Concurrent Programming

Session 9: Cilk++

Computer Engineering Department
Iran University of Science and Technology
Tehran, Iran

Lecturer: Nima Ghaemian
Distributed Systems Lab.
Computer Engineering Department,
Iran University of Science and Technology,
nima@comp.iust.ac.ir

Concurrency Platforms

- **Thread-Pool Libraries**
  - .NET's ThreadPool Class
- **Message Passing Libraries**
  - MPI
- **Data Parallel Languages**
  - RapidMind
- **Task Parallel Libraries**
  - TBB
  - Requires a multiprocessors or a multicore system
- **Parallel Linguistic Extensions**
  - OpenMP
  - Cilk++

What is Cilk?

- MIT Cilk
  - Fully Licensed to Cilk Arts Co.
- Extension of C++
  - Few keywords are added
- Multithreading Task Parallelism

Advantages

- Few Primitives!
  - cilk_spawn
  - cilk_sync
  - cilk_for
  - Cilk_main
- Code Simplicity
- Serializability and Re-Engineering
- Work Stealing Run-Time System
  - Few Overheads
- Tool Support
  - Parallel Analyzer
  - CilkScreen

Cilk++ Keywords

- cilk_spawn
  - Creates parallelism by forking off a parallel task.
- cilk_sync
  - There's an implicit one at the end of every function.
  - acts similar to the join system call of pthread library
- cilk_main
  - The replacement for the main function of C++.
- cilk_for
  - A parallel for that requires that the iteration variable be declared inside the for.
  - Iterators may be used as well

Fibonacci Calculation (Sequential)

```
#include <stdio.h>
#include <stdlib.h>
int fib(int n)  {
    if (n < 2) return n;
    else {
        int x = fib(n-1);
        int y = fib(n-2);
 return x + y;}}

 int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    int result = fib(n);
    printf("Fibonacci of %d is %d.\n", n, result);
    return 0; }
```

Fibonacci (POSIX Threads)

```
int fib(int n){
    if (n < 2) return n;
    else {
        int x = fib(n-1);
        int y = fib(n-2);
        return x + y;}}
```

```
typedef struct {
int input;
int output;
} thread_args;
```

```
void *thread_func ( void *ptr )
{
    int i = ((thread_args *) ptr)->input;
    ((thread_args *) ptr)->output = fib(i);
    return NULL;
}
```

Fibonacci (POSIX Threads)

```
int main(int argc, char *argv[]) {
    pthread_t thread;
    thread_args args;
    int status,  result,  thread_result;
    if (argc < 2) return 1;
    int n = atoi(argv[1]);
    if (n < 30) result = fib(n);
    else {
    args.input = n-1;
    status = pthread_create(&thread,NULL,thread_func,(void*) &args );
   // main can continue executing while the thread executes.
    result = fib(n-2);
   // Wait for the thread to terminate.
    pthread_join(thread, NULL);
    result += args.output; }

    printf("Fibonacci of %d is %d.\n", n, result);
    return 0;
}
```

Fibonacci (TBB)

```
long ParallelFib( long n ) {
long sum;
FibTask& a =
    *new(task::allocate_root())
    FibTask(n,&sum);
task::spawn_root_and_wait(a);
return sum;
}

class FibTask: public task {
const long n;
long* const sum;
FibTask( long n_, long* sum_ ) :
n(n_), sum(sum_)
}
```

```
task* execute()
task::execute
if( n<CutOff ) {
    *sum = SerialFib(n);
} else {
    long x, y;
   FibTask& a = *new( allocate_child() )
FibTask(n-1,&x);
   FibTask& b = *new( allocate_child() )
FibTask(n-2,&y);
set_ref_count(3);
    spawn( b );
    spawn_and_wait_for_all( a );
    *sum = x+y;}
return NULL;}};
```

Fibonacci (Cilk++)

```
int fib (int n) {
      if (n<2) return (n);
      else {
        int x,y;
        x = cilk_spawn fib(n-1);
        y = fib(n-2);
        cilk_sync;
        return (x+y);
      }
}
```

Comparison

- As stated before, Cilk++ allows easy transformation of programs to their parallel one.
- Cilk-ified programs also can be easily transformed to their original one.