Final Review

ICS432

Concurrent and High-Performance Programming

Henri Casanova (henric@hawaii.edu)

What to expect

- The final exam is non-cumulative
  - But of course having knowledge of pre-midterm material is needed (you can't forget what a lock is, etc.)
- Questions will be about post-midterm material
  - Speedup/Efficiency
  - Shared-memory programming
  - OpenMP
  - Programming for performance and locality
  - Lockfree programming

Sequential Program Optimization

- Make sure you go through the lecture notes and that you understand why some of the optimizations work
  - Loop unrolling
  - Array reference removals
  - Constant propagation
  - What can a profiler do for you?
  - etc.
- Any we should review now?

The Memory Bottleneck

- We have slow memories so our CPUs are not fully utilized for typical programs
- Therefore we came up with the concept of a cache: a small amount of memory that's "close" to the CPU
  - Therefore it's fast and affordable
- When a CPU references a byte in memory:
  - This byte and all of those "next to it" are brought into the cache
  - The memory is segmented as cache lines
- Therefore we have both *temporal and spatial locality*

Cache hit / Cache miss

- When referencing a byte in memory, the CPU first looks for it in the cache
- If it's in cache, we have a cache hit
  - Cache hits are good because fast
- If it's not in cache, we have a cache miss
  - Cache misses are bad because slow
  - But next time we need this byte or bytes next to it, it may be in cache
- Not all perfect: when the cache becomes full, cache lines are evicted from it
  - So you need to reuse the same data in cache often and use data next to it soon

Exercise

```
short Array[128];
for (i=0; i < 128; i++) {
    Array[i] = 42;
}
```

- How many cache misses assuming a 16-byte cache line and assuming that Array[0] is at the beginning of a cache line?
- Note that this is the best locality you could have

Exercise Solution

```
short Array[128];
for (i=0; i < 128; i++) {
    Array[i] = 42;
}
```

- How many cache misses assuming a 16-byte cache line and assuming that Array[0] is at the beginning of a cache line?

- Array[0]: miss, Array[1]...Array[7]: hit
- Array[8]: miss, Array[9]...Array[15]: hit
- …
- Array[120]: miss, Array[121]…Array[127]: hit
- Answer: 128 / 8 = 16 cache misses (hit rate = (128-16)/128)
  - The data is spread over 16 cache lines, which must be loaded

Common Assumptions

- To make things simpler, we typically make the following assumptions:
  - Arrays are aligned with cache lines
  - Initially the cache is empty
  - The Cache is fully-associative
    - It can store any cache line as long as the cache is not full

Exercise

```
int Array[128];
for (i=0; i < 128; i+=5) {
    Array[i] = 42;
}
```

- How many cache misses assuming an 80-byte cache line and assuming that Array[0] is at the beginning of a cache line?

Exercise Solution

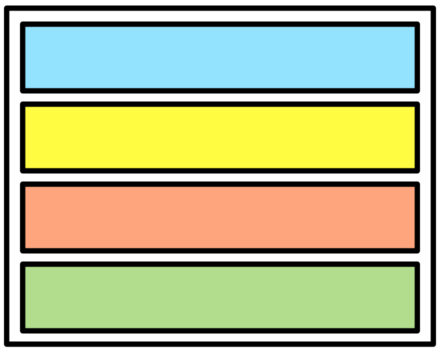
```
int Array[128];
for (i=0; i < 128; i+=5) {
    Array[i] = 42;
}
```

- There are 80/4 = 20 elements per cache line
- Due to the i+=5, we only use 4 elements in each cache line (indices 0, 5, 10, and 15)
- The first one is a miss, the next two are hits
- The patterns is   MHHH MHHH MHHH MHHH…
- We have a miss for 0, 20, 40, …, 120, i.e., 7 misses in total, for a total of 26 accesses
- We have 26 - 7 = 19 hits

2-D Arrays

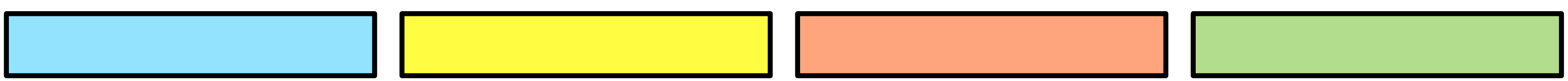
- Row-major vs. Column-major
- The implication of storing 2-D arrays into 1-D memory is that there are good ways and bad ways to cruise through the array
- Fundamental principle: contiguous memory accesses are good because of the cache and the use of cache line

Row-major Array



- Logical view of the array
  - A[i][j]: i = row, j = column

Array in memory



- Going through the rows (for i, for j) leads to perfectly sequential memory accesses (optimal hit rate)
- Going through the columns (for j, for i) leads to non-sequential memory accesses (worst hit rate)

