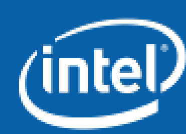
RCCE

A small library for many-core communication

Rob van der Wijngaart
Software and Services Group

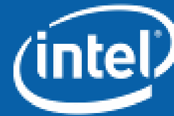
Tim Mattson
Intel Labs

Intel Labs Single-chip Cloud Computer Symposium
February 12, 2010



Agenda

- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management



Top Level Hardware Architecture

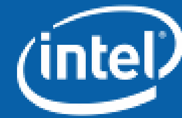
- 6x4 mesh 2 Pentium™ P54c cores per tile
- 256KB L2 Cache, 16KB shared MPB per tile
- 4 iMCs, 16-64 GB total memory

Memory Controller
Memory Controller
Memory Controller
Memory Controller
Tile
R
System I/F

Tile
L2$1
Core 1
Router
MPB
L2$0
Core 0

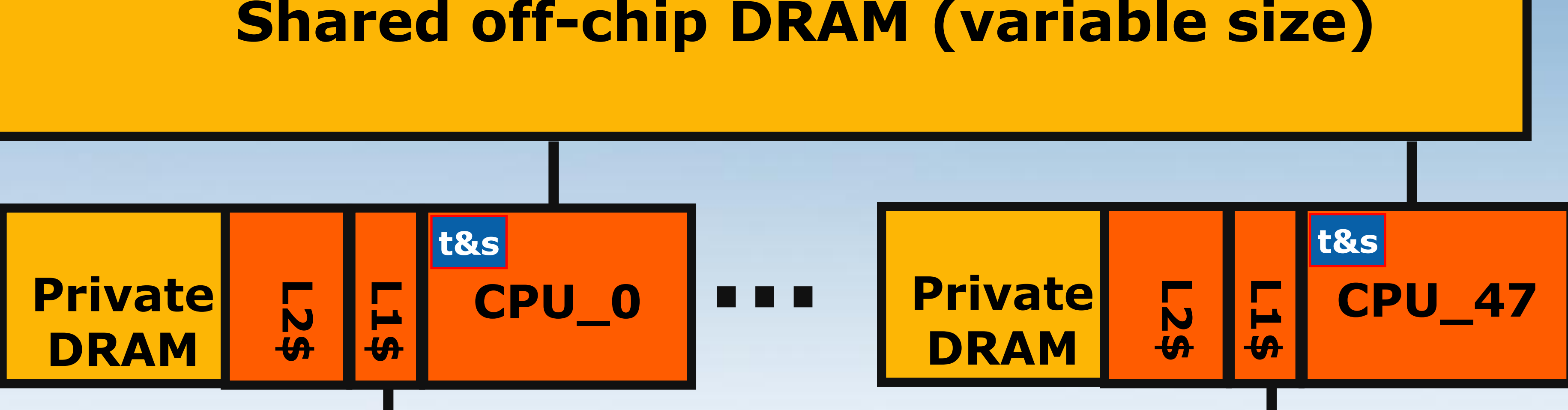
Tile area: ~17 mm^2
SCC die area: ~567 mm^2

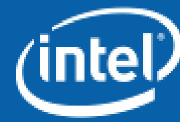
R = router, iMC = integrated Memory Controller, MPB = message passing buffer



Programmer's view of SCC

- 48 x86 cores with the familiar x86 memory model for Private DRAM
- 3 memory spaces, with fast message passing between cores
  (/ means on/off-chip)

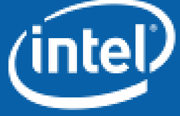




SCC Software research goals

- Understand programmability and application scalability of many-core chips.
- Answer question "what can you do with a many-core chip that has (some) shared non-cache-coherent memory?"
- Study usage models and techniques for software controlled power management
- Sample software for other programming model and applications researchers (industry partners, Flame group at UT Austin, UPCRC, YOU …)

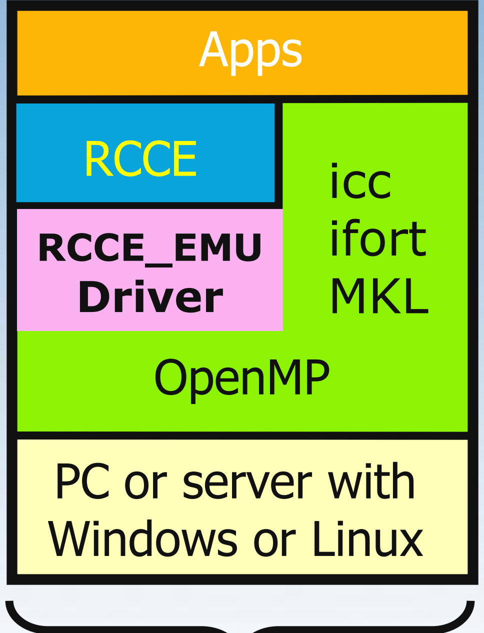
Our research resulted in a light weight, compact, low latency communication library called RCCE (pronounced "Rocky")



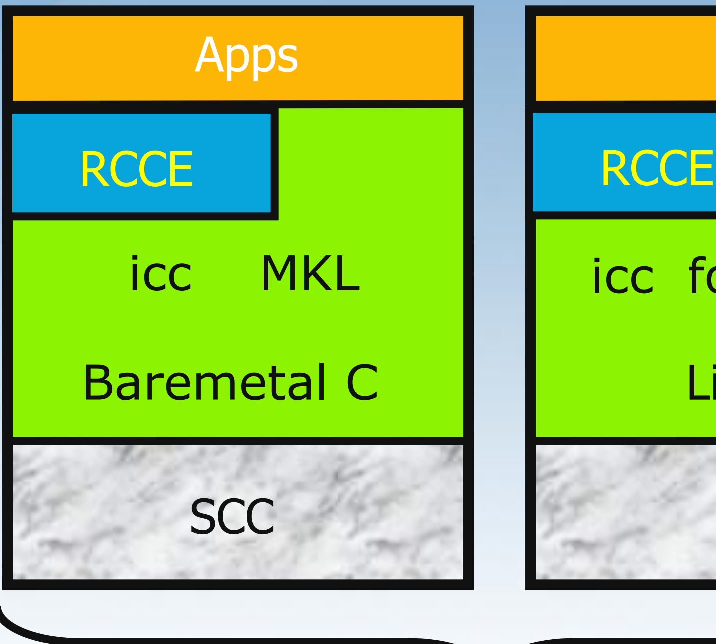
Third party names are the property of their owners.

SCC Platforms

- Three platforms for SCC and RCCE
  - Functional emulator (on top of OpenMP)
  - SCC board with two "OS Flavors" … Linux or Baremetal (i.e. no OS)



Functional emulator, based on OpenMP.



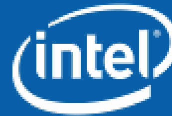
SCC board – NO OpenMP

RCCE supports greatest common denominator between the three platforms

Third party names are the property of their owners.

