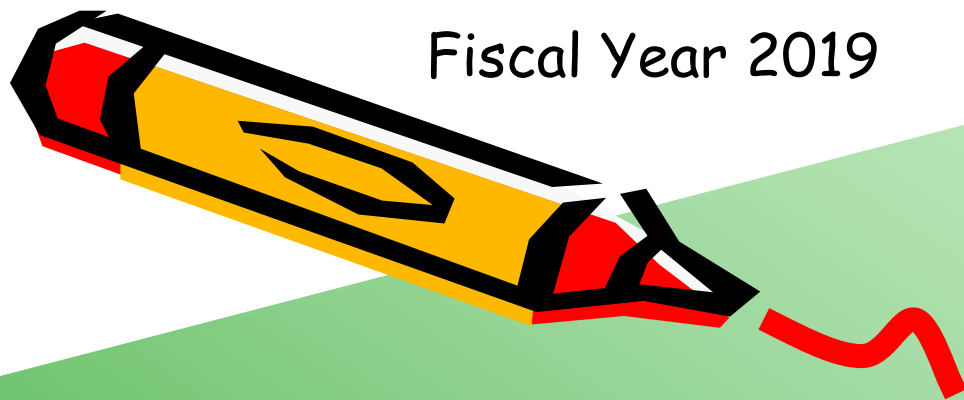


Fiscal Year 2019

Ver. 2020-01-24a



Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

13. Thread Level Parallelism: Coherence and Synchronization



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

Kenji Kise, Department of Computer Science
kise_at_c.titech.ac.jp

Final report

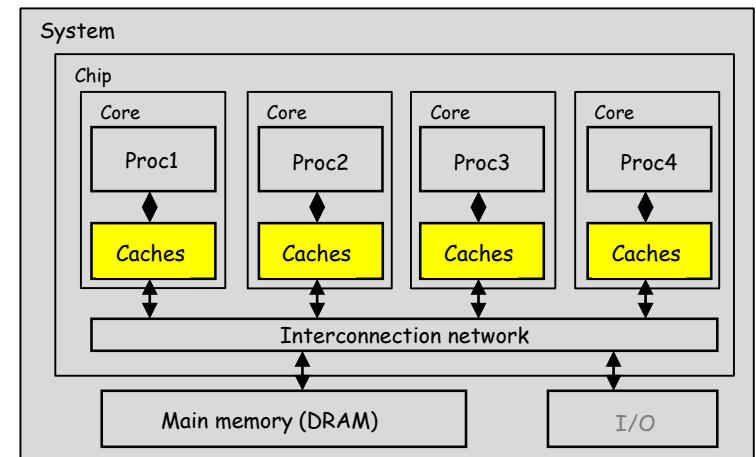


1. For details of the final report, please visit the lecture support page.
<http://www.arch.cs.titech.ac.jp/lecture/ACA>
2. Submit your final report in a PDF file via E-mail **by February 13, 2020**



