Additional Parallel Programming Models

Kai Shen

Difficulty of Parallel Programming

- So far we have: pthreads, MPI
  - Difficulty in explicitly coordinating parallel executions and implementing synchronization
  - Difficult to handle platform-specific performance issues
- 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

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);
    }
}
```

- Very much like threads

Runtime Task Scheduler

- Scheduler maintains a queue 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
**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

**OpenMP Example**

Examples from Wikipedia

```c
#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);
    }
}
```

**Parallel Loop and Scheduling**

- Parallel loop
  ```c
  #pragma omp parallel for
  for (i=0; i<n; i++)
      a[i] = 2 * i;
  ```
- Independent loop iterations
- Task assignment schemes:
  - Static
  - Dynamic
  - Guided (dynamic with variable allocation size)

**OpenMP-based SOR**

```c
#pragma omp parallel private(j)
{
    #pragma omp for schedule(dynamic, ..)
    for (i=1; i<n; i++)
        for (j=1; j<n; j++)
            temp[i][j] = (grid[i-1][j]+grid[i+1][j]+grid[i][j-1]+grid[i][j+1])/4;
    #pragma omp barrier
    #pragma omp for schedule(dynamic, ..)
    for (i=1; i<n; i++)
        for (j=1; j<n; j++)
            grid[i][j] = temp[i][j];
}
```

Compatible with sequential programs (Incremental parallelism)!
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

Automatic Parallelism

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

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

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

Transactional Memory

- Not full-flown parallel programming model, but augment parallel programming with simplified synchronization.
- Motivation: locks are hard to manage
  - fine-grained vs. coarse-grained?
- Programmer specifies a group of load and store instructions 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.