CSE 160
Lecture 2

Cache Memories
Address Space Organization
Shared Memory Hierarchies
Threads Programming Model
Today’s lecture

• Cache memories

• Address space organization
  ‣ Shared memory
  ‣ Distributed memory

• Threads programming model
The processor-memory gap

- The result of technological trends
- Difference in processing and memory speeds growing exponentially over time

“Moore’s Law”

Processor-Memory Performance Gap:
(grows 50% / year)

µProc 60%/yr.

DRAM 7%/yr.

©2013 Scott B. Baden / CSE 160 / Winter 2013
An important principle: locality

- Memory accesses exhibit two forms of locality
  - Temporal locality (time)
  - Spatial locality (space)
- Often involves loops
- Opportunities for reuse
- Idea: construct a small & fast memory to cache re-used data

```plaintext
for t=0 to T-1
  for i = 1 to N-2
    u[i]=(u[i-1] + u[i+1])/2
```

©2013 Scott B. Baden / CSE 160 / Winter 2013
The Benefits of Cache Memory

• Let say that we have a small fast memory that is 10 times faster (access time) than main memory …

• If we find what we are looking for 90% of the time (a **hit**), the access time approaches that of fast memory

\[ T_{\text{access}} = 0.90 \times 1 + (1-0.9) \times 10 = 1.9 \]

• Memory appears to be 5 times faster

• We organize the references by **blocks**

• We can have multiple levels of cache
Sidebar

- If cache memory access time is 10 times faster than main memory …
- Cache “hit time” \( T_{\text{cache}} = \frac{T_{\text{main}}}{10} \)
- \( T_{\text{main}} \) is the \textit{cache miss penalty}
- And if we find what we are looking for \( f \times 100\% \) of the time (“cache hit rate”) …
- Access time = \( f \times T_{\text{cache}} + (1 - f) \times T_{\text{main}} \)
  = \( f \times T_{\text{main}} /10 + (1 - f) \times T_{\text{main}} \)
  = \( (1-(9f/10)) \times T_{\text{main}} \)
- We are now \( 1/(1-(9f/10)) \) times faster
- To simplify, we use \( T_{\text{cache}} = 1, T_{\text{main}} = 10 \)
Bang’s Memory Hierarchy

- Intel “Clovertown”
- Intel Xeon E5355 (Intro: 2006)
- Two “Woodcrest” dies on a multichip module
- Source: *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Table 2.16

**Line Size** = 64B (L1 and L2)

- **Access latency, throughput (clocks)**
  - 3, 1
  - 14*, 2
  
  *Software-visible latency will vary depending on access patterns and other factors*

- **Write update policy**: Writeback

**Associativity**

- 8
- 16

**Techreport.com/articles.x/10021/2**

©2013 Scott B. Baden / CSE 160 / Winter 2013
Different types of caches

- Separate Instruction (I) and Data (D)
- Unified (I+D)
- Direct mapped / Set associative
- Write Through / Write Back
- Allocate on Write / No Allocate on Write
- Last Level Cache (LLC)
- Translation Lookaside Buffer (TLB)
Set associative cache

- Why use the middle bits for the index?

3Cs of Cache
- Cold Start
- Capacity
- Conflict

Randal E. Bryant and David R. O
Examining Bang’s Memory Hierarchy

• /proc/cpuinfo summarizes the processor
  ‣ vendor_id : GenuineIntel
  ‣ model name : Intel(R) Xeon(R) CPU E5345 @2.33GHz
  ‣ cache size : 4096 KB
  ‣ cpu cores : 4
• processor : 0 through processor : 7
• More detailed information at /sys/devices/system/cpu/cpu*/cache/index*//*
Today’s lecture

• Cache memories

• Address space organization
  ‣ Shared memory
  ‣ Distributed memory

• Threads programming model
Address Space Organization

• We classify the address space organization of a parallel computer according to whether or not it provides global memory.