Agenda

- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management

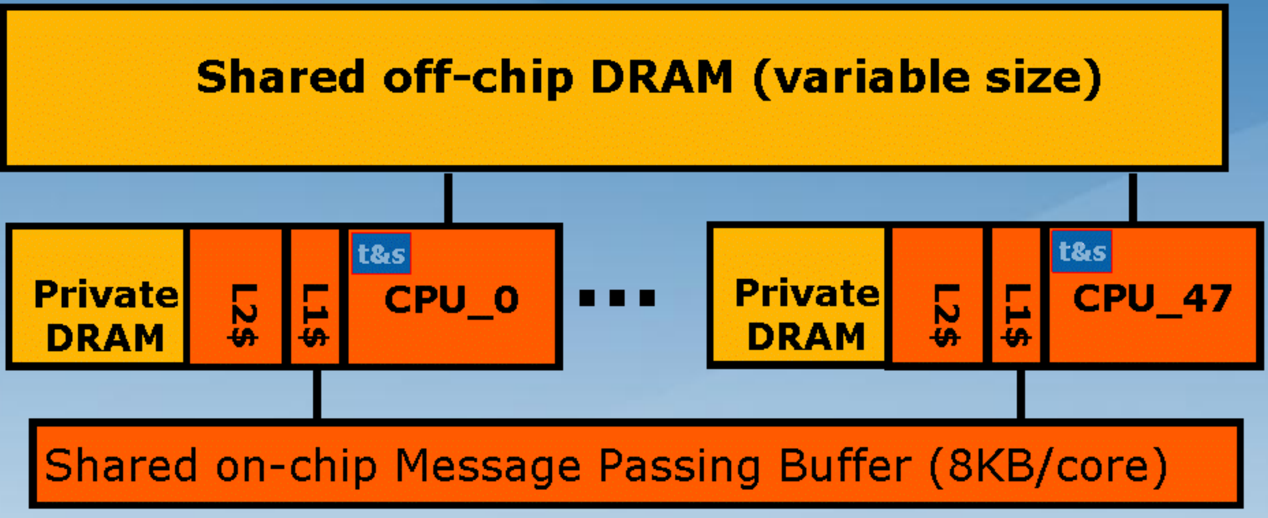


High level view of RCCE

- RCCE is a compact, lightweight communication environment.
  - SCC and RCCE were designed together side by side:
    - … a true HW/SW co-design project.
- RCCE is a research vehicle to understand how message passing APIs map onto many core chips.
- RCCE is for experienced parallel programmers willing to work close to the hardware.
- RCCE Execution Model:
  - Static SPMD:
    - identical UEs created together when a program starts (this is a standard approach familiar to message passing programmers)

UE: Unit of Execution … a software entity that advances a program counter (e.g. process of thread).

How does RCCE work? Part 1



Message passing buffer memory is special … of type MPBT

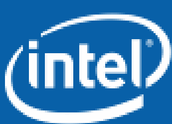
Cached in L1, L2 bypassed. Not coherent between cores

Data cached on read, not write. Single cycle op to invalidate all MPBT in L1 … Note this is not a flush

Consequences of MPBT properties:

- If data changed by another core and image still in L1, read returns stale data.
  - **Solution: Invalidate before read.**
- L1 has write-combining buffer; write incomplete line? expect trouble!
  - **Solution: Don't. Always push whole cache lines**
- If image of line to be written already in L1, write will not go to memory.
  - **Solution: Invalidate before write.**

Discourage user operations on data in MPB. Use only as a data movement area managed by RCCE … Invalidate early, invalidate often



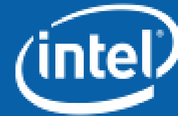
How does RCCE work? Part 2

- Treat Msg Pass Buf (MPB) as 48 smaller buffers … one per core.

Private DRAM | L2$ | L1$ | t&s | CPU_0 | … | Private DRAM | L2$ | L1$ | t&s | CPU_47

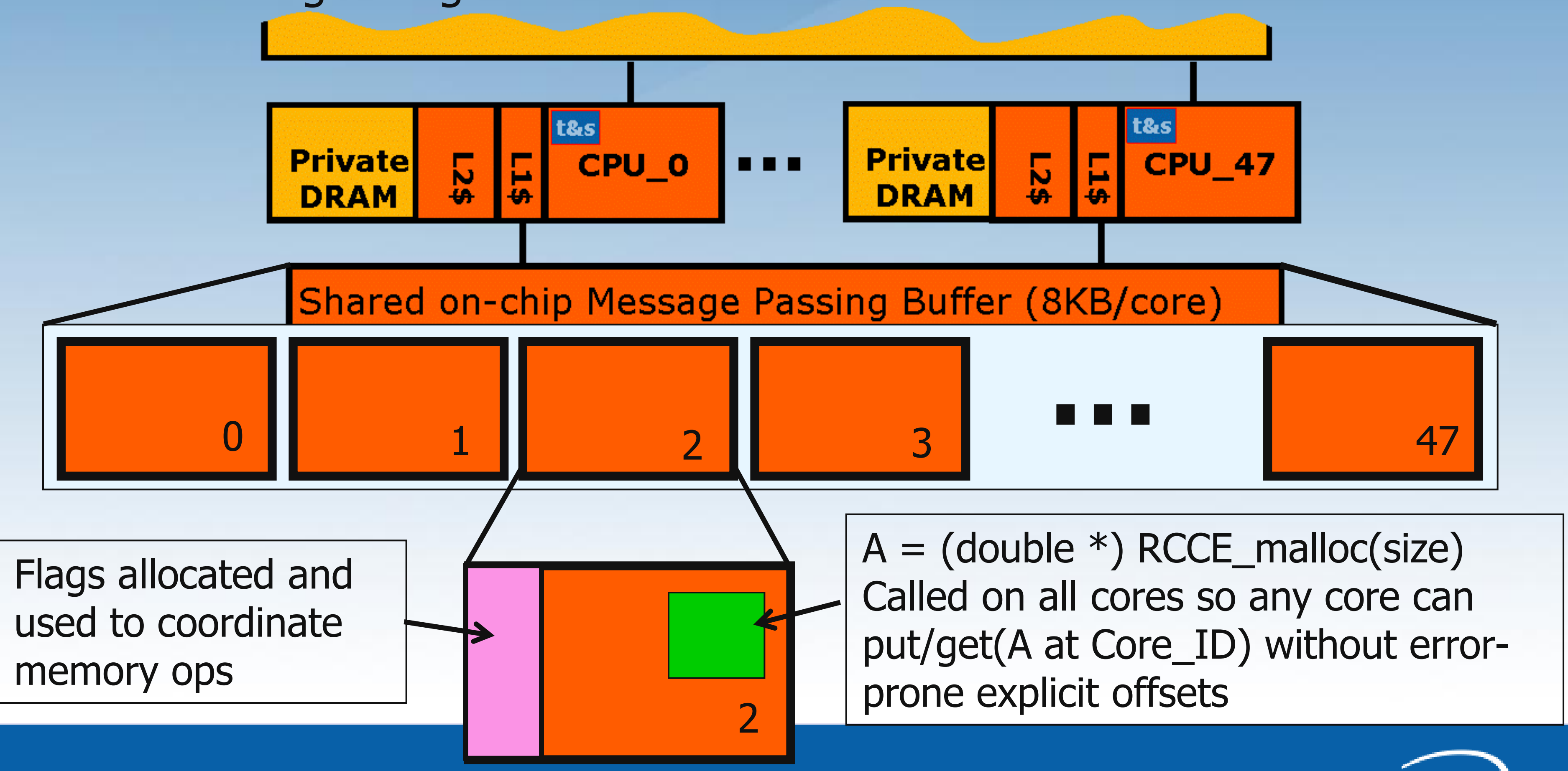
Shared on-chip Message Passing Buffer (8KB/core)

0 | 1 | 2 | 3 | … | 47



How does RCCE work? Part 2

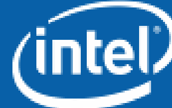
- Treat Msg Pass Buf (MPB) as 48 smaller buffers … one per core.
- Symmetric name space … Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.



How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- Symmetric name space … Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.

Private DRAM
L2$
L1$
t&s
CPU_0
…
Private DRAM
L2$
L1$
t&s
CPU_47
Private DRAM
L2$
L1$
t&s
CPU_0
Private DRAM
L2$
L1$
t&s
CPU_47
0
…
47



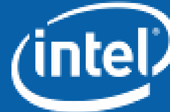
How does RCCE work? Part 3

- The foundation of RCCE is a one-sided put/get interface.
- Symmetric name space … Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.

