Integrated SW/ HW solutions for Network On Chip based Chip Multi-Processing challenges
Overview

- The problem with thread communication
- Integrated SW/HW solutions
  - Improving specific communication for general threads
  - Improving general communication for specific threads
- Adding affinity for private caches
- Conclusion
Thread comm. via shared memory

- Thread A. grabs a lock
  - cache line holding lock invalidated on another tile and fetched
- Thread B. tries to grab lock
  - Another cache fetch. In the good case, cache line will still be owned by thread A's tile.
- Thread A. updates buffer
  - empty buffer cache lines move to the calling tile
- Thread A. releases lock
  - An update to the cache line
- Thread B. grabs lock
  - Cache line travels again
- Thread B. reads data
Thread comm. via ports / pipes

- A natural way to describe communication
- Use several locks
- The data must pass through one or more OS buffers
- Good chance data will run to and from another tile running the OS service
Solutions

- Decrease the amount of synchronization needed
  - Algorithmic problem (with trade-offs)
- Decrease the cost of synchronization across the system
  - HW problem (with trade-offs)
- Accelerate specific forms of synchronization
  - Synchronization variables, communication
- Accelerate synchronization between specific threads
  - Thread and data scheduling
What are we looking for?

- SW should be independent of specific chip "revision". (e.g. size of N.O.C, wire speed)
- Use as much existing software as possible. from most to least important: programing model, application code, application binaries, libraries, OS
- Keep HW simple
- Keep power and latency minimal, especially "normal" transactions which do not "use" the coherency mechanism
Shared L2 cache

• Decrease the cost of synchronization across the system
  – We still have private L1 caches!
• Increases interference between unrelated threads
• Increase the cost of “normal” accesses to data
  – Usually this is the big majority.
  – Effect grows with the size of the N.O.C
• Another solution: shared cache for shared data only (NAHALAL)
  – still not scalable
Accelerating specific forms of synchronization for general threads
Common synchronization

- In common program models, communication between tasks takes two basic forms
  - Synchronization variables – carry very little data, need to be distributed efficiently amongst all tiles.
  - Data communication – moving actual data between threads. Normally involves clearing data off producers output buffer and filling it into consumer's input buffer.
- The good news: applications usually invoke OS or library calls to get that job done.
Efficient synchronization

- Handle synchronization variables inside the memory controller
- Synchronization variables are not cached

Efficient Synchronization for Embedded On-Chip Multiprocessors
Matteo Monchiero, Gianluca Palermo, Cristina Silvano, Oreste Villa, 2006
Efficient Synchronization

- The SB handles both spin-locks and events
- n entries are kept. n is "not large" since the system only uses few synchronization variables at the same time.
- A queue of maximum size P (number of processors) * T (number of simultaneous threads per processor) is kept for each entry.
**Efficient inter-thread communication**

- Scratchpad memory + DMA can be used to communicate between threads

![Diagram](image.png)

*Figure 1: Hardware extensions for scratch-management: (left) original-(right) extended*

An Integrated Hardware/Software Approach For Run-Time Scratchpad Management
Poletti Francesco, Paul Marchal, David Atienza, Luca Benini, Francky Catthoor, Jose M. Mendias, 2004
Adapting SW

Some changes to code for DMA mem handling

```c
TASK A

while(input){
    key[i] = malloc(4*32);
    ...
}

TASK B

int X[N*N], Y[N*N], Z[N*N];
int i,j,k;

for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
        Z[i*N+j] = 0;
        for (int k=0; k<N; k++)
            Z[i*N+j] +=
            X[i*N+k] * Y[j+k*N]; /* local memory */
    }
}

OS-boot

SMcreateManager(scratch,2kB,
               region,Manager1)

TASK A

while(input){
    key[i] = SMMalloc(4*32, Manager1);
    ...
}

TASK B

int X[N*N], Y[N*N], Z[N*N];
int i,j,k;

for (int i=0; i<N; i++) {
    for (int j=0; j<N; j++) {
        DMAM2add((uint)((int*)X+i*N),object[0]);
        DMAnewstate('to',object[0],true);
        Z[i*N+j] = 0;
        for (int k=0; k<N; k++)
            Z[i*N+j]+= X_{lr}[k] * Y[j+k*N];
    }
}
```

Figure 2: Motivational example: original code (left), integrated approach (right)
Adapting SW

- implement a flexible message passing library using MPI.
- OpenMP is possible, but compiler dependent.

