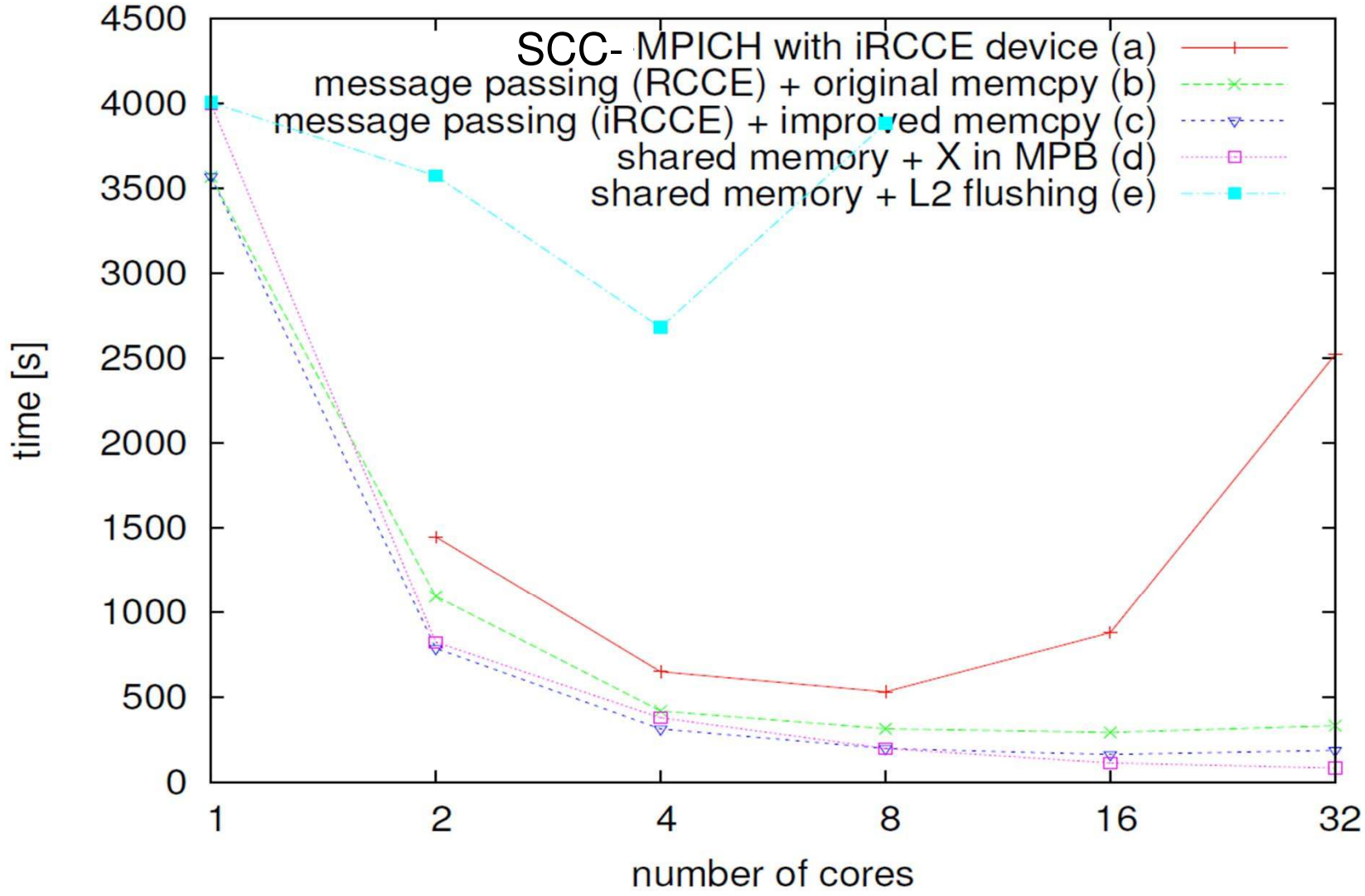
- **The Shared-Memory Case:**
  → where to place *A*, *b* and *x*?
- **Read-only Accesses for A and b**
  → off-die Shared-Memory and *enable* Caches for them
- **Write Accesses for Vector *x***
  → three thinkable approaches:

  1. put *x* into the shared off-die memory (caches disabled)
  2. enable caches and flush L1 and L2 after each iteration
  3. put *x* into the MPB and flush L1 after each iteration

- **The Message-Passing Case:**
  → use (i)RCCE or MPI to exchange *x* after each iteration