How does RCCE work? Part 3

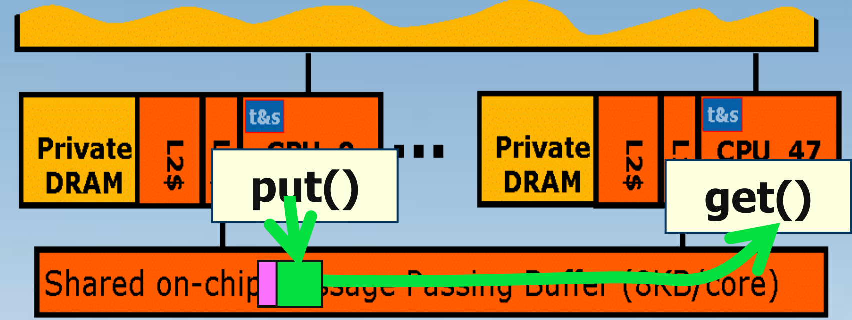
- The foundation of RCCE is a one-sided put/get interface.
- Symmetric name space … Allocate memory as a collective op. Each core gets a variable with the given name at a fixed offset from the beginning of a core's MPB.

… and use flags to make the puts and gets "safe"

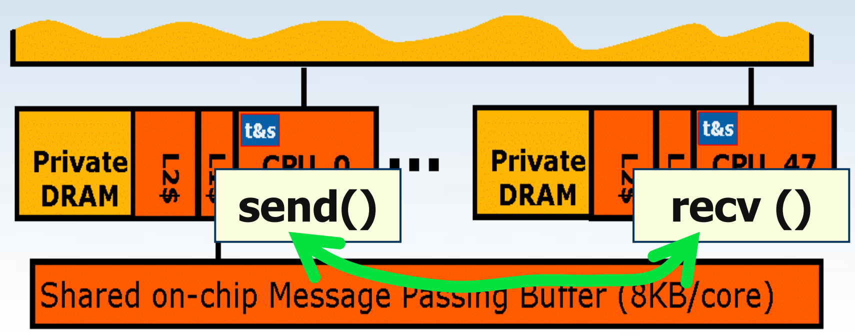


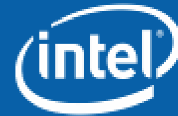
The RCCE library

- RCCE API provides the basic message passing functionality expected in a tiny communication library:
  - One + two sided interface (put/get + send/recv) with synchronization flags and MPB management exposed.
    - The "gory" interface for programmers who need the most detailed control over SCC



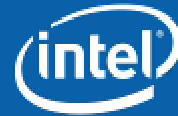
  - Two sided interface (send/recv) with most detail (flags and MPB management) hidden.
    - The "basic" interface for typical application programmers.





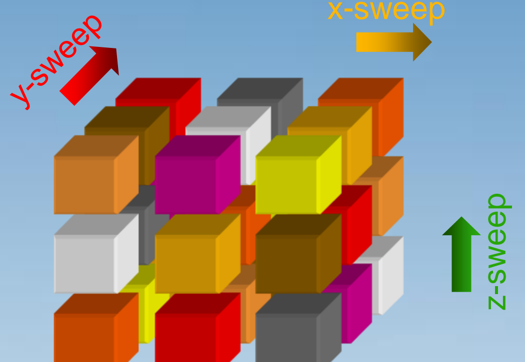
Agenda

- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management



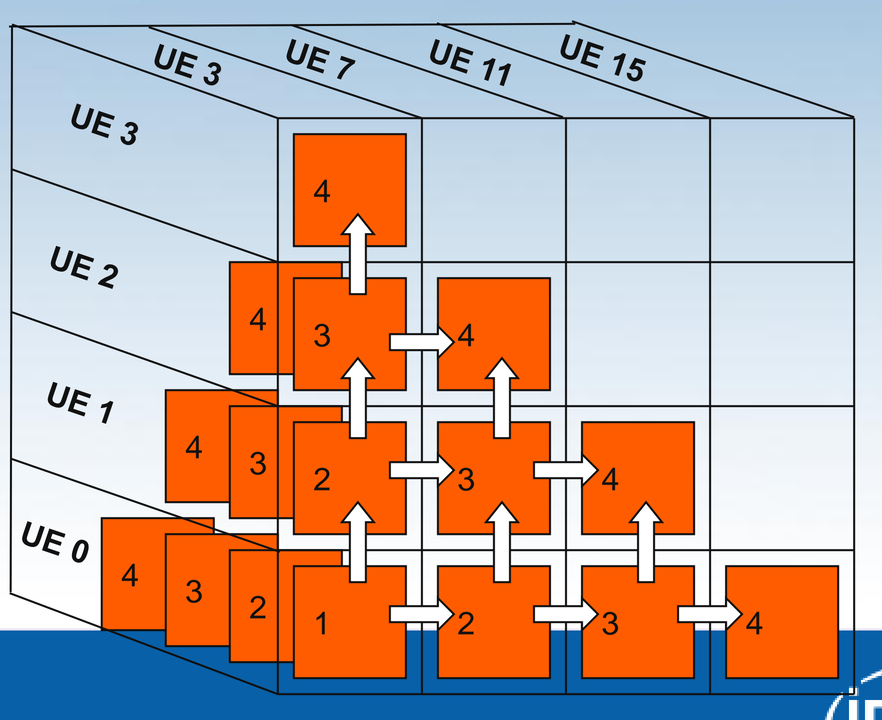
Linpack and NAS Parallel benchmarks

1. Linpack (HPL): solve dense system of linear equations
   – Synchronous comm. with “MPI wrappers” to simplify porting



2. BT: Multipartition decomposition
   – Each core owns multiple blocks (3 in this case)
   – update all blocks in plane of 3x3 blocks
   – send data to neighbor blocks in next plane
   – update next plane of 3x3 blocks

