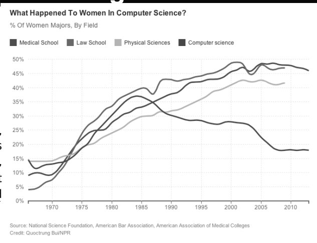
inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 21 Thread Level Parallelism II

DaMniki GaLurscitiag

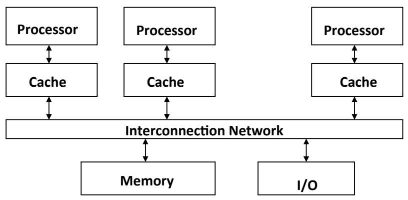


A lot of computing pioneers … were women… for decades, the number of women studying computer science was growing faster than the number of men. But in 1984, something changed…. early personal computers weren't much more than toys…. And these toys were marketed almost entirely to men and boys.

`http://www.npr.org/blogs/money/2014/10/21/357629765/when-women-stopped-coding`

Parallel Processing: Multiprocessor Systems (MIMD)

- MP - A computer system with at least 2 processors:



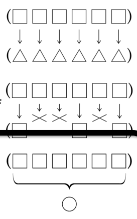
- Q1 – How do they share data?
- Q2 – How do they coordinate?
- Q3 – How many processors can be supported?

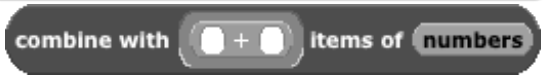
Shared Memory Multiprocessor (SMP)

- Q1 – Single address space shared by all processors/cores
- Q2 – Processors coordinate/communicate through shared variables in memory (via loads and stores)
  - Use of shared data must be coordinated via synchronization primitives (locks) that allow access to data to only one processor at a time
- All multicore computers today are SMP

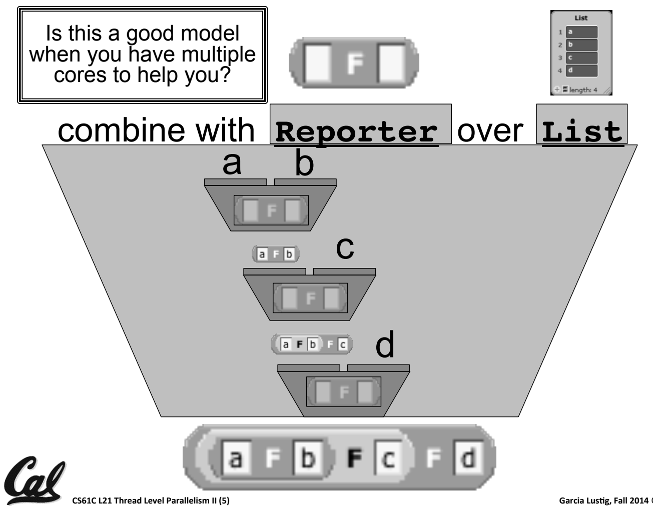
CS10 Review : Higher Order Functions with "CS10: Sum Reduction"

- Useful HOFs (you can build your own!)
  - map <u>`Reporter`</u> over <u>`List`</u>
    - Report a new list, every element `E` of `List` becoming `Reporter(E)`
  - keep items such that <u>`Predicate`</u> from <u>`List`</u>
    - Report a new list, keeping only elements `E` of `List` if `Predicate(E)`
  - combine with <u>`Reporter`</u> over <u>`List`</u>
    - Combine all the elements of `List` with `Reporter(E)`
    - This is also known as "reduce"





combine with (+) items of numbers


Is this a good model when you have multiple cores to help you?
F
List
a
b
c
d
length: 4
combine with Reporter over List
a
b
c
d
a F b
a F b F c
a F b F c F d

CS61C Example: Sum Reduction

- Sum 100,000 numbers on 100 processor SMP
  - Each processor has ID: 0 ≤ Pn ≤ 99
  - Partition 1000 numbers per processor
  - Initial summation on each processor [Phase I]
  - Aka, "the map phase"

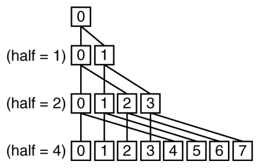
```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
    sum[Pn] = sum[Pn] + A[i];
```

- Now need to add these partial sums [Phase II]
  - Reduction: divide and conquer in "the reduce phase"
  - Half the processors add pairs, then quarter, …
  - Need to synchronize between reduction steps

Example: Sum Reduction

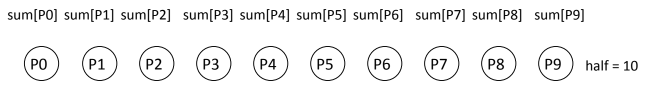
Second Phase:
After each processor has computed its "local" sum

