TaskMan: Simple Task-Parallel Programming in C++
“Let me tell you how it will be…”

Derek Hower & Steve Jackson
University of Wisconsin – Madison
19 December 2007
Executive Summary

- Task-Parallel Programming is gaining steam.
- Existing support in C++ sacrifices programmability for performance
- TaskMan - A task programming interface & runtime
  - Simple interface
  - Feels like serial code
- Results
  - Comparable to existing systems with large tasks
  - Slower with small tasks
    - ...but we haven’t yet applied optimizations!
Task Parallel Programming

- **Task**
  - An *independent* unit of work
  - Typically smaller than a thread
  - Many more tasks than cores

- **Tasks executed by runtime**
  - Schedules and synchronizes tasks
  - Load balancing

- **Examples**
  - Loops with no loop-carried dependence
  - Tree traversal algorithms
  - Recursion
Runtime Characteristics

- Tasks Tuple: \( < \text{func}, \text{arg}_1, \text{arg}_2, ... > \)
  - Stored on a task queue
- Always-present helper threads
- Task Queues
  - Logically global, practically local
  - One per helper thread (i.e. per core)
  - A thread that runs out of local work steals from another queue
Existing Systems

- Threading Building Blocks (TBB)
  - C++ Library from Intel
  - Object-Oriented approach to task programming
  - Task syntax clunky (in our opinion)
Existing Systems

- Threading Building Blocks (TBB)
  - C++ Library from Intel
  - Object-Oriented approach to task programming
  - Task syntax clunky (in our opinion)

- Cilk
  - C compiler & runtime from MIT
  - Task spawns look like function calls
    - Programmer-specified sync points
  - C only, heavyweight
Existing Systems

- Threading Building Blocks (TBB)
  - C++ Library from Intel
  - Object-Oriented approach to task programming
  - Task syntax clunky (in our opinion)

- Cilk
  - C compiler & runtime from MIT
  - Task spawns look like function calls
    - Programmer-specified sync points
  - C only, heavyweight

- Thread Parallel Library (TPL, aka ParallelFX)
  - C# library from Microsoft
  - Task syntax similar to TaskMan
  - Proprietary
  - First preview release came out on December 5
    - No, we haven’t tried it
int fib(int n)
{
    if (n < 2)
        return (n);
    else {
        int x, y;
        x = fib(n-1);
        y = fib(n-2);
        return (x + y);
    }    
}
TaskParallelism

TaskMan

Results

Conclusion

TaskMan Example

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

```c
int fib(int n)
{
    if (n < 2)
        return (n);
    else {
        result<int> x, y;
        x = task(fib, n-1);
        y = task(fib, n-2);
        return (*x + *y);
    }
}
```
TaskMan Implementation

\texttt{task( ... )}

- Push the task on top of thread's work queue, then continue executing
- Extensive use of templates
  - \texttt{task()} can accept any combination of arguments
  - Type safety
  - Explosively verbose error messages
TaskMan Implementation

result<...>

- Represents a *future*
- `operator*` forces the future
  - Pending tasks are evaluated until result is ready
  - Once launched, a task never leaves its thread

+ Simple approach, no need for continuation passing
- Potentially deep recursions
Results

- microbenchmark: stat
- Converted Cilk benchmarks: heat, plu, matmul
- Othello AI

Unless otherwise noted, performance numbers are for an 8 core Intel system.
Microbenchmark: Statistically Distributed Task Sizes

Create and run no-op tasks that take time \( t \) to complete, where \( t \) is produced via a statistical distribution.
Converted Cilk Benchmark: plu

PLU - 2048

Speedup

# threads
Converted Cilk Benchmark: heat

Heat - 4096x4096 200 t

- TBB
- Cilk
- TaskMan
- Perfect

Speedup vs. # threads
Converted Cilk Benchmark: Matrix Multiply

Matmul - 1500

Speedup vs # threads

- TBB
- Cilk
- TaskMan
- Perfect
Othello Benchmark

- A recursive minimax AI for the game Othello (Reversi)
- Two different board evaluators:
  - Simple: evaluation function is a count of pieces on the board → shorter tasks
  - Strategic: evaluation function considers board position (corners, edges, etc.) → longer tasks
Othello vs. TBB

Task Parallelism

Results

Conclusion
Othello on Niagara: 8 cores x 4 threads = 32 threads
Future Directions

- Optimize TaskMan for performance
- Side-by-side comparison of work queue implementations
  - Lock-free structures?
  - Transactional memory?
  - Dedicated task-management hardware?
- Extend programming model
  - e.g. parallel loops
  - But avoid needlessly complex syntax
  - Compiler may become necessary
Concluding Remarks

- Task parallelism is a useful programming model
  - Much easier to write than raw pthreads code!
  - Particularly well-suited to certain problems
    - (And not for certain others)
- The work-stealing task queue algorithm supports this model
  - A simple, untuned implementation can achieve significant speedup
  - Optimized implementations are still better
But, there is beauty in simplicity:
Backup: Fibonacci in Cilk

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);
    }
}
class FibTask: public task {

public:
    int* const sum;
    const int n;

FibTask( long _n, long* _sum ): sum(_sum), n(_n) {} 

    task* execute() {
        int 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;
};