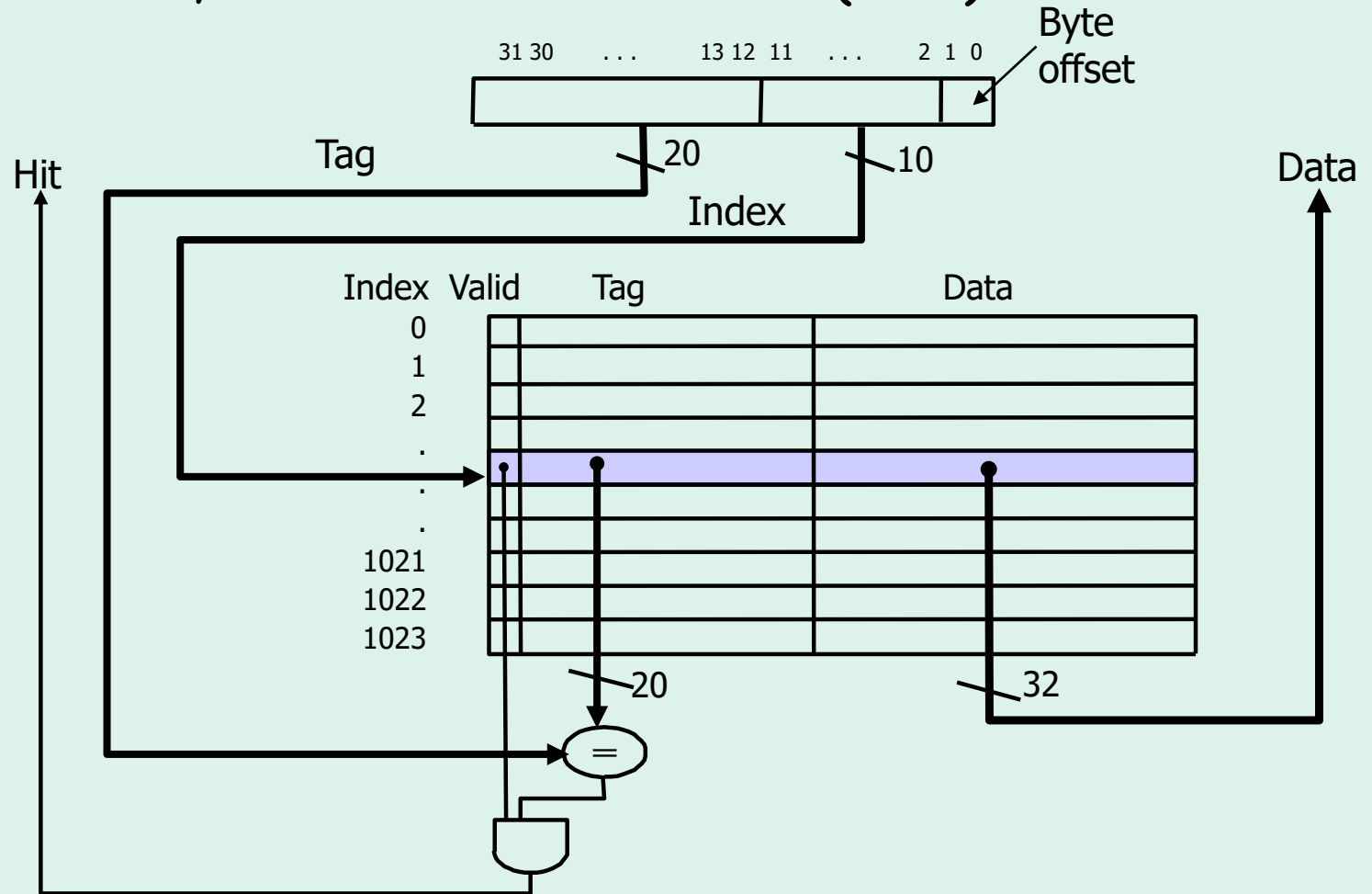
Key components of many-core processors

- Interconnection network
 - connecting many modules on a chip achieving high throughput and low latency
- Main memory and caches
 - Caches are used to reduce latency and to lower network traffic
 - A parallel program has private data and shared data
 - New issues are **cache coherence** and memory consistency
- Core
 - High-performance superscalar processor providing a hardware mechanism to support thread synchronization



MIPS Direct Mapped Cache Example

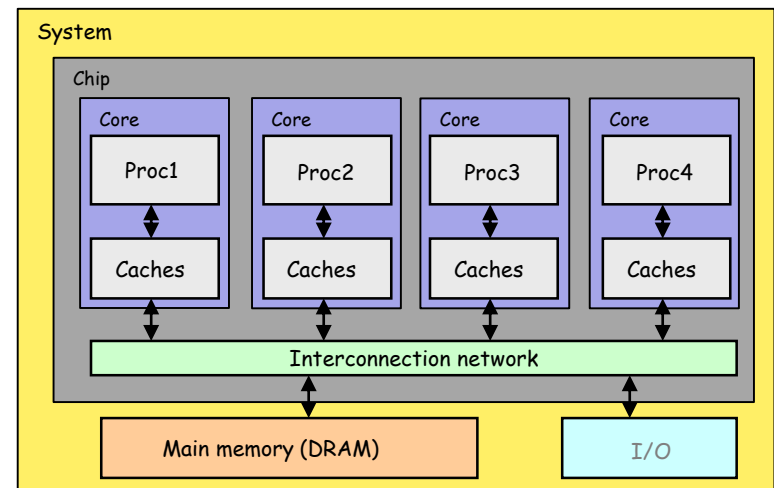
- One word/block, cache size = 1K words (4KB)



What kind of locality are we taking advantage of?