This code runs simultaneously on each core



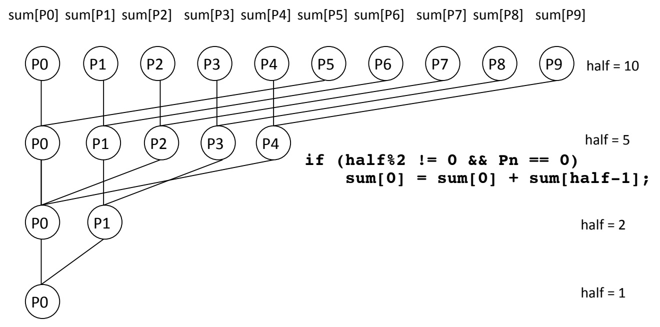
```
half = 100;
repeat
  synch();
  /* Proc 0 sums extra element if there is one */
  if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);
```

An Example with 10 Processors



sum[P0] sum[P1] sum[P2] sum[P3] sum[P4] sum[P5] sum[P6] sum[P7] sum[P8] sum[P9]
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9
half = 10

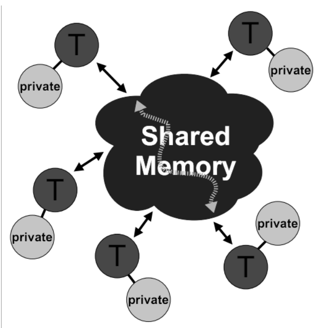
An Example with 10 Processors



sum[P0] sum[P1] sum[P2] sum[P3] sum[P4] sum[P5] sum[P6] sum[P7] sum[P8] sum[P9]
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9
half = 10
P0 P1 P2 P3 P4
half = 5
P0 P1
half = 2
P0
half = 1

```
if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
```

Memory Model for Multi-threading



- ✔ *All threads have access to the same, globally shared, memory*
- ✔ *Data can be shared or private*
- ✔ *Shared data is accessible by all threads*
- ✔ *Private data can only be accessed by the thread that owns it*
- ✔ *Data transfer is transparent to the programmer*
- ✔ *Synchronization takes place, but it is mostly implicit*

CAN BE SPECIFIED IN A LANGUAGE WITH MIMD SUPPORT – SUCH AS OPENMP

Peer Instruction

```
half = 100;
repeat
  synch();
  /* Proc 0 sums extra element if there is one */
  if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
until (half == 1);
```

What goes in Shared? What goes in Private?

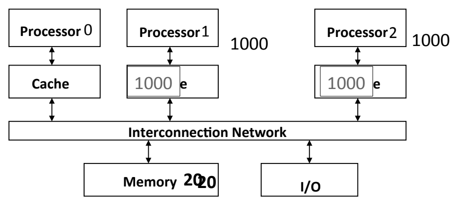
	half	sum	Pn
(a)	PRIVATE	PRIVATE	PRIVATE
(b)	PRIVATE	PRIVATE	SHARED
(c)	PRIVATE	SHARED	PRIVATE
(d)	SHARED	SHARED	PRIVATE
(e)	SHARED	SHARED	SHARED

Three Key Questions about Multiprocessors

- Q3 – How many processors can be supported?
- Key bottleneck in an SMP is the memory system
- Caches can effectively increase memory bandwidth/open the bottleneck
- But what happens to the memory being actively shared among the processors through the caches?

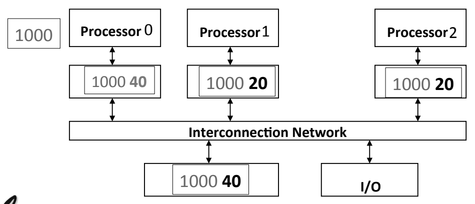
Shared Memory and Caches

- What if?
  - Processors 1 and 2 read Memory[1000] (value 20)



Shared Memory and Caches

- What if?
  - Processors 1 and 2 read Memory[1000]
  - Processor 0 writes Memory[1000] with 40



Processor 0 Write Invalidates Other Copies (Easy! Turn **Valid bit** off)

Keeping Multiple Caches Coherent

- Architect's job: shared memory ➔ keep cache values coherent
- Idea: When any processor has cache miss or writes, notify other processors via interconnection network
  - If only reading, many processors can have copies
  - If a processor writes, invalidate all other copies
- Shared written result can "ping-pong" between caches

How Does HW Keep $ Coherent?

Each cache tracks state of each *block* in cache:
Shared: up-to-date data, not allowed to write
other caches may have a copy
copy in memory is also up-to-date
Modified: up-to-date, changed (dirty), OK to write
no other cache has a copy,
copy in memory is out-of-date
- must respond to read request
Invalid: Not really in the cache

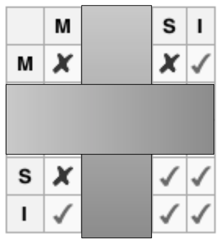
2 Optional Performance Optimizations of Cache Coherency via new States

Exclusive: up-to-date data, OK to write (change to modified)
no other cache has a copy,
copy in memory up-to-date
- Avoids writing to memory if block replaced
- Supplies data on read instead of going to memory

