Announcements

Homework #2, with a programming component, has been posted

Programming Task (50 points)

This programming task builds on what you learned in the threading labs. You will write a program to calculate the product of two square matrices. You'll perform the calculation with and without threads, and compare the run-times.

Matrix multiplication is a binary operation that takes as input a pair of matrices, and produces an output: the product, matrix. It is an often-used procedure in many algebra-intensive applications and high-performance programs. For the square matrices $A$ and $B$ shown above, their product is the following:

$$AB = \begin{pmatrix} \sum_{k=1}^{n} A_{ik}B_{kj} \\ \vdots \end{pmatrix}$$

where $A_{ik} = \sum_{j=1}^{n} A_{ij}B_{jk}$

A view of matrix multiplication.

Calculating the product of square matrices, using the naive (above) method, is $O(n^3)$, where $n$ is the dimension of the square matrix.

Finally, matrices should be composed of doubles or integers. It is up to you to ensure that they are generated, or hard-coded (for example, matrix $A$ can be matrix $B$ composed of all 2s).

Matrices are handled by the programmer. The programmer should implement them as a data structure. For example, you could use a 2D array or a map. The programmer should also specify the type of matrix (for example, a generator for random matrices).

Question 1: (25 points) Assume you are an employee of We-Foresee-Errors, a software company that specializes in writing programs intended to be run concurrently. You are working on a program for which you have the executable, but not the source code (you cannot modify the program). All that you have been told is that the program has 3 threads:

- Thread A: a1 < a2 < a3 < a4
- Thread B: b1 < b2 < b3
- Thread C: c1 < c2 < c3

Thread A has 4 instructions, Thread B has 3, and Thread C has 2. Each thread's instructions are executed sequentially (where < specifies the ordering that we've seen in lecture). However, the three threads are run concurrently, so the ordering among instructions from different threads is unpredictable from one invocation of the program to another. For example, the following two are instruction histories that might be realized if the program is run concurrently:

1. History: a1 < a2 < b1 < a3 < a4 < c1 < c2 < b2 < b3
2. History: a1 < b1 < a2 < b2 < c1 < c2 < a3 < a4

Also assume that dependencies among variables in use by the threads necessitate that the following constraints (Instruction Histories) be maintained:

- Constraint 1: a1 < a3
- Constraint 2: c2 < a4

We-Foresee-Errors guarantees that their software to be correct to a certain fault-tolerant level. The question you must answer: what percentage of the invocations of the software (when run concurrently) will give an incorrect result? An incorrect result is achieved when an execution history violates one of the constraints.

Question 2: (10 points) Why do semaphores have no get methods? (strictly speaking, a get method is a method that only retrieves the value of a variable and which does not update the variable's value)?
Last week’s lab

1. Provide the dimension for your 3D array
2. Specify how many asynchronous threads you issued and explain WHY
3. Provide a screen shot of the output of your program (as for example above)
4. Discuss/explain why having two threads does not necessarily reduce the execution time (summation) by half. If the speedup of using threads is not as expected, speculate why that may be the case.
From last time ...

<table>
<thead>
<tr>
<th>Q: How/when do the two threads complete executing?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread A</td>
</tr>
<tr>
<td>a1</td>
</tr>
<tr>
<td>sem1.wait()</td>
</tr>
<tr>
<td>sem2.signal()</td>
</tr>
<tr>
<td>a2</td>
</tr>
<tr>
<td>Thread B</td>
</tr>
<tr>
<td>b1</td>
</tr>
<tr>
<td>sem2.wait()</td>
</tr>
<tr>
<td>sem1.signal()</td>
</tr>
<tr>
<td>b2</td>
</tr>
</tbody>
</table>
Q: How/when do the two threads complete executing?

Thread A

```
a1
sem1.wait()
sem2.signal()
a2
```

Thread B

```
b1
sem2.wait()
sem1.signal()
b2
```

```
sem1 = Semaphore(0)
sem2 = Semaphore(0)
```

Deadlock, regardless of whether thread A or B starts first.
From last time ...