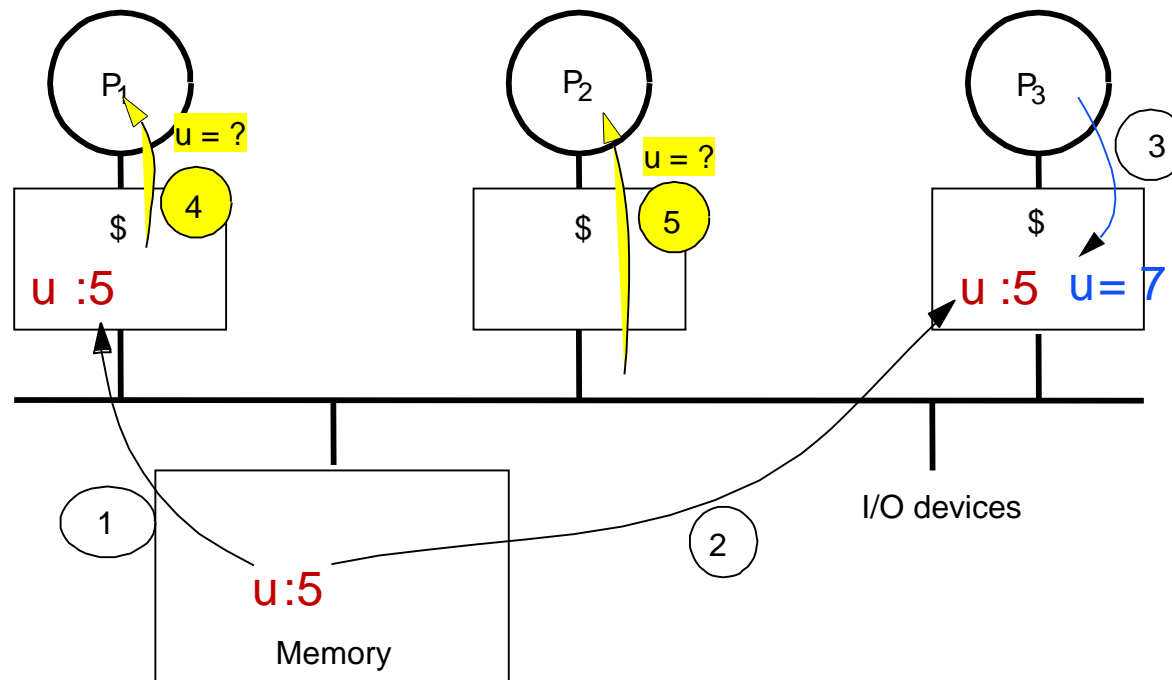
Cache writing policy

- **Write-through**
 - writing is done synchronously both to the cache and to the main memory. All stores update the main memory.
- **Write-back**
 - initially, writing is done only to the cache. The write to the main memory is postponed until the modified content is about to be replaced by another cache block.
 - reduces the required network and memory bandwidth.
- **Which policy is better for many-core?**



Cache Coherence Problem

- Processors see different values for shared data **u** after event 3
- With **write-back caches**, value written back to memory depends on which cache flushes or writes back value when
 - Processes accessing main memory may see stale (out-of-date) value
- Unacceptable for programming, and its frequent!



Cache Coherence Problem

- Processors may see different values through their caches
 - assuming a write-back cache
 - after the value of X has been written by A, A's cache contains the new value, but B's cache and the main memory do not

| Time | Event | Cache contents for processor A | Cache contents for processor B | Memory contents for location X |
|------|-----------------------------|--------------------------------|--------------------------------|--------------------------------|
| 0 | | | | 1 |
| 1 | Processor A reads X | 1 | | 1 |
| 2 | Processor B reads X | 1 | 1 | 1 |
| 3 | Processor A stores 0 into X | 0 | 1 | 1 |



Cache Coherence and enforcing coherence

- Cache Coherence

- All reads by any processor must return the most recently written value
- Writes to **the same location** by any two processors are seen in the same order by all processors

- Cache coherence protocols

- Snooping (write invalidate / write update)
 - Each core tracks sharing status of each block
- Directory based
 - Sharing status of each block kept in one location



Snooping coherence protocols using bus network

- Write invalidate

- On write, invalidate all other copies by an invalidate broadcast
- Use bus itself to serialize
 - Write cannot complete until bus access is obtained

| Processor activity | Bus activity | Contents of processor A's cache | Contents of processor B's cache | Contents of memory location X |
|-----------------------------|--------------------|---------------------------------|---------------------------------|-------------------------------|
| | | | | 0 |
| Processor A reads X | Cache miss for X | 0 | | 0 |
| Processor B reads X | Cache miss for X | 0 | 0 | 0 |
| Processor A writes a 1 to X | Invalidation for X | 1 | | 0 |
| Processor B reads X | Cache miss for X | 1 | 1 | 1 |

- Write update

- On write, update all copies

Snooping coherence protocols using bus network

- Cache lines marked as **invalid**, **shared** or **modified (exclusive)**
 - The **shared** state indicates that the block in the private cache is potentially shared.
 - The **modified** state indicates that the block has been updated in the private cache; note that the modified state implies that the block is exclusive.
 - Only writes to shared lines need an invalidate broadcast
 - After this, the line is marked as exclusive