Three kinds access patterns

- Named after what the array indexing does
- Constant:
  - for (i=0; i < N; i++) { a[0][j] = 12; }
  - Keeps accessing the same element
  - Perfect temporal locality
- Sequential:
  - for (j=0; j < N; j++) { a[0][j] = 12; }
  - Goes through a row
  - Perfect spatial locality
- Strided:
  - for (i=0; i < N; i++) { a[i][0] = 12; }
  - Goes through a column
  - Worst spatial locality

Matrix-Multiply

- In class we saw a characterization of accesses of the inner-loop of Matrix Multiplication
  - Each access to each of the three matrices was labelled as constant / sequential / strided

- Should we go through this again? or is this clear at this point?

Speedup/Efficiency

- We're accelerating a sequential program by parallelizing a function and we want to compute relevant quantities
  - Fraction of the sequential execution **time** that is due to that function: f (a number between 0 and 1)
    - OR 1 minus that fraction
  - Number of cores used: p
    - We assume perfect parallelization of the function
- Speedup = Seq time / Parallel time
  - Seq time = T (= (1-f) T + f T)
  - Parallel time = (1-f) T + f T / p
- Therefore, speedup = 1 / ((1-f) + f/p)
- Efficiency = Speedup / p
- And ... that's .... IT!
  - Find the unknown based on known quantities

Sample Exercise

- Say that the function accounts for 70% of the execution time
- How many cores should be used to achieve a speedup of 3? (what is p?)

Sample Exercise

- Say that the function accounts for 70% of the execution time
- How many cores should be used to achieve a speedup of 3? (what is p?)
- I define f as the fraction spent in the function
- speedup = 1 / ((1 - f) + f / p)
  - speedup = 3
  - f = 0.7
  - Solve for p!
- 1 - 0.7 + 0.7 / p = 1 / 3

➡ 0.3 p + 0.7 ~ 0.33p   (bad approx of 1/3)

➡ p ~ 0.7 / 0.03 ~ 23.33

➡ **p must be integer:** answer is 24

Sample Exercise

- We want an efficiency at 80% using p=10 cores
- What fraction of the execution time should the function account for?

Sample Exercise

- We want an efficiency at 80% using p=10 cores
- What fraction of the execution time should the function account for?
- I define f as the time spent in the function
- speedup = 1 / (1 - f + f/p)
- efficiency = speedup / p = 1 / (p - f p + f)
  - p = 10
  - efficiency = 0.8
  - solve for f!
- 0.8 = 1 / (10 - f * 10 + f)

➡ 10 - 9f = 1/0.8

➡ f ~ 0.97

Sample Exercise

- If f = 90% (the fraction we know how to parallelize), what’s the best speedup you can hope to achieve?

Sample Exercise

- If $f = 90\%$ (the fraction we know how to parallelize), what's the best speedup you can hope to achieve?

- speedup = $1 / (1 - f + f / p)$
- with $f = .9$:
  - speedup = $1 / (1 - 0.9 + 0.9 / p)$
  - $= 1 / (0.1 + 0.9 / p)$
- As $p \rightarrow \infty$, speedup $\rightarrow 1 / 0.1 = 10$
- Answer: 10
- Easy to determine without any math: with an infinite number of processors 90% of the time become zero, leaving only 10% of the time, hence a speedup of 10

OpenMP

- You should know the basic pragmas provided by OpenMP
- One “cool” feature of OpenMP is the “schedule” schedule clause
  - Make sure you understand the content in the “Scheduling” slides
- The fundamental trade-off:
  - If you have large work units, then overhead is low but load-balancing can be bad
  - If you have small work units, then load-balancing is good, but the overhead can be bad

OpenMP

- If you know your work units (e.g., loop iterations) are identical, you do static partitioning across threads
  - Once and for all: no overhead (your first Pthread assignment)
- If don't know how long all your work units take, then you can do dynamic partitioning
  - A thread "gets" a work unit, does it, and goes to the next one

```
while (1) {
    lock(mutex);
    index_to_work_on = i;
    i++;
    unlock(mutex);
    if (index_to_work_on >= N)
        break;
    do_iteration(index_to_work_on);
}
```

Transaction Memory

- What's the motivation?
- What problem does it solve?
- What's the difference between eager/lazy schemes
- Clearly nothing too in-depth since we didn't have any hands-on assignments

Lockfree Programming

- There will be some general questions on lockfree programming
- What is the point of it?
- Is it easy/hard?
- What's the ABA problem?
- Clearly nothing too in-depth since we didn't have any hands-on assignments

The End

- For questions that say “what is….”: NO NEED TO WRITE A NOVEL :)
  - keywords are important
- The final is scheduled for 2 hours, but should be doable in much less time.