



Difficulty of Parallel Programming

- Enhanced threads
 - Better scheduling: Cilk
 - Easier synchronization: Transactional memory
- Programs expose parallelism only
 - Runtime system coordinate parallel executions and implement synchronization
 - Programs can expose fine-grained parallelism that does not have to be exploited on a particular running environment
- Automatic parallelization
 - Compiler expose parallelism; runtime system manage the rest
 - Speculative parallelism
- Specialized parallelism
 - Big data: MapReduce/Hadoop
 - Server

3/6/2014

CSC 258/458 - Spring 2014



Cilk (developed at MIT)

An example

```
cilk int fib (int n) {
   if (n < 2)
      return n;
   else {
      int x, y;
      x = spawn fib (n-1);
      y = spawn fib (n-2);
      sync;
      return (x+y);
   }
}</pre>
```

Very much like threads; in fact, Cilk = C silk, silk = nice thread

3/6/2014

CSC 258/458 - Spring 2014



Runtime Task Scheduler

- Scheduler maintains a gueue of tasks for each processor
 - Better locality and less synchronization contention compared to a central task queue
- When out of work, a processor steals some task from another queue with available tasks
 - Load balancing
- Owner processor works on one end of the queue, thief works on the other
 - Less synchronization contention

3/6/2014

CSC 258/458 - Spring 2014

5



Transactional Memory

- Not full-flown parallel programming model, but augment threads programming with simplified synchronization.
- Motivation: locks are hard to manage
 - fine-grained vs. coarse-grained?
- Programmer specifies a group of load and store operations to be executed in an atomic way.
- When atomic groups are too large
 - serializing them limits parallelism
 - speculatively executing them in parallel, detect conflicts and manage them (rollback speculative run)
- Significant research done here by Prof. Michael Scott and students.

3/6/2014

CSC 258/458 - Spring 2014



OpenMP

- Shared-memory parallel programming
- Advantages over threads
 - Simpler programming interface
 - Richer set of primitives, integrated with programming languages
 - Programmers expose parallelism, but do not have to coordinate the parallel execution

3/6/2014

CSC 258/458 - Spring 2014



OpenMP Example

Examples from Wikipedia

```
#pragma omp parallel private(th_id)
{
   th_id = omp_get_thread_num();
   printf("Hello World from thread %d\n", th_id);
   #pragma omp barrier
   if ( th_id == 0 ) {
       nthreads = omp_get_num_threads();
       printf("There are %d threads\n",nthreads);
   }
}
```

3/6/2014

CSC 258/458 - Spring 2014

10

12



Parallel Loop and Scheduling

Parallel loop

```
#pragma omp parallel for
for (i=0; i<n; i++)
    a[i] = 2 * i;</pre>
```

- Independent loop iterations
- Task assignment schemes:
 - Static: loop iterations equally divided between tasks
 - Dynamic: loop iterations assigned (in small units) dynamically
 - Guided (dynamic with variable allocation size)

3/6/2014

CSC 258/458 - Spring 2014



OpenMP-based SOR

Compatible with sequential programs (Incremental parallelism)!

3/6/2014 CSC 258/458 - Spring 2014

-

High-Performance Fortran

- First developed at Rice University
- Parallel extension to Fortran
- Data-parallel model
 - Define an array of data
 - Operate on array elements in parallel

3/6/2014

CSC 258/458 - Spring 2014

11



Automatic Parallelism

- Compiler analyzes dependencies and exposes parallelism.
- Runtime system manages parallel execution and synchronization to observe the dependencies.

3/6/2014

CSC 258/458 - Spring 2014



Automatic Parallelism

Pointers aren't friends of automatic parallelism

int x;
int y;
int* ptr = &y - INPUT;
write(*ptr);
read(x);

3/6/2014

CSC 258/458 - Spring 2014

7

Speculative Parallelization

- An example scenario:
 - The two procedures may run in parallel (no dependency), but I am not sure
 - I am concerned with writes to shared data
 - So if I run them speculatively in parallel and no such writes actually happen, then the parallelization is safe
- Research by Prof. Chen Ding and students:
 - Run the two procedures in separate processes with write protection on shared data
 - If no access fault during runs, we succeed and merge parallel results
- Generalization
 - Expose speculative parallelism
 - Perform checks to validate or discard speculative run
 - Valuable when computing resources are plenty, but parallel programming is difficult

3/6/2014

CSC 258/458 - Spring 2014

14

16



Specialized Parallelism: MapReduce

- Motivation:
 - Parallelism for big data processing
 - Traditional threads/MPI models are challenging to programmers
- Two-stage data processing
 - Data can be divided into many chunks
 - A map task processes input data and generates local results for one or a few chunks
 - A reduce task aggregates and merges local results from multiple map tasks
- Issues
 - Easy to program but limited semantics
 - Need good system support for performance, scalability, and fault tolerance

3/6/2014

CSC 258/458 - Spring 2014

15

13

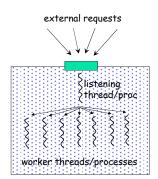


Specialized Parallelism: Server

- Computer application serving (potentially many) interactive clients
 - Parallelism: many requests run concurrently in a server
- But unlike the parallel applications we have seen so far
 - Easily partitioned to fine-grained requests without inter-request dependencies, no need for synchronization
 ⇒ embarrassingly parallel
 - Highly multiprogrammed, many context switches
 - More work in the OS, so OS parallelism matters more
 - Performance (quality-of-service) of each request

3/6/2014

CSC 258/458 - Spring 2014



4