Snooping coherence protocols using bus network

- The coherence mechanism of a private cache

| | Request | Source | State of addressed cache block | Type of cache action | Function and explanation |
|----|------------|-----------|--------------------------------|----------------------|--|
| | Read hit | Processor | Shared or modified | Normal hit | Read data in local cache. |
| | Read miss | Processor | Invalid | Normal miss | Place read miss on bus. |
| | Read miss | Processor | Shared | Replacement | Address conflict miss: place read miss on bus. |
| | Read miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place read miss on bus. |
| | Write hit | Processor | Modified | Normal hit | Write data in local cache. |
| C1 | Write hit | Processor | Shared | Coherence | Place invalidate on bus. These operations are often called upgrade or <i>ownership</i> misses, since they do not fetch the data but only change the state. |
| | Write miss | Processor | Invalid | Normal miss | Place write miss on bus. |
| | Write miss | Processor | Shared | Replacement | Address conflict miss: place write miss on bus. |
| | Write miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place write miss on bus. |
| | Read miss | Bus | Shared | No action | Allow shared cache or memory to service read miss. |
| C2 | Read miss | Bus | Modified | Coherence | Attempt to share data: place cache block on bus and change state to shared. |
| C3 | Invalidate | Bus | Shared | Coherence | Attempt to write shared block; invalidate the block. |
| C4 | Write miss | Bus | Shared | Coherence | Attempt to write shared block; invalidate the cache block. |
| C5 | Write miss | Bus | Modified | Coherence | Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache. |



Snooping coherence protocols using bus network

- The coherence mechanism of a private cache

| | Request | Source | State of addressed cache block | Type of cache action | Function and explanation |
|----|------------|-----------|--------------------------------|----------------------|--|
| | Read hit | Processor | Shared or modified | Normal hit | Read data in local cache. |
| | Read miss | Processor | Invalid | Normal miss | Place read miss on bus. |
| | Read miss | Processor | Shared | Replacement | Address conflict miss: place read miss on bus. |
| | Read miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place read miss on bus. |
| | Write hit | Processor | Modified | Normal hit | Write data in local cache. |
| C1 | Write hit | Processor | Shared | Coherence | Place invalidate on bus. These operations are often called upgrade or <i>ownership</i> misses, since they do not fetch the data but only change the state. |
| | Write miss | Processor | Invalid | Normal miss | Place write miss on bus. |
| | Write miss | Processor | Shared | Replacement | Address conflict miss: place write miss on bus. |
| | Write miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place write miss on bus. |
| | Read miss | Bus | Shared | No action | Allow shared cache or memory to service read miss. |
| C2 | Read miss | Bus | Modified | Coherence | Attempt to share data: place cache block on bus and change state to shared. |
| C3 | Invalidate | Bus | Shared | Coherence | Attempt to write shared block; invalidate the block. |
| C4 | Write miss | Bus | Shared | Coherence | Attempt to write shared block; invalidate the cache block. |
| C5 | Write miss | Bus | Modified | Coherence | Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache. |



Snooping coherence protocols using bus network

- The coherence mechanism of a private cache

| | Request | Source | State of addressed cache block | Type of cache action | Function and explanation |
|----|------------|-----------|--------------------------------|----------------------|--|
| C1 | Read hit | Processor | Shared or modified | Normal hit | Read data in local cache. |
| | Read miss | Processor | Invalid | Normal miss | Place read miss on bus. |
| | Read miss | Processor | Shared | Replacement | Address conflict miss: place read miss on bus. |
| | Read miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place read miss on bus. |
| | Write hit | Processor | Modified | Normal hit | Write data in local cache. |
| | Write hit | Processor | Shared | Coherence | Place invalidate on bus. These operations are often called upgrade or <i>ownership</i> misses, since they do not fetch the data but only change the state. |
| | Write miss | Processor | Invalid | Normal miss | Place write miss on bus. |
| C2 | Write miss | Processor | Shared | Replacement | Address conflict miss: place write miss on bus. |
| | Write miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place write miss on bus. |
| | Read miss | Bus | Shared | No action | Allow shared cache or memory to service read miss. |
| C3 | Read miss | Bus | Modified | Coherence | Attempt to share data: place cache block on bus and change state to shared. |
| C4 | Invalidate | Bus | Shared | Coherence | Attempt to write shared block; invalidate the block. |
| C5 | Write miss | Bus | Shared | Coherence | Attempt to write shared block; invalidate the cache block. |
| | Write miss | Bus | Modified | Coherence | Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache. |



