Applications with synchronization

Scott B. Baden
Announcements

• Makeup session on 10/7
  ‣ 1:00 to 2:20, EBU3B 1202
  ‣ Will be recorded and available for viewing
Today’s lecture

• Callbacks with ref type arguments
• Sidebar on vectors and iterators
• Performance analysis of Merge sort
• More on synchronization
• Stencil Methods
• Performance Characterization
Creating references in thread callbacks

• The thread constructor copies each argument into local private storage, once the thread has launched

• Consider this call
  
  ```cpp
  int t = ..., vector<int64_t>& candidates = ... 
  thread(testPrimes, t, candidates, primes)
  ```

  testPrimes:
  for (i=low; i<hi; i++)
      if (isPrime (candidates[i])){
          _nPrimes++;
          primes[i] = TRUE;
      }

• If `candidates` is a large vector, it will be copied at great expense when each thread is launched
Sources

- Show prime number example in section
- Also use examples in cse160/wi10/Lec03, e.g. vector multiplication
- /fa08/Lectures/Week7/Lec14.ppt: The computation void *summ(void *arg){
- /fa08/Lectures/Week2/Lec03.ppt: The computation void *summ(void *arg){
- ./fa08/Lectures/Week2/sumFixed.C: void *summation(void *arg){
- ./Wi11/Lectures/Week1/SimpleThread/sum9a.C:// g++ summation.C -lpthread
Creating references in thread callbacks

• The solution is to use a `ref( )`
  ```cpp
  int *primes= ...., vector<int64_t>& candidates = …
  thread(testPrimes, t, ref(candidates), primes)
  ```

  ```cpp
  testPrimes:
  for (i=low;i<hi; i++)
    if (isPrime (candidates[i])){
      _nPrimes++;
      primes[i] = TRUE;
    }
  }
  ```

• Avoids the copying overhead
• Also useful in handling **output parameters**
• Arrays need not be passed via `ref( )`
C++11 provides an extensible array called a vector
vector<int64_t> candidates;
testPrimes:
for (auto i=n; i>0; i--)
    candidates.push_back(atoll(*++argv));
assert(n == candidates.size());

Vector length automatically shrinks or grows as needed, potentially wasting space

Range based for loops may be used to visit array elements without explicitly knowing the length
cout << "Candidate primes: " << endl;
for (auto c : candidates )
    cout << c << endl;

See en.cppreference.com/w/cpp/container/vector
en.cppreference.com/w/cpp/language/range-for
Working with iterators over vectors

• Vector iterators are random-access iterators
  ‣ increment it++, ++it, it--, --it, *it behaves like a pointer
  ‣ Equality and inequality it1 == it2 and it1 != it2
  ‣ Add/subtract constant steps it ± n, it±=n
  ‣ Difference it1 – it2 gives offset
  ‣ Comparison <, >, etc.

• What does this code do?

```cpp
std::vector<int> *keys; int i0, i1;
std::vector<int>::iterator I0 = keys->begin()+i0;
std::vector<int>::iterator I1 = keys->begin()+i1+1;
for (std::vector<int>::iterator it=I0; it!=I1; ++it)
  std::cout << ' ' << *it;
```

©2013  Scott B. Baden / CSE 160 / Fall 2013
Today’s lecture

• Callbacks with ref type arguments
• Sidebar on vectors and iterators
• Performance analysis of parallel merge
• More on synchronization
• Stencil Methods
• Performance Characterization
Recall merge sort

- Divide and conquer algorithm
- Running time $O(N \lg N)$
- Traditional algorithm uses sequential merge, running in time $O(m+n)$, 2 vectors of size $m$ & $n$
- We can partition the merges into smaller ones to reduce the running time

Dan Harvey, S. Oregon Univ.
Parallel Merge Strategy

- We saw that if there are \( N = m + n \) elements, then the larger of the recursive merges processes \( \frac{3}{4}N \) elements.
- Parallelism of merge sort, serial merge: \( \Theta(lg \ n) \)
- Parallelism of parallel merge: \( \Theta(n/lg^2 n) \)
Parallel Merge Strategy

A

0

m/2

m-1

A[0:m/2-1]

A[m/2:m-1]

B

0

n-1

B[0:j]

B[j:n-1]

Binary search

Charles Leiserson
void P_Merge(int *C, int *A, int *B, int m, int n) {
    if (m < n) {
        ... thread(P_Merge,C,B,A,n,m);
    } else if (m == 1) {
        if (n == 0)
            C[0] = A[0];
        else {
            C[0] = (A[0]<B[0]) ? A[0] : B[0];     // Minimum
        }
    } else {
        int m2 = m/2;
        int j = BinarySearch(A[m2], B, n);
        ... thread(P_Merge,C, A, B, m2, j));
        ... thread(P_Merge,C+m2+j, A+m2, B+j, m-m2, nb-j);
    }
}

Parallel Merge Algorithm

Charles Leiserson
Assignment #1

- Implement parallel merge sort with parallel merge
- Stop the recursion at a threshold value of $N_{\text{min}}$
- There is an optimal $N_{\text{min}}$
  - $P = 1$: $N$
  - $P > 1$: $< N$
Today’s lecture