Q: How do we use semaphores to restrict access to a shared variable so that at most a single thread is trying to access and modify the variable’s value at one time
From last time ...

Q: How do we use semaphores to restrict access to a shared variable so that at most a single thread is trying to access and modify the variable’s value at one time

Thread A

```java
mutex.wait()
count = count + 1
mutex.signal()
```

Thread B

```java
mutex.wait()
count = count + 1
mutex.signal()
```

A decrements: mutex -> 0
B decrements: mutex -> -1 (attempts, self blocks)
A updates count
A increments: mutex -> 0 (B unblocked)
B updates count
B increments: mutex -> 1

mutex = Semaphore(1)

Thread A < Thread B
Thread B < Thread A
From last time ...

In the “real” world, there are usually far more threads and/or processes than one, or two. **Q: How do we use semaphores to limit the number of threads that can access a shared variable?**
In the “real” world, there are usually far more threads and/or processes than one, or two. **Q: How do we use semaphores to limit the number of threads that can access a shared variable?**

```
multplex = Semaphore(2)

Semaphore  
2

Thread A
- multiplex.wait()
- x = x + 1
- multiplex.signal()

Thread B
- multiplex.wait()
- x = x - 12
- multiplex.signal()

Thread c
- multiplex.wait()
- x = 4
- multiplex.signal()
```
Chapter 1
Processors, Cache, and Memory
Modern processors have large but slow “main” memory.

Smaller (than memory) but faster (than memory) caches (multiple levels).

Limited size but very fast registers and ALUs.
Modern processors have large but slow “main” memory.

Smaller (than memory) but faster (than memory) caches (multiple levels).

Limited size but very fast registers and ALUs.

Q: How does this modern processor architecture dictate how we should design software?
Modern processors have large but slow “main” memory.

Smaller (than memory) but faster (than memory) caches (multiple levels).

Limited size but very fast registers and ALUs.

Q: How does this modern processor architecture dictate how we should design software?

A program that is “too large” to fit into cache must be fetched from main memory, which is slow. Hence the goal is to write programs that are cache friendly.
Modern Processors - Cache

But even IF a program can fit in its entirety in the cache ...

---

Diagram:

- Registers
  - Level 1 Cache
    - Level 2 Cache
      - Main Memory
        - Electronic Disk
Modern Processors - Cache

But even IF a program can fit in its entirety in the cache ...

Scenario: For a hypothetical program that can fit in cache, assume that Register 2 (R2) needs such-and-such data ... where does the computer/program look for that data, and if not found, where does it get it from?
But even IF a program can fit in its entirety in the cache ...

Scenario: For a hypothetical program that can fit in cache, assume that Register 2 (R2) needs such-and-such data ... where does the computer/program look for that data, and if not found, where does it get it from?

When the CPU issues a read (load the registers) request, first-level cache logic checks whether that data item is already in cache. If ...

- Yes -> __________
- No -> __________
Modern Processors - Cache

But even IF a program can fit in its entirety in the cache ...

Scenario: For a hypothetical program that can fit in cache, assume that Register 2 (R2) needs such-and-such data ... where does the computer/program look for that data, and if not found, where does it get it from?

When the CPU issues a read (load the registers) request, first-level cache logic checks whether that data item is already in cache. If ...

- Yes -> Cache hit
- No -> Cache miss

Q: If a cache miss occurs, where is the sought-after item retrieved from?
Modern Processors - Cache

But even IF a program can fit in its entirety in the cache ...

Scenario: For a hypothetical program that can fit in cache, assume that Register 2 (R2) needs such-and-such data ... where does the computer/program look for that data, and if not found, where does it get it from?

If there is a level 1 cache miss, level 2 cache is checked. If that is a miss, then main memory, and potentially electronic disc (Hard Disk) is checked.

Q: If data is retrieved from L2 cache, what does that mean?

Q: Where in L1 cache should data be placed?

(on board discussion)
But even IF a program can fit in its entirety in the cache ...