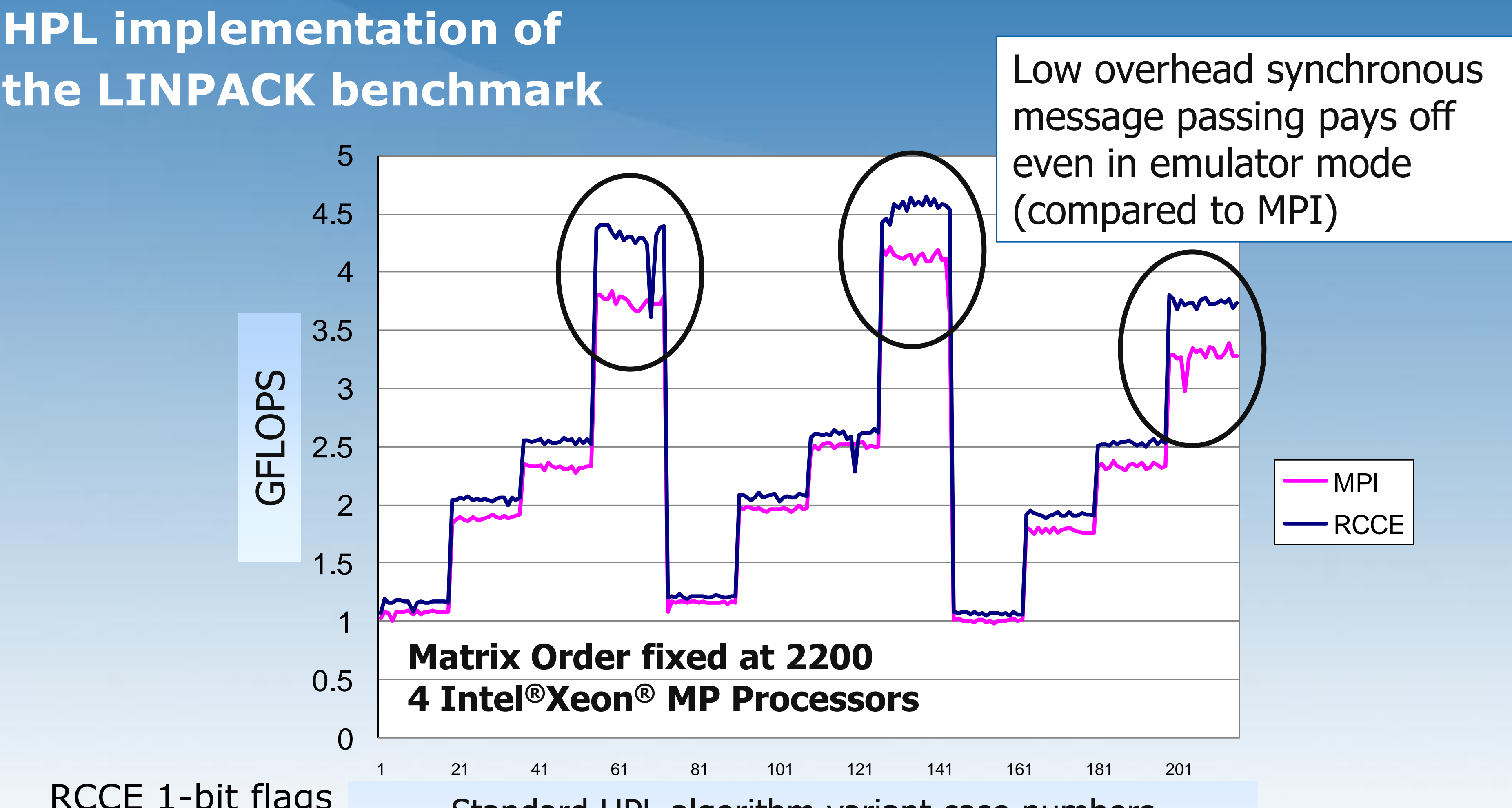
3. LU: Pencil decomposition
   Define 2D-pipeline process
   – await data (bottom+left)
   – compute new tile
   – send data (top+right)



Third party names are the property of their owners.

RCCE functional emulator vs. MPI

HPL implementation of the LINPACK benchmark



These results provide a comparison of RCCE and MPI on an older 4 processor Intel® Xeon® MP SMP platform* using a tiny 4x4 block size. These are not official MP-LINPACK results.

*3 GHz Intel® Xeon® MP processor in a 4 socket SMP platform (4 cores total), L2=1MB, L3=8MB, Intel® icc 10.1 compiler, Intel® MPI 2.0

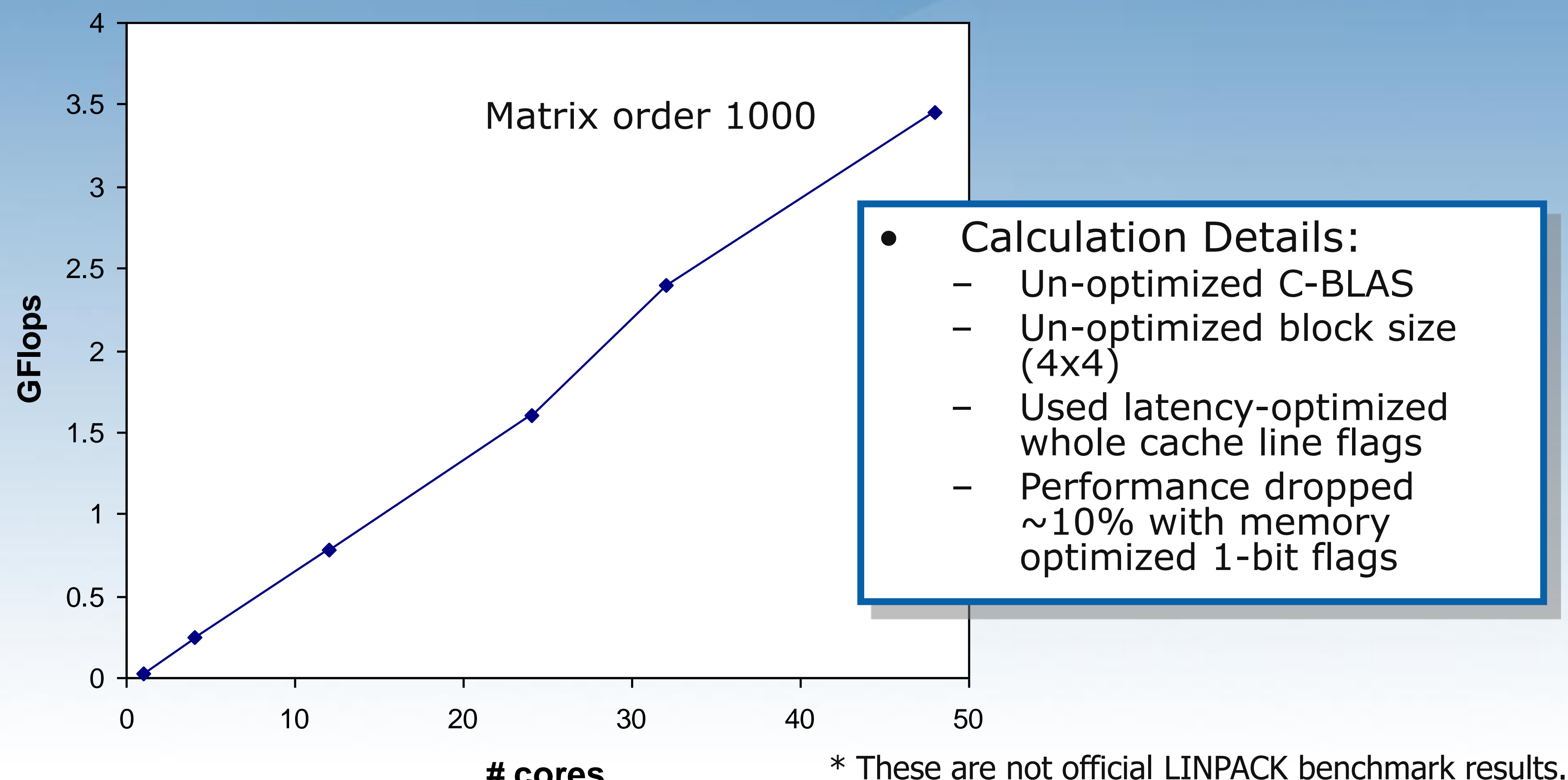
Third party names are the property of their owners.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

intel

Linpack, on the Linux SCC platform

- Linpack (HPL)* strong scaling results:
  - GFLOPS vs. # of cores for a fixed size problem (1000).
  - This is a tough test … scaling is easier for large problems.



* These are not official LINPACK benchmark results.

SCC processor 500MHz core, 1GHz routers, 25MHz system interface, and DDR3 memory at 800 MHz.