• Callbacks with ref type arguments
• Sidebar on vectors and iterators
• Performance analysis of Merge sort

• More on synchronization
• Stencil Methods
• Performance Characterization
Costs of synchronization

• Locks are expensive, should be used sparingly

local_sum = 0;
for (int64_t i=0; i<N; i++)
    local_sum += x[i];
mtx.lock();
global_sum += local_sum;
mtx.unlock();

N=1M
Running time on 8 threads: 0.001 sec

global_sum = 0;
for (int64_t i=i0; i<i1; i++){
    mtx.lock();
    global_sum += x[i];
    mtx.unlock();
}

Time on 8 threads: 1.29 sec.
Observing data race errors

• If we take out the critical section, we don’t observe program failure!

```
% sum 2 16384    // 16K on 2 threads
The sum of 1 to 16384 is: 134225920
Run took 0.194073 milliseconds
Result verified to be CORRECT.
```

• Scheduler issues affect the outcome
  ‣ Busy waiting or spinning
  ‣ Pre-emption by scheduler forces thread to yield

• Hardware support
  ‣ Test and set
  ‣ Cache coherence protocol provides syncing
Other kinds of data race errors

```c
int64_t  global_sum = 0;

void sumIt(int TID) {
    mtx.lock();
    sum += (TID+1);
    mtx.unlock();
    if (TID == 0)
        cout << "Sum of 1 : " << NT << " = " << sum << endl;
}
```

```
% ./sumIt 5
# threads: 5
The sum of 1 to 5 is 1
After join returns, the sum of 1 to 5 is: 15
```
Barrier synchronization

• Why was the sum reported incorrectly?
• We can’t read a location updated by other threads until it has had the chance to produce its contribution (*true dependence*)
• We can’t overwrite values used by other threads in the current iteration until consumed (*anti-dependence*)
• A barrier provides the synchronization needed to avoid these hazards
• No thread can move past a barrier until all have arrived

```c++
mtx.lock();
sum += 2*(TID+1);
mtx.unlock();
Barrier();
if (TID == 0)
    cout << "Total sum is " << sum << endl;
```
Today’s lecture

• Callbacks with ref type arguments
• Sidebar on vectors and iterators
• Performance analysis of Merge sort
• More on synchronization
• Stencil Methods
• Performance Characterization
Stencil methods

• Many physical problems are simulated on a uniform mesh in 1, 2 or 3 dimensions
• Field variables defined on a discrete set of points
• A mapping from ordered pairs to physical observables like temperature and pressure
• Important applications
  ‣ Differential equations
  ‣ Image processing
Digital Image Representation

RGB representation

Ryan Cuprak

©2013 Scott B. Baden / CSE 160 / Fall 2013
Image smoothing algorithm

- Repeat as many times as necessary

\[
\text{for } (i,j) \text{ in } 0:N-1 \times 0:N-1 \\
I_{\text{new}}[i,j] = \frac{I[i-1,j] + I[i+1,j] + I[i,j-1] + I[i,j+1]}{4} \\
I = I_{\text{new}}
\]
Parallel Implementation

- Partition data into parts, assigning each to a unique thread
- Dependences on values found on neighboring processes
- Threads share boundary values
Multithreaded Smoother()

Global Change, I[:, :], Inew[:, :]
Local mymin = 1 + ($TID * n/$NT),
mymax = mymin + n/$NT-1;
Local done = FALSE;

while (!done) do
    Local myChange = 0;
    Change = 0;
    update Inew and myChange
    Change += myChange;
    if (Change < Tolerance) done = TRUE;
    Swap pointers: I ↔ Inew
end while

update Inew and myChange:
for i = mymin to mymax do
    for j = 1 to n do
        Inew[i,j] = ...
        myChange += (Inew[i,j]−I[−,j])²
    end for
end for

Is this code correct?
Correctness

Global Change, I[:, :], I_{new}[:, :]
Local mymin = 1 + (TID * n/NT),
    mymax = mymin + n/NT-1;
Local done = FALSE;
while (!done) do
    Local myChange = 0;
    BARRIER
    Only on thread 0: Change = 0;
    BARRIER
    update I_{new} and myChange
    CRITICAL SEC: Change += myChange
    BARRIER
    if (myChange < Tolerance) done = TRUE;
    Only on thread 0: Swap pointers: I ← I_{new}
end while

Does this code use minimal synchronization?
Building a linear time barrier with locks

Mutex arrival=UNLOCKED, departure=LOCKED;  // Shared
    int count=0;  // Shared

void Barrier( )
    arrival.lock( );  // atomically count the
    count++;  // waiting threads
    if (count < $NT) arrival.unlock( );
else departure.unlock( );  // last processor
    // enables all to go
    departure.lock( );
    count--;  // atomically decrement
    if (count > 0) departure.unlock( );
else arrival.unlock( );  // last processor resets state
Fin