**Miss rate**: typically 3-10% for L1 cache, and less than 2% for L2 cache

**Hit time**: Time to deliver a line in cache to the processor. Typically 1 clock cycle to L1 cache, 3-8 clock cycles for L2 cache
Modern Processors - Cache

But even IF a program can fit in its entirety in the cache ...

- **Miss rate**: typically 3-10% for L1 cache, and less than 2% for L2 cache

- **Hit time**: Time to deliver a line in cache to the processor. Typically 1 clock cycle to L1 cache, 3-8 clock cycles for L2 cache

- **Miss penalty**: time incurred because of cache miss. Typically 25-100 cycles

- **Average access time** = hit time + miss rate * miss penalty
Modern Processors - Cache

Q: How does cache improve performance?

Q: If cache access is 20ns, and memory access is 40ns, is a computer with cache twice as fast as a computer without cache?
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of \( \tau \) “faster” than memory?

Q: What factors other than cache and memory access speed determine the speed of execution of a program?
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta$ = cache reuse ratio; the fraction of loads or reads that can be reused from cache

Q: Intuitively, do we want a higher or lower $\beta$?
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache}$

$T_m = \text{access time to main memory}$

$T_c$

Q: What is $T_c$ in terms of $T_m$?
Modern Processors - Cache

**Q: How does cache improve performance?**

**Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?**

$\beta$ = cache reuse ratio; the fraction of loads or reads that can be reused from cache

$T_m$ = access time to main memory

$T_c$ = $T_m / \tau$ = access time for cache

**Q: What is $\tau$?**
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta$ = cache reuse ratio; the fraction of loads or reads that can be reused from cache

$T_m$ = access time to main memory

$T_c$ = $T_m / \tau$ = access time for cache

$\tau$ is the ratio of $T_m/T_c$

Q: What is the average access time of data in terms of $T_m$, $T_c$, and $\beta$?
Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache}$

$T_m = \text{access time to main memory}$

$T_c = T_m / \tau = \text{access time for cache}$

$T_{av} = \beta T_c + (1-\beta)T_m$

Q: What value should go here?
Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of \( \tau \) “faster” than memory?

\[ \beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache} \]

\[ T_m = \text{access time to main memory} \]

\[ T_c = T_m / \tau = \text{access time for cache} \]

\[ T_{av} = \beta T_c + (1-\beta)T_m \]
Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of \( \tau \) “faster” than memory?

\[ \beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache} \]

\[ T_m = \text{access time to main memory} \]

\[ T_c = \frac{T_m}{\tau} = \text{access time for cache} \]

\[ T_{av} = \beta T_c + (1 - \beta) T_m \]

\[ G(\tau, \beta) = \frac{T_m}{T_{av}} = \text{access performance gain} \]

(derivation, on the board)
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta$ = cache reuse ratio; the fraction of loads or reads that can be reused from cache

$T_m$ = access time to main memory

$T_c$ = $T_m / \tau$ = access time for cache

$T_{av}$ = $\beta T_c + (1-\beta)T_m$

$G(\tau, \beta) = T_m / T_{av}$ = access performance gain

= $\tau T_c / (\beta T_c + (1-\beta)\tau T_c)$

= $\tau / (\beta + \tau (1-\beta))$

Q: Using this formula, under what conditions does using cache have a significant improvement over a system that does not use cache?
Modern Processors - Cache

Q: How does cache improve performance?

Q: How do we estimate the performance gain of a cache that is a factor of $\tau$ “faster” than memory?

$\beta$ = cache reuse ratio; the fraction of loads or reads that can be reused from cache

$T_m$ = access time to main memory

$T_c = T_m / \tau$ = access time for cache

$T_{av} = \beta T_c + (1 - \beta) T_m$

$G(\tau, \beta) = T_m / T_{av} = \text{access performance gain}$

$= \tau T_c / (\beta T_c + (1 - \beta) \tau T_c)$

$= \tau / (\beta + \tau(1 - \beta))$

Q: Is cache that is twice as fast going to result in a system that is twice as good? (the salesman pitch)
Modern Processors - Cache

\[ G(\tau, \beta) = \frac{\tau}{\beta + \tau(1-\beta)} \]
\[ \beta = \text{cache reuse ratio} \]
\[ \tau = \frac{T_m}{T_c} \]

Assume a program with cache reuse ratio of 0.6, and a computer with 60ns cache access time and 120ns access time to main memory.