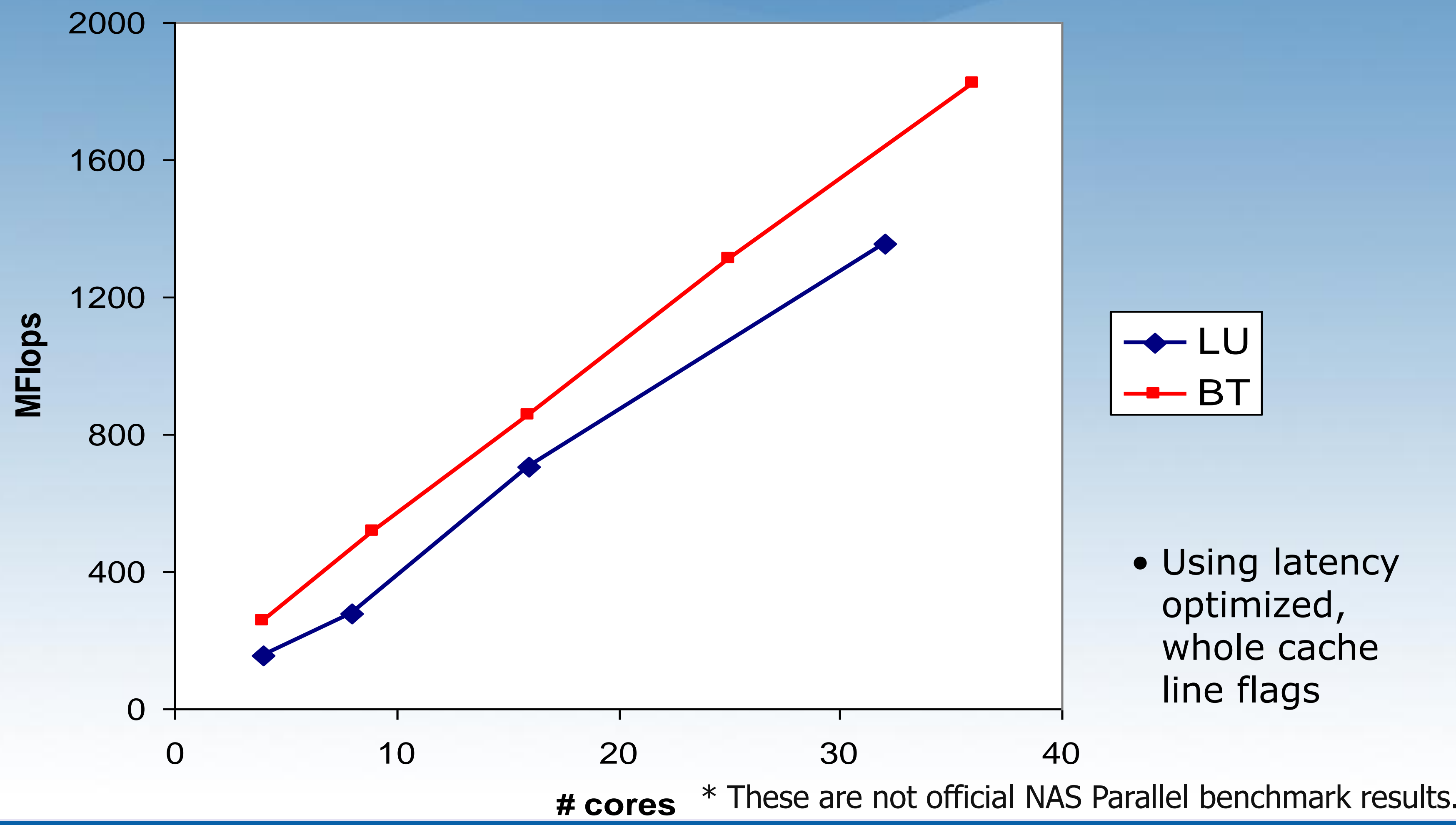
Third party names are the property of their owners.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

intel

LU/BT NAS Parallel Benchmarks, SCC

Problem size: Class A, 64 x 64 x 64 grid*



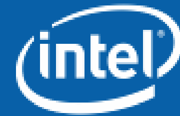
SCC processor 500MHz core, 1GHz routers, 25MHz system interface, and DDR3 memory at 800 MHz.

Third party names are the property of their owners.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference <http://www.intel.com/performance> or call (U.S.) 1-800-628-8686 or 1-916-356-3104.

Agenda

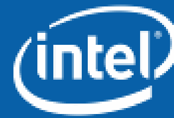
- Views of SCC: HW, SW and Platforms
- RCCE: A communication environment for application programmers.
- Benchmarks and Results
- Power management



RCCE Power Management API

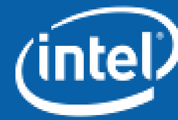
- RCCE power management emphasizes safe control: V/GHz changed together within each 4-tile (8-core) power domain.
  - A Master core sets V + GHz for all cores in domain.
    - RCCE_istep_power():
      - steps up or down V + GHz, where GHz is max for selected voltage.
    - RCCE_wait_power():
      - returns when power change is done
    - RCCE_step_frequency():
      - steps up or down only GHz

- Power management latencies
  - V changes: Very high latency, *O*(Million) cycles.
  - GHz changes: Low latency, *O*(few) cycles.



Conclusions

- RCCE software works
  - RCCE's restrictions (Symmetric MPB memory model and blocking communications) have not been a fundamental obstacle
  - Functional emulator is a useful development/debug device
- SCC architecture
  - The on-chip MPB was effective for scalable message passing applications
  - Software controlled power management works … but it's challenging to use because (1) granularity of 8 cores and (2) high latencies for voltage changes
  - The Test&set registers (only one per core) will be a bottleneck.
    - Sure wish we had asked for more!
- Future work
  - Add shmalloc() to expose shared off-chip DRAMM (in progress).
  - Move resource management into OS/drivers so multiple apps can work together safely.
  - We have only just begun to explore power management capabilities … we need to explore additional usage models.



SW Acknowledgements

- SCC System software:

Management Console software	Michael Riepen
BareMetalC workflow	Michael Riepen
Linux for SCC	Thomas Lehnig Paul Brett
System Interface FPGA development	Matthias Steidl
TCP/IP network driver	Werner Haas

- SCC Application software:

RCCE library and apps	Rob Van der Wijngaart Tim Mattson
Developer tools (Intel compilers and math libraries)	Patrick Kennedy
Mandelbrot app + visualization	Michael Riepen

intel

Backup slides

- Power Management
- Using RCCE and example RCCE code
- Additional RCCE implementation details
- RCCE and the MPI programmer
- SSC Literature reference

Power and memory-controller domains

Memory

Voltage

RC package

MC

Bus to PCI

Frequency

Power ~ F V^2

- Power Control domains (RPC):
  - 7 voltage domains … 6 4-tile blocks and one for on-die network.
  - 1 clock divider register per tile (i.e. 24 frequency domains)
  - One RPC register so can process only one voltage request at a time; other requestors block

RCCE Power Management API

- RCCE power management emphasizes safe control: V/GHz changed together within each 4-tile (8-core) power domain.
  - A Master core sets V + GHz for all cores in domain.
    - RCCE_istep_power():
      - steps up or down V + GHz, where GHz is max for selected voltage.
    - RCCE_wait_power():
      - returns when power change is done
    - RCCE_step_frequency():
      - steps up or down only GHz
- Power management latencies
  - V changes: Very high latency, *O*(Million) cycles.
  - GHz changes: Low latency, *O*(few) cycles.