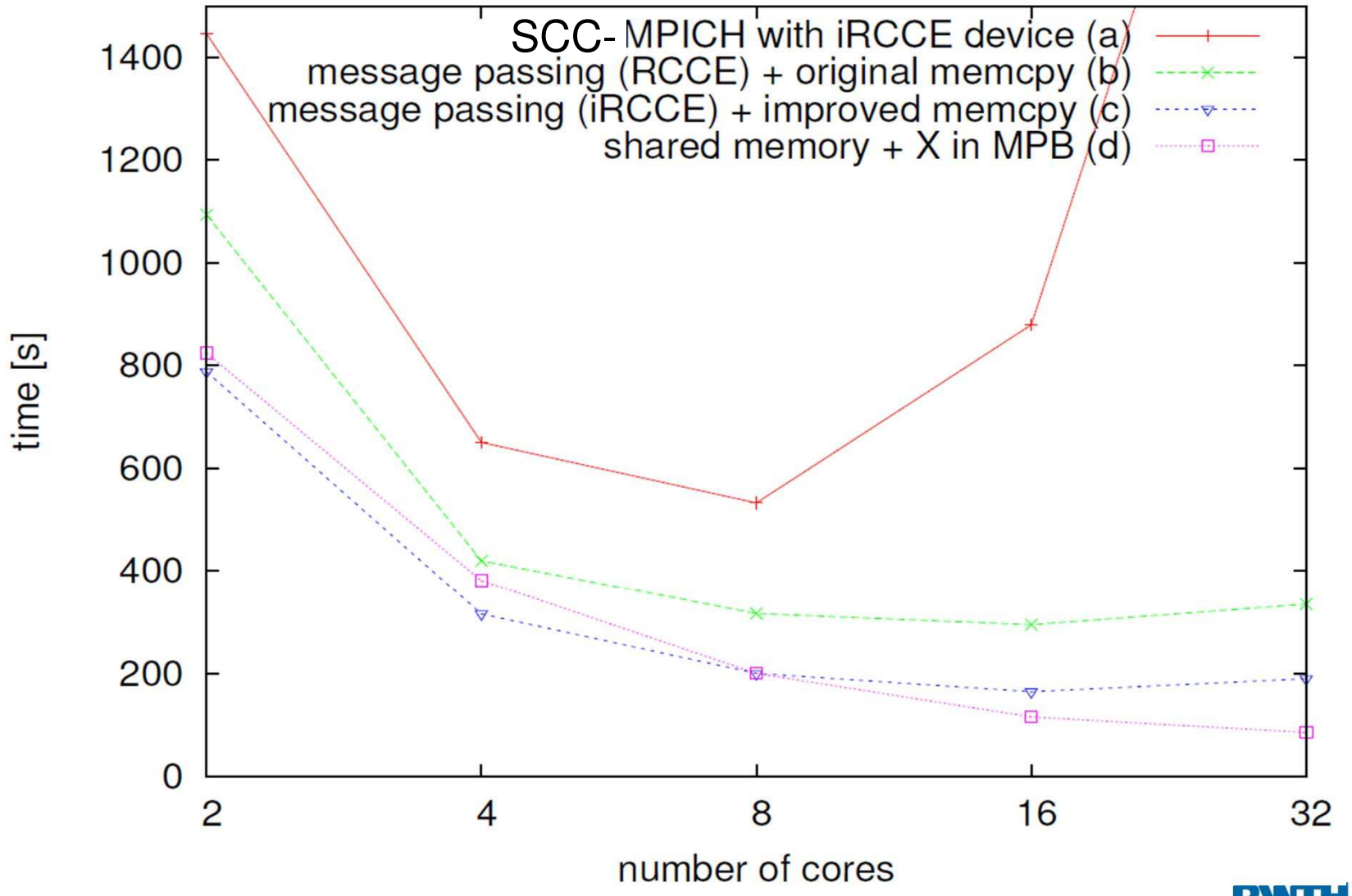
Selected Performance Results

Jacobi Solver: 256x256 matrix, 865000 iterations



Selected Performance Results

Jacobi Solver: 256x256 matrix, 865000 iterations



Conclusions and Outlook

- **Is the SCC a Many-core Processor?**
  → the SCC is at least a prototype for future architectures

- **Do we really need to waive the Cache Coherency?**
  → it seems that cache coherency limits the scalability

- **Is Message-Passing the Model of Choice?**
  → hybrid architectures need hybrid programming models

- **How can I participate in this Many-core related Software Research?**
  → join the Intel *Many-core Application Research Community* (MARC) **`http://communities.intel.com/community/marc`**

Thank you for your attention!
Any Questions?

Acknowledgement:
This research and development was
supported by Intel Corporation

International Workshop on New Algorithms and Programming Models for the Many-core Era (APMM 2011)
As part of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011)

LEHRSTUHL FÜR BETRIEBSSYSTEME
Univ.-Prof. Dr. habil. Thomas Bemmerl

RWTH AACHEN UNIVERSITY