Snooping coherence protocols using bus network

- The coherence mechanism of a private cache

C1

| Request | Source | State of addressed cache block | Type of cache action | Function and explanation |
|-----------|-----------|--------------------------------|----------------------|--|
| Read hit | Processor | Shared or modified | Normal hit | Read data in local cache. |
| Read miss | Processor | Invalid | Normal miss | Place read miss on bus. |
| Read miss | Processor | Shared | Replacement | Address conflict miss: place read miss on bus. |
| Read miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place read miss on bus. |
| Write hit | Processor | Modified | Normal hit | Write data in local cache. |
| Write hit | Processor | Shared | Coherence | Place invalidate on bus. These operations are often called upgrade or <i>ownership</i> misses, since they do not fetch the data but only change the state. |

C2

| | | | | |
|------------|-----------|----------|-------------|---|
| Write miss | Processor | Invalid | Normal miss | Place write miss on bus. |
| Write miss | Processor | Shared | Replacement | Address conflict miss: place write miss on bus. |
| Write miss | Processor | Modified | Replacement | Address conflict miss: write-back block, then place write miss on bus. |
| Read miss | Bus | Shared | No action | Allow shared cache or memory to service read miss. |
| Read miss | Bus | Modified | Coherence | Attempt to share data: place cache block on bus and change state to shared. |

C3

| | | | | |
|------------|-----|--------|-----------|--|
| Invalidate | Bus | Shared | Coherence | Attempt to write shared block; invalidate the block. |
|------------|-----|--------|-----------|--|

C4

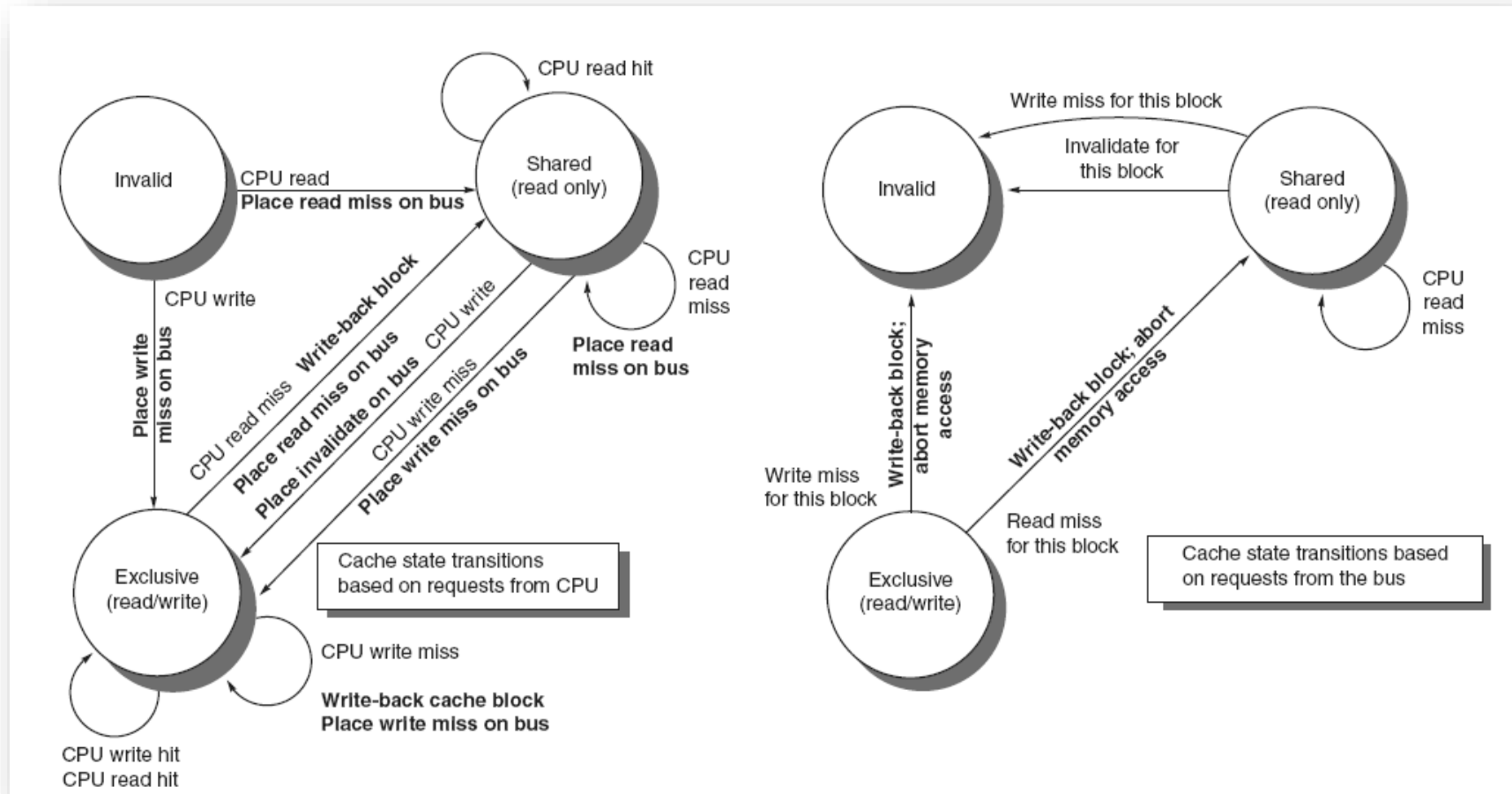
| | | | | |
|------------|-----|--------|-----------|--|
| Write miss | Bus | Shared | Coherence | Attempt to write shared block; invalidate the cache block. |
|------------|-----|--------|-----------|--|