Flexible hardware/software support for message passing on a distributed shared memory architecture

```
PROGRAM bcast
INCLUDE 'mpif.h'
INTEGER imsg(4)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
IF (myrank==0) THEN
  DO i=1,4
    imsg(i) = i
  ENDDO
ELSE
  DO i=1,4
    imsg(i) = 0
  ENDDO
ENDIF
PRINT *, 'Before:', imsg
CALL MP_FLUSH(1)
CALL MPI_BCAST(imsg, 4, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
PRINT *, 'After :', imsg
CALL MPI_FINALIZE(ierr)
END
```
Integrated HW/SW Solutions - Attitude II

Accelerating general synchronization between specific threads
A word about thread scheduling

- Scheduling is a crucial component of correct N.O.C usage
- Scheduler must be aware of many parameters:
  - Communication with other threads
  - Private cache usage
  - I/O vs. CPU bound
  - types of tile most fitting for the task
    - high speed / low speed / reconfigurable
  - etc..
A word about thread scheduling

- Greedy isn't always the best way..

Incremental Run-time Application Mapping for Homogeneous NoCs with Multiple Voltage Levels, Chen-Ling Chou, Radu Marculescu, 2007
A word about thread scheduling

But how will putting two threads closer help?
**Shared Cache Affinity**

- Divide the shared cache on page by page basis
- Have OS choose how much cache each process deserves

Managing Distributed, Shared L2 Caches through OS-Level Page Allocation
Sangyeun Cho, Lei Jin, 2006
Shared Cache Affinity

- Applications / VMs will not interfere one with the other
  - Cache over-usage of one will not flush other entries
  - Less traffic hazards over the network
  - Shorter path for most memory requests
- Risk: Hot pages

Managing Distributed, Shared L2 Caches through OS-Level Page Allocation
Sangyeun Cho, Lei Jin, 2006
Private cache affinity

- Even when caches are private, the library is still a common resource
- The directory is staticly spreaded amongst the tiles
- Just like the case with shared caches, spreading the directory page-by-page will allow the OS to manage that resource.

Proximity-Aware Directory-based Coherence for Multi-core Processor Architectures
Dean Tullsen, Rakesh Kumar, Jeffery A. Brown, 2007
Private cache affinity

- Shorter control messages
  - Threads using specific data will be close to the directory.
- Fewer control messages
  - Thread most common for coherency message for some data can be placed on the same tile as it's directory (producer / consumer).
- Prevent directory overload on a certain tile
- Enjoying the benefits of private cache and of affinity.
Private cache affinity

- Most affect when switching between parallel and serial code.
- Directory for data moved should be placed in the serial tile.
- Inter-cache data transfer will only require 2 messages.
Some numbers..

• Up to 60% of private L2$ misses turn out to be cache-to-cache misses
• Turning a 3-hop miss that crosses the chip (4*4) into a 3-hop miss to a nearby node may save around 20% of the latency for that miss.
• Turning a 3-hop miss into a 2-hop miss may save about 40-50% of the latency for that miss.
• On some programs, switching all 3-hop misses into 2-hop misses can create overall speedups of over 1.25.

To Summerize...

General Threads
Specific communication

Synchronization Buffer

General communication
Specific Threads

Thread Scheduling

Scratchpad communication

Flow Related

Data Scheduling

Data Related
Conclusion

- Communication through memory will in some cases become inefficient – even between threads running on the same chip (sharing “static” information is fine).
- Aspects of programing models meant to communicate between process should be used to communicate between threads.
- HW must be aware of the way SW uses synchronization and focus on solving the “real” problems.
- Private L2$ affinity is worth exploring.