Power management test

- A three-tier master-worker hierarchy,
  - one overall master, one team-lead per power domain, Team-members (cores) to do the work.
- Workload: A stencil computation to solve a PDE.

scc_eco_q.wmv - VLC media player

scc_pm - Mozilla Firefox

SCC Power Management

intel

Advanced Workload Aware Power Management Technology

Fine-grained dynamic frequency and voltage Control

Power Management

OFF ON

121 Watts

jmhoward on mrllab1000: /shared/DEMOS/ECO_Q - Shell - Konsole

```
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/> /shared/DEMOS/ECO_Q/new_pwr_app.exe
/shared/DEMOS/ECO_Q/new_pwr_app.exe
Successfully opened RCKMEM driver devices!
My rank is 47, physical core ID is 47
UE 47, Core ID 47; size of V dom 5 is 8, F dom 23 is 2
```

Rock Creek performance meter

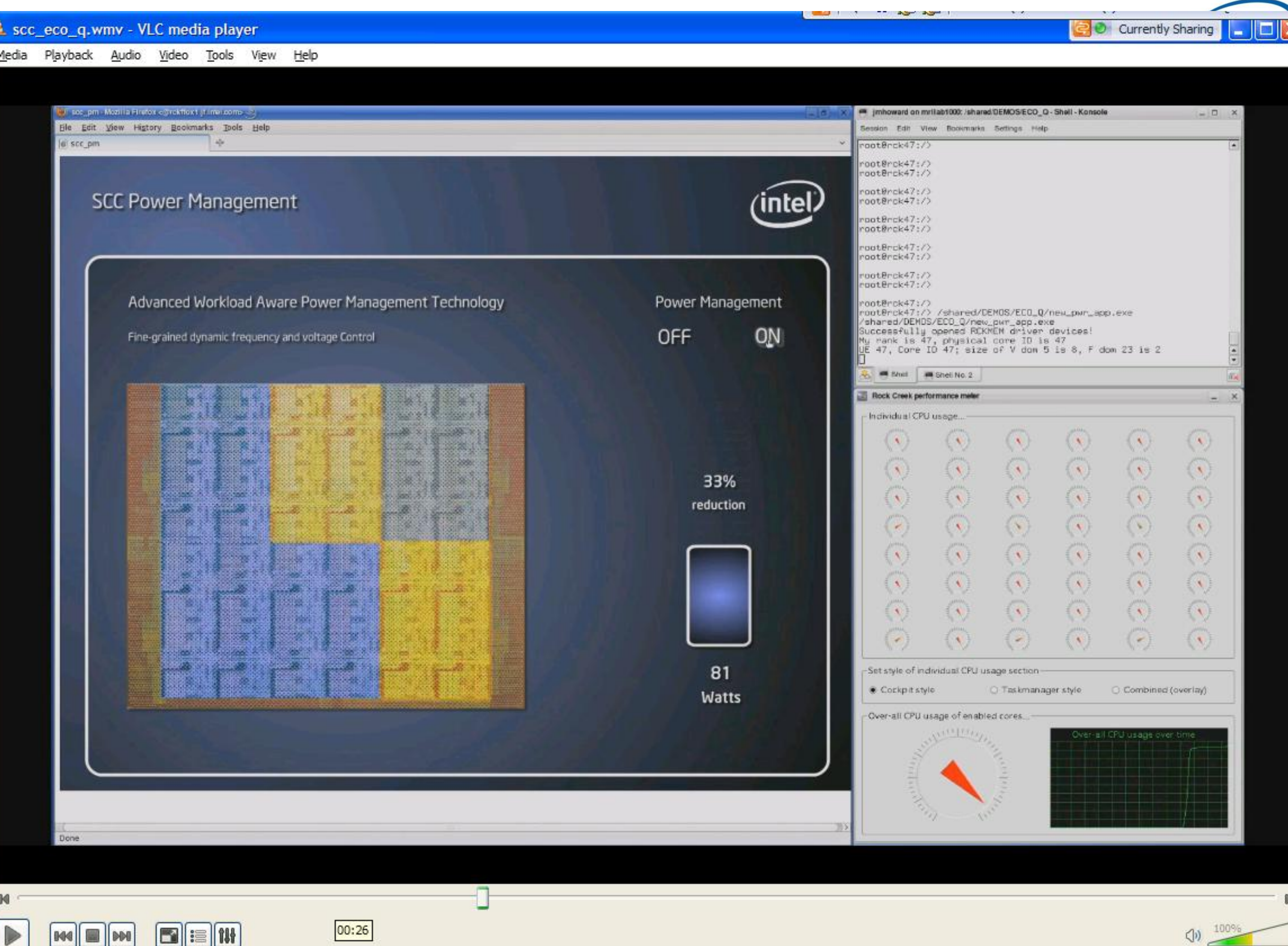
Individual CPU usage...

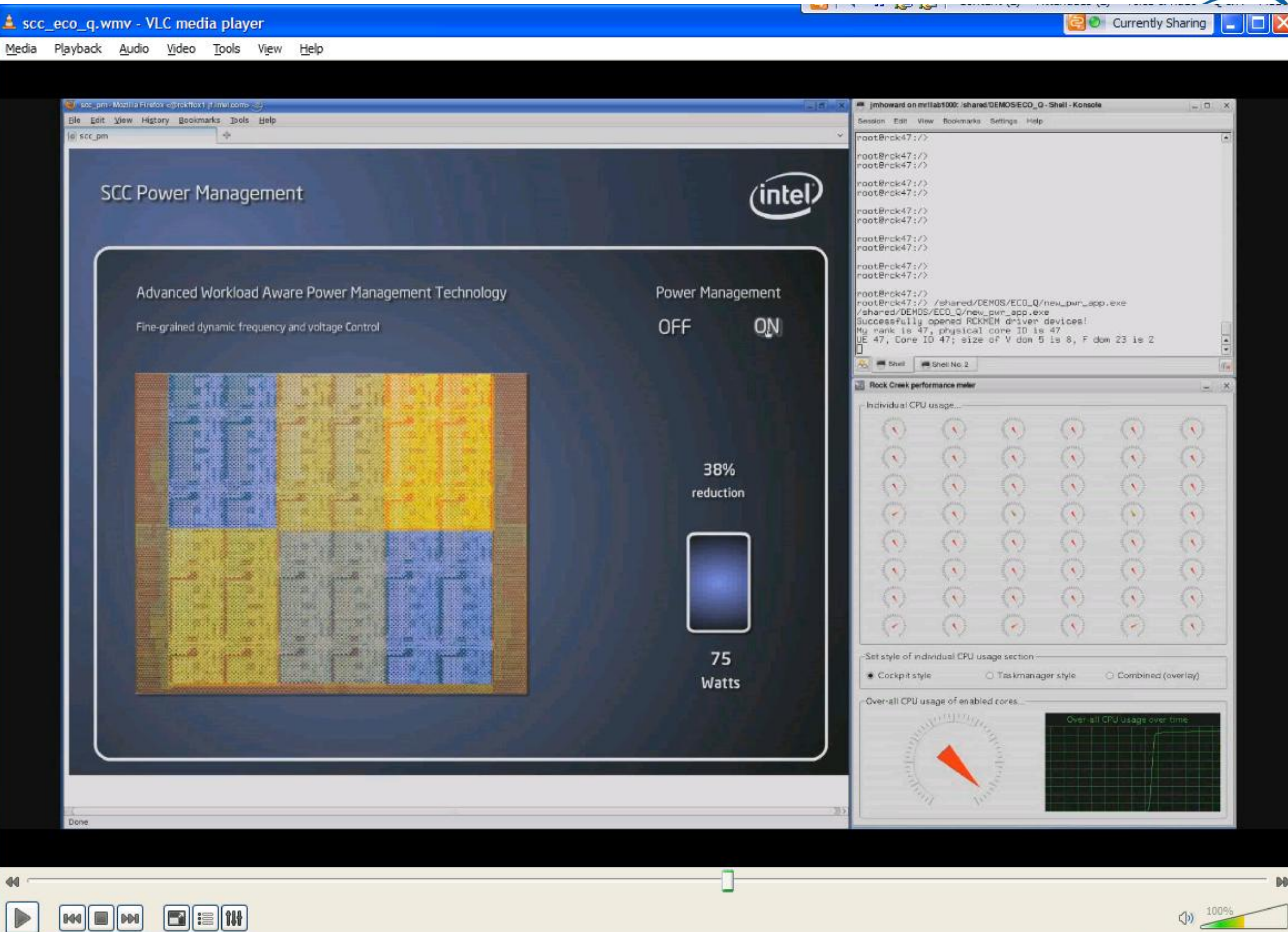
Set style of individual CPU usage section

Cockpit style ○ Taskmanager style ○ Combined (overlay)

Over-all CPU usage of enabled cores...