C5

| | | | | |
|------------|-----|----------|-----------|---|
| Write miss | Bus | Modified | Coherence | Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache. |
|------------|-----|----------|-----------|---|

Snooping coherence protocols using bus network

- A write invalidate, cache coherence protocol for a private write-back cache showing the states and state transitions for each block in the cache



Snooping coherence protocols using bus network

- The basic coherence protocol
 - **MSI** (Modified, Shared, Invalid) protocol
- Extensions
 - **MESI** (Modified, **Exclusive**, Shared, Invalid) protocol
 - **MOESI** (MESI + **Owned**) protocol



Directory Protocols



- Snooping coherence protocols are based on the use of bus network.
What are the protocols for mesh topology NoC?
- Directory protocols
 - A logically-central directory keeps track of where the copies of each cache block reside. Caches consult this directory to ensure coherence.



Coherence influences cache miss rate



- Coherence misses
 - True sharing misses
 - Write to shared block (transmission of invalidation)
 - Read
 - False sharing misses



Sequential version as the baseline

- A sequential program main01.c and the execution result
- Computations in blue color are fully parallel

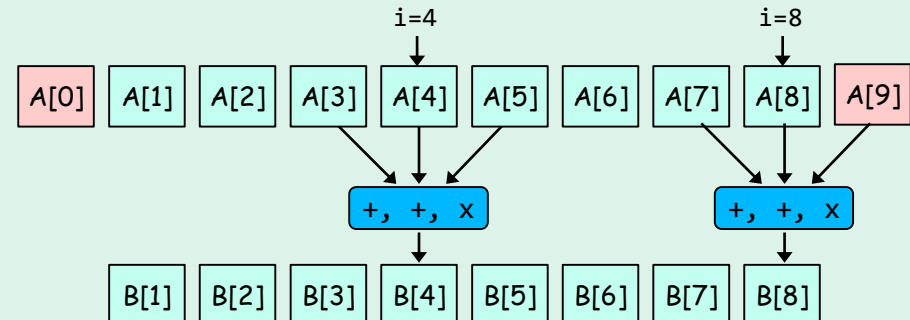
```
#define N 8      /* the number of grids */
#define TOL 15.0 /* tolerance parameter */
float A[N+2], B[N+2];

void solve () {
    int i, done = 0;
    while (!done) {
        float diff = 0;
        for (i=1; i<=N; i++) {
            B[i] = 0.333 * (A[i-1] + A[i] + A[i+1]);
            diff = diff + fabsf(B[i] - A[i]);
        }
        if (diff < TOL) done = 1;
        for (i=1; i<=N; i++) A[i] = B[i];

        for (i=0; i<=N+1; i++) printf("%6.2f ", B[i]);
        printf("| diff=%6.2f\n", diff); /* for debug */
    }
}

int main() {
    int i;
    for (i=1; i<=N-1; i++) A[i] = 100+i*i;
    solve();
}
```