• If there is global memory we have a “shared memory” or “shared address space” architecture:
  > multiprocessor vs partitioned global address space

• When there is no global memory, we have a “shared nothing” architecture, also known as a multicomputer.
Multiprocessor organization

- Hardware automatically performs the global to local mapping using address translation mechanisms
- 2 types, depends on uniformity of memory access times
  - **UMA:** *Uniform Memory Access* time
    - Also called a Symmetric Multiprocessor (SMP)
  - **NUMA:** *Non-Uniform Memory Access* time
NUMA

- Non-Uniform Memory Access time
  - Processors see distant-dependent access times to memory
  - Implies physically distributed memory

- We often call these *distributed shared memory architectures*
  - Stanford Dash
  - Commercial example: SGI Origin Altix, up to 512 cores
  - Gordon system at San Diego Supercomputer Center
  - Software/hardware support to monitor sharers
Architectures without shared memory

- A core has direct access to local memory only
- Send and receive messages to obtain copies of data from other nodes
- We call this a *shared nothing* architecture, or a *multicomputer*
Parallel processing this course

- We will start by programming at the SMP level: multicore programming with threads
- We will then use multiple nodes: message passing
Today’s lecture

• Memory locality

• Address space organization
  ‣ Shared memory
  ‣ Distributed memory

• Threads programming model
**Threads execution model**

- Program executes a collection of independent instruction streams, called *threads*.

- A thread is similar to a procedure call with notable differences:
  - A new storage class: shared data
  - A procedure call is “synchronous:”
    - A return indicates completion
  - A spawned thread executes asynchronously until it completes
  - Both share global storage with caller
  - Synchronization may be needed when updating shared state (thread safety)

```
<table>
<thead>
<tr>
<th>Private</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared</td>
</tr>
</tbody>
</table>
```
Why threads?

- Processes are “heavy weight” objects scheduled by the OS
  - Protected address space, open files, and other state

- A thread AKA a lightweight process (LWP)
  - Threads share the address space and open files of the parent, but have their own stack
  - Reduced management overheads, e.g. thread creation
  - Kernel scheduler multiplexes threads
Threads programming model

• Start with a single root thread
• Fork-join parallelism to create concurrently executing threads
• Threads may or may not execute on different processors, and might be interleaved
• Threads communicate via shared memory
• Scheduling behavior dealt with separately

©2013 Scott B. Baden / CSE 160 / Winter 2013
SPMD execution model

• Most parallel programming is implemented under the Same Program Multiple Data programming model = “SPMD”
  ‣ Threads
  ‣ Message passing
  ‣ Other names: “loosely synchronous” or “bulk synchronous”

• Programs execute as a set of P processes or threads
  ‣ We specify P when we run the program
  ‣ Each thread/process usually assigned to a different physical processor

• Each process or thread
  ‣ Is initialized with the same code
  ‣ Has an associated index or rank, a unique integer
  ‣ Executes instructions at its own rate

• Threads communicate through shared memory
Multithreading in Practice

• POSIX Threads “standard” (pthreads):
  IEEE POSIX 1003.1c-1995
    ‣ Low level interface
    ‣ Beware of non-standard features

• Java threads not used in high performance computation

• C++11

• OpenMP – program annotations

• Parallel programming languages
  ‣ Co-array FORTRAN
  ‣ UPC
Hello world with pthreads

```cpp
#include <pthread.h> ...

void *Hello(void *arg) {
    sleep(1);
    int64_t _tid = reinterpret_cast<int64_t>(arg);
    int TID = _tid;
    cout << "Hello from thread " << TID << endl;
    pthread_exit(NULL); return 0;
}

int main(int argc, char *argv[]) {
    int NT = 3;
    pthread_t th[NT];
    for(int t=0;t<NT;t++) {
        int64_t t64 = t;
        assert(!pthread_create(&th[t], NULL, Hello,
                               reinterpret_cast<void*>(t64)));
    }
    for(int t=0;t<NT;t++)
        assert(!pthread_join(th[t], NULL));
    pthread_exit(NULL);
}
```

% g++ hello.cpp -lpthread
% a.out
Hello from thread 0
Hello from thread 1
Hello from thread 2
% a.out
Hello from thread 1
Hello from thread 0
Hello from thread 2
%a.out
Hello from thread 1
Hello from thread 0
Hello from thread 2
24 ©2013 Scott B. Baden / CSE 160 / Winter 2013

PUB = /share/class/public/cse160-wi13