Q: What performance improvement can you expect if you upgrade your cache so that it has a 30ns access time?

In class exercise
Q: When $\tau$ is increased from 5 to 10, how is performance improved?
Modern Processors - Cache

Q: Is there a scenario when cache does not have a positive effect on performance?

Cache improves performance ONLY if data access has \textit{locality of reference}: data items are used from cache before they are evicted.

Temporal locality:

Spatial locality:
Q: Is there a scenario when cache does not have a positive effect on performance?

Cache improves performance ONLY if data access has **locality of reference**: data items are used from cache before they are evicted.

**Temporal locality**: reuse of cache memory location; the concept that a resource that is referenced at time \(x\) will be referenced soon in the near future.

**Spatial locality**: if a cache memory location is used, then it is likely that a nearby memory location will be referenced in the near future.
Q: Is there a scenario when cache does not have a positive effect on performance?

Cache improves performance ONLY if data access has **locality of reference**: data items are used from cache before they are evicted.

**Temporal locality**: reuse of cache memory location; the concept that a resource that is referenced at time $x$ will be referenced soon in the near future.

**Spatial locality**: if a cache memory location is used, then it is likely that a nearby memory location will be referenced in the near future.

**Spatial locality is concerned with how data is stored in cache. This is language specific.**
Modern Processors - Cache

Example: Matrix summation

<table>
<thead>
<tr>
<th>row</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q: What is the sum of the entries of the matrix? Assume $M \gg N$

Task: What is possible pseudocode?
Modern Processors - Cache

Example: Matrix summation

<table>
<thead>
<tr>
<th>row</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache:
Modern Processors - Cache

Example: Matrix summation

Matrix stored by rows

Cache:
Example: Matrix summation

Matrix stored by rows

Cache:
Modern Processors - Cache

Example: Matrix summation

Matrix stored by rows

Cache:
Example: Matrix summation

Matrix stored by rows

Cache:
Example: Matrix summation

Matrix stored by rows

Cache:
Modern Processors - Cache

Example: Matrix summation

Matrix stored by rows

Cache:
Modern Processors - Cache

Example: Matrix summation

Notice that the matrix data items are being placed into cache in rows, and that some data items in cache span different rows.

Also all columns of some rows, but not all columns of all rows, fit into cache.

What are the consequence of that?
Assume a square matrix (m = n)

```c
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```c
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

Q: What is different between these two pieces of code?

(on the board discussion)
Modern Processors - Cache

Assume a square matrix (m = n)

For each row, sum the columns

For each column, sum the rows

Q: Do these two methods output the same result?
Q: Do these two methods consume the same amount of time?
Assume a square matrix \(( m = n \) )

For each row, sum the columns

For each column, sum the rows

Q: Do these two methods output the same result?

Q: Do these two methods consume the same amount of time?

```
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    }
    return sum;
}
```
Modern Processors - Cache

Assume a square matrix \((m = n)\)

For each column, sum the rows

Using `sumArray1`, you sum the **rows** of the array, each of which is “small enough” to fit into cache, so for each \(a[i]\), the \([j]\) component is in cache, and does not need to be fetched from memory. For the completion of the inner-most for loop, a cache miss is **NOT** guaranteed.
Modern Processors - Cache

Assume a square matrix (m = n)

For each row, sum the columns

For each column, sum the rows

Using sumArray2, you sum the \textbf{columns} of each array, but ALL of the columns for each row are NOT stored in cache, because cache stores arrays “by row,” and is not big enough to hold a large array. For the completion of the inner-most for loop, there is a GUARANTEED cache miss.

```
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    }
    return sum;
}
```
Up Next ...

Concurrent Matrix Multiplication
Shared Variable Programming