| | | | | | | | | | | |
|------|-------|--------|--------|--------|--------|-------|-------|-------|------|-------------|
| 0.00 | 68.26 | 104.56 | 109.56 | 116.55 | 125.54 | 86.91 | 45.29 | 0.00 | 0.00 | diff=129.32 |
| 0.00 | 57.55 | 94.03 | 110.11 | 117.10 | 109.56 | 85.83 | 44.02 | 15.08 | 0.00 | diff= 55.76 |
| 0.00 | 50.48 | 87.15 | 106.97 | 112.14 | 104.06 | 79.72 | 48.26 | 19.68 | 0.00 | diff= 42.50 |
| 0.00 | 45.83 | 81.45 | 101.99 | 107.62 | 98.54 | 77.27 | 49.17 | 22.63 | 0.00 | diff= 31.68 |
| 0.00 | 42.38 | 76.35 | 96.92 | 102.61 | 94.38 | 74.92 | 49.64 | 23.91 | 0.00 | diff= 26.88 |
| 0.00 | 39.54 | 71.81 | 91.87 | 97.87 | 90.55 | 72.91 | 49.44 | 24.49 | 0.00 | diff= 23.80 |
| 0.00 | 37.08 | 67.67 | 87.10 | 93.34 | 87.02 | 70.89 | 48.90 | 24.62 | 0.00 | diff= 22.12 |
| 0.00 | 34.88 | 63.89 | 82.62 | 89.06 | 83.67 | 68.87 | 48.09 | 24.48 | 0.00 | diff= 21.06 |
| 0.00 | 32.89 | 60.40 | 78.44 | 85.03 | 80.45 | 66.81 | 47.10 | 24.17 | 0.00 | diff= 20.26 |
| 0.00 | 31.07 | 57.19 | 74.55 | 81.23 | 77.35 | 64.72 | 45.98 | 23.73 | 0.00 | diff= 19.47 |
| 0.00 | 29.39 | 54.21 | 70.92 | 77.63 | 74.36 | 62.62 | 44.77 | 23.21 | 0.00 | diff= 18.70 |
| 0.00 | 27.84 | 51.46 | 67.52 | 74.23 | 71.47 | 60.52 | 43.49 | 22.64 | 0.00 | diff= 17.95 |
| 0.00 | 26.41 | 48.89 | 64.34 | 71.00 | 68.67 | 58.43 | 42.17 | 22.02 | 0.00 | diff= 17.23 |
| 0.00 | 25.07 | 46.50 | 61.35 | 67.94 | 65.97 | 56.37 | 40.84 | 21.38 | 0.00 | diff= 16.53 |
| 0.00 | 23.83 | 44.26 | 58.54 | 65.02 | 63.36 | 54.34 | 39.49 | 20.72 | 0.00 | diff= 15.85 |
| 0.00 | 22.68 | 42.17 | 55.88 | 62.24 | 60.85 | 52.34 | 38.14 | 20.05 | 0.00 | diff= 15.20 |
| 0.00 | 21.59 | 40.20 | 53.38 | 59.60 | 58.42 | 50.39 | 36.81 | 19.38 | 0.00 | diff= 14.58 |



Decomposition and assignment

- Single Program Multiple Data (SPMD)

- Decomposition: there are eight tasks to compute $B[i]$
- Assignment: the first four tasks for core 1, and the last four tasks for core 2

```
float A[N+2], B[N+2]; /* these are in shared memory */
float diff=0;          /* variable in shared memory */

void solve_pp (int pid, int ncores) {
    int i, done = 0;          /* private variables */
    int mymin = 1 + (pid * N/ncores); /* private variable */
    int mymax = mymin + N/ncores - 1; /* private variable */
    while (!done) {
        float mydiff = 0;
        for (i=mymin; i<=mymax; i++) {
            B[i] = 0.333 * (A[i-1] + A[i] + A[i+1]);
            mydiff = mydiff + fabsf(B[i] - A[i]);
        }
        diff = diff + mydiff;

        if (diff < TOL) done = 1;
        if (pid==1) diff = 0;
        for (i=mymin; i<=mymax; i++) A[i] = B[i];
    }
}

int main() { /* solve this using two cores */
    initialize shared data A and B;
    create thread1 and call solve_pp(1, 2);
    create thread2 and call solve_pp(2, 2);
}
```

Computation

Decomposition

B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8]