Owner: up-to-date data, OK to write (if invalidate shared copies first then change to modified)
other caches may have a copy (they must be in Shared state)
copy in memory not up-to-date
- So, owner must supply data on read instead of going to memory

Common Cache Coherency Protocol: MOESI (snoopy protocol)

- Each block in each cache is in one of the following states:
  - Modified (in cache)
  - Owned (in cache)
  - Exclusive (in cache)
  - Shared (in cache)
  - Invalid (not in cache)

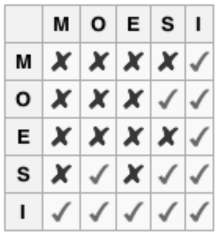


	M		S	I
M	✗		✗	✓
S	✗		✓	✓
I	✓		✓	✓

Compatability Matrix: Allowed states for a given cache block in any pair of caches

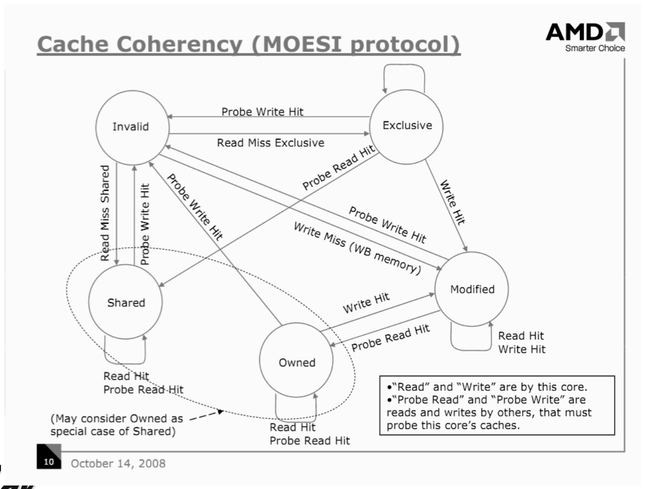
Common Cache Coherency Protocol: MOESI (snoopy protocol)

- Each block in each cache is in one of the following states:
  - Modified (in cache)
  - Owned (in cache)
  - Exclusive (in cache)
  - Shared (in cache)
  - Invalid (not in cache)



	M	O	E	S	I
M	✗	✗	✗	✗	✓
O	✗	✗	✗	✓	✓
E	✗	✗	✗	✗	✓
S	✗	✓	✗	✓	✓
I	✓	✓	✓	✓	✓

Compatability Matrix: Allowed states for a given cache block in any pair of caches



Cache Coherency (MOESI protocol)

- "Read" and "Write" are by this core.
- "Probe Read" and "Probe Write" are reads and writes by others, that must probe this core's caches.

Cache Coherency and Block Size

- Suppose block size is 32 bytes
- Suppose Processor 0 reading and writing variable X, Processor 1 reading and writing variable Y
- Suppose in X location 4000, Y in 4012
- What will happen?
- Effect called *false sharing*
- How can you prevent it?

DaMniki's Laptop? sysctl hw

```
hw.ncpu: 2
hw.byteorder: 1234
hw.memsize: 8589934592
hw.activecpu: 2
hw.physicalcpu: 2
hw.physicalcpu_max: 2
hw.logicalcpu: 2
hw.logicalcpu_max: 2
hw.cputype: 7
hw.cpusubtype: 4
hw.cpu64bit_capable: 1
hw.cpufamily: 2028621756
hw.cacheconfig: 2 1 2 0 0 0 0 0 0 0
hw.cachesize: 8321499136 32768 6291456 0 0 0 0 0 0 0
hw.pagesize: 4096
hw.busfrequency: 1064000000
hw.busfrequency_min: 1064000000
hw.busfrequency_max: 1064000000
hw.cpufrequency: 3060000000
hw.cpufrequency_min: 3060000000
hw.cpufrequency_max: 3060000000
hw.cachelinesize: 64
hw.l1icachesize: 32768
hw.l1dcachesize: 32768
hw.l2cachesize: 6291456
hw.tbfrequency: 1000000000
hw.packages: 1
hw.optional.floatingpoint: 1
hw.optional.mmx: 1
hw.optional.sse: 1
hw.optional.sse2: 1
hw.optional.sse3: 1
hw.optional.supplementalsse3: 1
hw.optional.sse4_1: 1
hw.optional.sse4_2: 0
hw.optional.x86_64: 1
hw.optional.aes: 0
hw.optional.avx1_0: 0
hw.optional.rdrand: 0
hw.optional.f16c: 0
hw.optional.enfstrg: 0
hw.machine = x86_64
```

Be careful! You can *change* some of these values with the wrong flags!

And In Conclusion, …

- Sequential software is slow software
  - SIMD and MIMD only path to higher performance
- Multiprocessor (Multicore) uses Shared Memory (single address space)
- Cache coherency implements shared memory even with multiple copies in multiple caches
  - False sharing a concern
- Next Time: OpenMP as simple parallel extension to C