Over-all CPU usage over time


scc_eco_q.wmv - VLC media player
Media Playback Audio Video Tools View Help
SCC Power Management
intel
Advanced Workload Aware Power Management Technology
Fine-grained dynamic frequency and voltage Control
Power Management
OFF
ON
33%
reduction
81
Watts
Rock Creek performance meter
Individual CPU usage
Set style of individual CPU usage section
Cockpit style
Taskmanager style
Combined (overlay)
Over-all CPU usage of enabled cores
root@rck47:/> /shared/DEMOS/ECO_Q/new_pwr_app.exe
/shared/DEMOS/ECO_Q/new_pwr_app.exe
Successfully opened RCKMEM driver devices!
My rank is 47, physical core ID is 47
UE 47, Core ID 47; size of V dom 5 is 8, F dom 23 is 2
Currently Sharing
00:26
scc_eco_q.wmv
1.00x
00:39/01:47


scc_eco_q.wmv - VLC media player
Currently Sharing
Media
Playback
Audio
Video
Tools
View
Help
SCC Power Management
intel
Advanced Workload Aware Power Management Technology
Fine-grained dynamic frequency and voltage Control
Power Management
OFF
ON
38%
reduction
75
Watts
Rock Creek performance meter
Individual CPU usage
Set style of individual CPU usage section
Cockpit style
Taskmanager style
Combined (overlay)
Over-all CPU usage of enabled cores
Over-all CPU usage over time
root@rck47:/> /shared/DEMOS/ECO_Q/new_pwr_app.exe
/shared/DEMOS/ECO_Q/new_pwr_app.exe
Successfully opened RCKMEM driver devices!
My rank is 47, physical core ID is 47
UE 47, Core ID 47; size of V dom 5 is 8, F dom 23 is 2
scc_eco_q.wmv
1.00x
01:00/01:47

scc_eco_q.wmv - VLC media player

Currently Sharing

Media Playback Audio Video Tools View Help

scc_pm - Mozilla Firefox

File Edit View History Bookmarks Tools Help

scc_pm

SCC Power Management

intel

Advanced Workload Aware Power Management Technology

Fine-grained dynamic frequency and voltage Control

Power Management

OFF ON

74%
reduction

31
Watts

Done

jmhoward on mrltab1000: /shared/DEMOS/ECO_Q - Shell - Konsole

Session Edit View Bookmarks Settings Help

```
root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/>

root@rck47:/>
root@rck47:/> /shared/DEMOS/ECO_Q/new_pwr_app.exe
/shared/DEMOS/ECO_Q/new_pwr_app.exe
Successfully opened RCKMEM driver devices!
My rank is 47, physical core ID is 47
UE 47, Core ID 47; size of V dom 5 is 8, F dom 23 is 2
root@rck47:/>
root@rck47:/>
```

Shell Shell No. 2

Rock Creek performance meter

Individual CPU usage

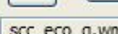
Set style of individual CPU usage section

Cockpit style Taskmanager style Combined (overlay)

Over-all CPU usage of enabled cores

Over-all CPU usage over time

100%



scc_eco_q.wmv 1.00x 01:24/01:47

Backup slides

- Power Management
- Using RCCE and example RCCE code
- Additional RCCE implementation details
- RCCE and the MPI programmer
- SSC Literature reference

RCCE API: Writing and running RCCE programs

(intel)

- We provide two interfaces for the RCCE programmer:
  - **Basic Interface** (general purpose programmers):
    - FLAGS and Message Passing Buffer memory management hidden from the programmer.
  - **Gory interface** (hard core performance programmers):
    - One sided and two sided
    - Message Passing Buffer management is explicit
    - Flags allocated and managed by programmer.
- Build you job linking to the appropriate RCCE library, then run with rccerun

```
rccerun –nue N [optional params] program[params]
```

– **`program`** executes on N UEs as if it were invoked as: "program params" (no parameters allowed for Baremetal)

– Optional parameters

- ➢ -f hostfile: lists physical core IDs available to execute code
- ➢ -emulator: run on functional emulator

RCCE API: Circular Shift one sided

```
#include "RCCE.h"
 int RCCE_APP() {

  RCCE_init(&argc, &argv);
  NUES = RCCE_num_ues();
  ID = RCCE_ue();

  ID_right = (ID+1)%NUES;
  ID_left = (ID-1+NUES)%NUES;
  size = BUFSIZE*sizeof(double);
  buffer  = (double *) malloc(size);
  cbuffer = (double *) RCCE_malloc(size);

/* create and initialize flag variables */
   RCCE_flag_alloc(&flag_sent);
   RCCE_flag_alloc(&flag_ack);
   RCCE_flag_write(&flag_sent,
          RCCE_FLAG_UNSET, ID))
   RCCE_flag_write(&flag_ack,
          RCCE_FLAG_SET, ID_left))
```

```
for (int round=0; round<nrounds; round++) {

    RCCE_wait_until(flag_ack, RCCE_FLAG_SET);
    RCCE_flag_write(&flag_ack,
                      RCCE_FLAG_UNSET, ID);
    RCCE_put(cbuffer, buffer, size, ID_right);
    RCCE_flag_write(&flag_sent,
                      RCCE_FLAG_SET, ID_left);

    RCCE_wait_until(flag_sent,
                      RCCE_FLAG_SET);
    RCCE_flag_write(&flag_sent,
                      RCCE_FLAG_UNSET, ID);
    RCCE_get(buffer, cbuffer, size, ID);
    RCCE_flag_write(&flag_ack,
                 RCCE_FLAG_SET, ID_left);
}
```

BUFSIZE must be divisible by 4
Message must fit inside Msg Buff

RCCE API: Circular Shift one-sided

```
#include "RCCE.h"
 int RCCE_APP() {

  RCCE_init(&argc, &argv);
```

RCCE_FLAG flg;

RCCE_flag_alloc(&flg);

RCCE_flag_set(flg, RCCE_FLAG_SET, ID); ***or RCCE_FLAG_UNSET***

RCCE_wait_until(flg, RCCE_FLAG_SET,ID); ***or RCCE_FLAG_UNSET***

RCCE_put(cbuffer, buffer, size, ID);

Put my private memory (buffer) into the msg buffer (cbuffer) of core ID

RCCE_get(buffer, cbuffer, size, ID));

Get cbuffer from core ID and move it into my private memory (buffer)

```
   RCCE_flag_write(&flag_sent,
          RCCE_FLAG_UNSET, ID))
   RCCE_flag_write(&flag_ack,
          RCCE_FLAG_SET, ID_left))
```

```
for (int round=0; round<nrounds; round++) {

    RCCE_wait_until(flag_ack, RCCE_FLAG_SET);
    RCCE_flag_write(&flag_ack,
}
```

BUFSIZE must be divisible by 4
Message must fit inside Msg Buff

RCCE API: “Basic” interface, two sided

```
RCCE_wait_until(flag_ack, RCCE_FLAG_SET);
RCCE_flag_write(&flag_ack,
                    RCCE_FLAG_UNSET, ID);
RCCE_put(cbuffer, buffer, size, ID_right);
RCCE_flag_write(&flag_sent,
                    RCCE_FLAG_SET, ID_left);
```

- flags needed to make transfers safe.
- Large messages must be broken up to fit into the Msg Buff.

- We can hide these details by letting library manage flags +MPB:

RCCE_send(buffer, size, ID); *Send private memory (buffer) to core ID*
RCCE_recv(buffer, size, ID)); *Receive into private memory (buffer) from core ID*

- This is Synchronous message passing … the send and receive do not return until the communication is complete on both sides.