Assignment

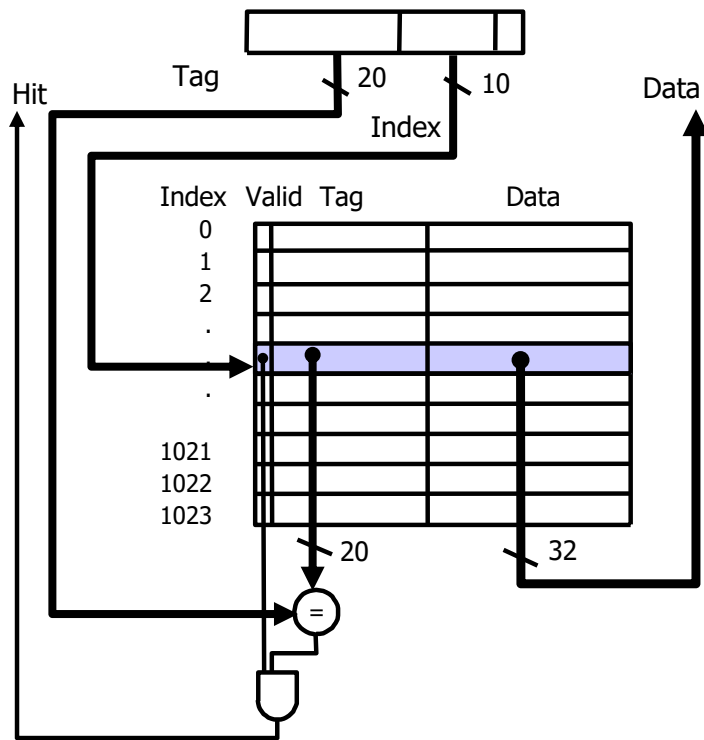
Core 1

B[1] B[2] B[3] B[4]

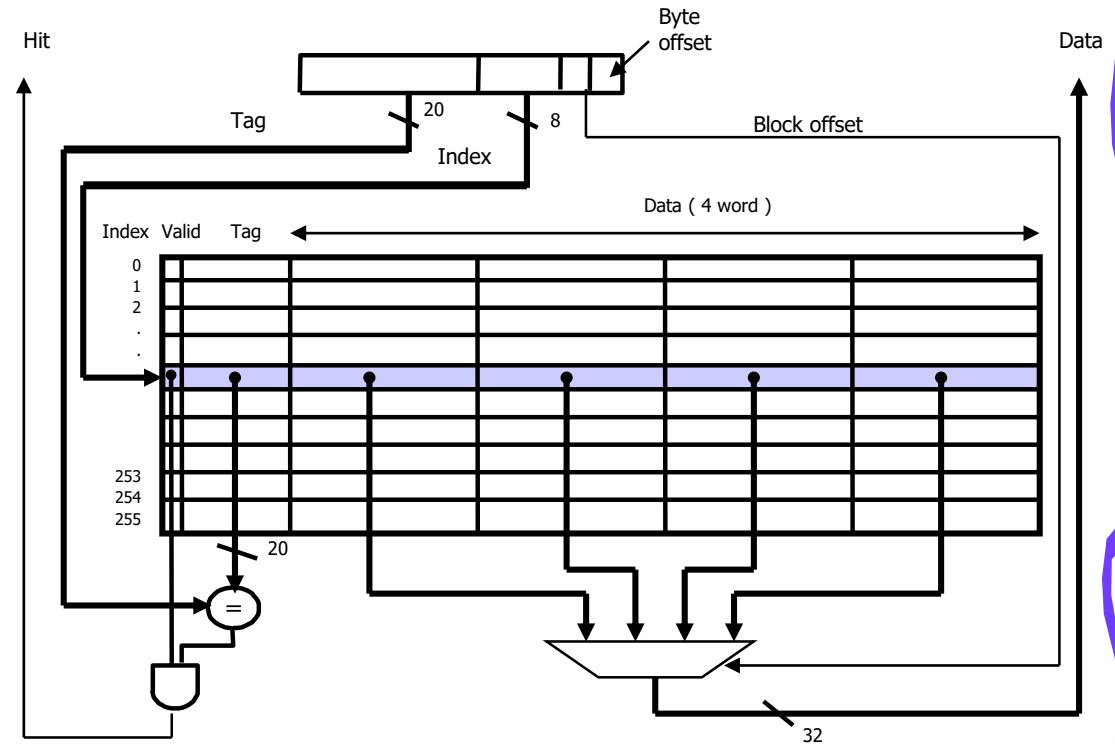
Core 2

B[5] B[6] B[7] B[8]

Two caches of different block sizes



One word/block



Four words/block

Data cache of four word block (block size is 16byte)



Core 1

| | | |
|--|--|------------------------|
| | | A[0], A[1], A[2], A[3] |
| | | A[4], A[5], A[6], A[7] |
| | | |
| | | |
| | | |

Core 2

| | | |
|--|--|------------------------|
| | | |
| | | A[4], A[5], A[6], A[7] |
| | | A[8], A[9] |
| | | |
| | | |

| | | |
|--|--|------------------------|
| | | A[0], A[1], A[2], A[3] |
| | | A[4], A[5], A[6], A[7] |
| | | A[8], A[9] |
| | | |
| | | |

| | | |
|--|--|------------------------|
| | | A[0], A[1], A[2], A[3] |
| | | A[4], A[5], A[6], A[7] |
| | | |
| | | |
| | | |

| | | |
|--|--|------------------------|
| | | |
| | | A[4], A[5], A[6], A[7] |
| | | A[8], A[9] |
| | | |
| | | |



One word/block