RCCE API: Circular Shift with 2-sided Basic interface

```
#include <string.h>
#include "RCCE.h"
int RCCE_APP() {

  RCCE_init(&argc, &argv);
  NUES = RCCE_num_ues();

  ID = RCCE_ue();

  ID_right  = (ID+1)%NUES;
  ID_left  = (ID-1+NUES)%NUES;
  int size  = BUFSIZE*sizeof(double);
  buffer   = (double *) malloc (size);
  buffer2 = (double *) malloc (size);
```

```
for (int round=0; round<nrounds; round++) {

  for (int c = 0; c<2; c++) {
      if ((ID+c)%2)
           RCCE_send(buffer, size, ID_right);
      else
           RCCE_recv(buffer2, size, ID_left);
   }
   memcpy(buffer, buffer2, size);
}
```

Hides buffer and flag allocation, messages "packetizing", and flag synchronization.

Anticipate most programmers will use this RCCE version

BUFSIZE may be anything
Message need not fit inside Msg Buf

Backup slides

- Power Management
- Using RCCE and example RCCE code
- Additional RCCE implementation details
- RCCE and the MPI programmer
- SSC Literature reference

RCCE Implementation details:

One-sided message passing; safely but blindly transport data between private memories

```
RCCE_put(char *target, char *source, size_t size, int ID)
{
  target = target + (RCCE_MPB[ID]-RCCE_MPB[RCCE_IAM]);
  RCCE_cache_invalidate();
  memcpy(target, source, size);
}
RCCE_get(char *target, char *source, size_t size, int ID)
{
  source = source + (RCCE_MPB[ID]-RCCE_MPB[RCCE_IAM]);
  RCCE_cache_invalidate();
  memcpy(target, source, size);
}
```

offsets to “remote” MPB

RCCE_MPB[ID] = start of MPB for UE “ID”
RCCE_IAM = library shorthand for calling UE
target/source cache line aligned, size%32=0, data fits inside MPB

RCCE Implementation details:

Two-sided message passing; safely transport data between private memories, with handshake.

```
RCCE_send(char *privbuf, char *combuf, RCCE_FLAG *ready,
          RCCE_FLAG *sent, size_t size, int dest) {
  RCCE_put(combuf, privbuf, size, RCCE_IAM);
  RCCE_flag_write(sent, SET, dest);
  RCCE_wait_until(*ready, SET);
  RCCE_flag_write(ready, UNSET, RCCE_IAM);}
```

HANDSHAKES
sent, ready: synchronization flags stored in MPB

```
RCCE_recv(char *privbuf, char *combuf, RCCE_FLAG *ready,
          RCCE_FLAG *sent, size_t size, int source) {
  RCCE_wait_until(*sent, SET);
  RCCE_flag_write(sent, UNSET, RCCE_IAM);
  RCCE_get(privbuf, combuf, size, source);
  RCCE_flag_write(ready, SET, source); }
```

- Body gets called in a loop (+ padding if necessary) for large messages
- send and recv asymmetric: needed to avoid deadlock
- No size or alignment restrictions
- We get rid of these parameters in our “basic” interface (≈MPI)

RCCE Implementation Details:
Flags

- Flags implemented two ways
  1. whole MPB memory line (96 flags, 30% of MPB)
  2. single bit (1 MPB memory line for all flags)
     - Control write access through atomic test&set register, implementing lock.
     - No need to protect read access.

- Implications of the two types of flags:
  – Single bit saves MPB memory but you pay with a higher latency.
  – Whole cache line wastes memory but lowers latency.

RCCE Implementation Details:
RCCE flag write scenario (single bit)

```
void RCCE_flag_write(RCCE_FLAG *flag, RCCE_FLAG_STATUS val, int ID) {
  volatile unsigned char val_array[RCCE_LINE_SIZE];

  /* acquire lock so nobody else fiddles with the flags on the target core */
  RCCE_acquire_lock(ID);
  /* copy line containing flag to private memory                         */
  RCCE_get(val_array, flag->line_address, RCCE_LINE_SIZE, ID);
  /* write "val" into single bit corresponding to flag                   */
  RCCE_write_bit_value(val_array, flag->location, val);
  /* copy line back to MPB                                               */
  RCCE_put(flag->line_address, val_array, RCCE_LINE_SIZE, ID);
  /* release write lock for the flags on the target core */
  RCCE_release_lock(ID);
}
void RCCE_acquire_lock(int ID) {
  while (!((*(physical_lockaddress[ID])) & 0x01));
}
void RCCE_release_lock(int ID) {
*(physical_lockaddress[ID]) = 0x0;
}
```

physical_lockaddress[ID]: address of test&set register on core with rank ID.
RCCE_flag_read does not need lock protection.

Backup slides

- Power Management
- Using RCCE and example RCCE code
- Additional RCCE implementation details
- RCCE and the MPI programmer
- SSC Literature reference

RCCE vs MPI

- No opaque data types in RCCE, so no MPI-style handles, only pointers
- No RCCE_datatype, except for reductions
- No communicators, except in collective communications
- Only synchronous communications
  - + No message bookkeeping
  - – No overlap of computations/communications
  - – Deadlock?
- RCCE has low overhead due short communication stack:
  - RCCE_send→RCCE_put→memcpy

RCCE vs MPI: Avoiding deadlock

- If sending and receiving UE sets overlap, deadlock is possible. Cause: cycles in communication graph (cyclic dependence).
- If no cycles, communication may serialize
- Solution:
  - Divide communication pattern into disjoint send-receive UE sets (bipartite graphs), execute in phases.
  - Number of phases depends on pattern.
  - For permutation pattern, two phases min, three max:
    1. Each permutation can be divided into cycles (length L)
    2. If L even, red/black coloring suffices.
    3. If L odd (2n+1), apply 2. to 2n UEs, then finish communications for last UE. Each cycle takes *O*(1) time.
- Note: coloring is wrt position in cycle, not UE rank; may need different phase colorings for different patterns.

RCCE vs MPI: Avoiding deadlock

you are here

Programmer just posts (i)sends and (i)receives as needed

recv send

MPI

you are here

send recv

RCCE

Programmer must pair all sends and receives

RCCE vs MPI: Avoiding deadlock

– pseudo-code example from HPC application:

```
MPI:  if (!IAM_LEFTMOST) {
          MPI_irecv(from_left);
          MPI_wait(on_isend);
          MPI_wait(on_irecv);
      }
      compute;

      if (!IAM_RIGHTMOST) MPI_isend(to_right);
```

```
RCCE: if (!IAM_LEFTMOST)
       for (phase = 0; phase < 3; phase++) {
          if (send_color==phase) RCCE_send(to_right);
          if (recv_color==phase) RCCE_recv(from_left);
       }
       compute;
```

– Notes:

- MPI version cell based; RCCE version interface based
- RCCE fairly easy to grok, but requires restructuring to interleave sends/recvs

Backup slides

- Power Management
- Using RCCE and example RCCE code
- Additional RCCE implementation details
- RCCE and the MPI programmer
- SSC Literature reference

Official SSC reference

"A 48-Core IA-32 Message Passing Processor with DVFS in 45nm CMOS", ISSCC 2010.

J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, T. Mattson

- **Abstract:**
  - A 567mm2 processor in 45nm CMOS integrates 48 IA-32 cores and 4 DDR3 channels in a 6×4 2D-mesh network. Cores communicate through message passing using 384KB of on-die shared memory. Fine grain power management takes advantage of 8 voltage and 28 frequency islands to allow independent DVFS of cores and mesh. As performance scales, the processor dissipates between 25W and 125W.