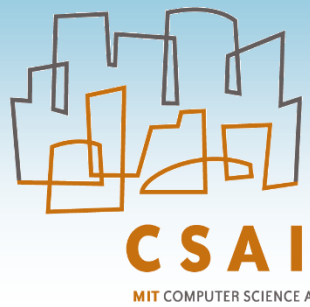


FROM

MODERN ALGORITHMS WORKSHOP

Parallel Algorithms

Prof. Charles E. Leiserson

Dr. Tao B. Schardl

September 19, 2018

Outline

- Introduction
- Cilk Model
- Detecting Nondeterminism
- What Is Parallelism?
- Scheduling Theory Primer
- *Lunch Break*
- Analysis of Parallel Loops
- Case Study: Matrix Multiplication
- Case Study: Jaccard Similarity
- Post-Moore Software

CILK MODEL

History of Cilk

1994–2006	Cilk project formed at MIT LCS. Cilk offers simple C-based multithreaded programming combined with execution efficiency.
2006–2009	Cilk Arts, Inc., spun out of MIT CSAIL. Cilk++ provides support for C++, parallel loops, and reducer hyperobjects.
2009–2014	Intel Corporation acquires Cilk Arts. Cilk Plus offers Cilk++ and vector operations in ICC and GCC.
2014–2017	Due to attrition in Intel's Cilk team, the development of Cilk at Intel stagnates.
2017	Intel announces it is dropping support for Cilk Plus, and GCC follows suit.
2018	Cilk Hub is formed to support and develop a new Open Cilk platform.

Components of Open Cilk

Component	
Language	Cilk++ → linguistic enhancements
Compiler	Based on Tapir/LLVM
Runtime	Cilk Plus → new
Instrumentation	Generic compiler instrumentation based on CSI
Productivity tools	Cilksan race detector, Cilkscale scalability analyzer

Nested Parallelism in Cilk

```
int64_t fib(int64_t n) {
  if (n < 2) {
    return n;
  } else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

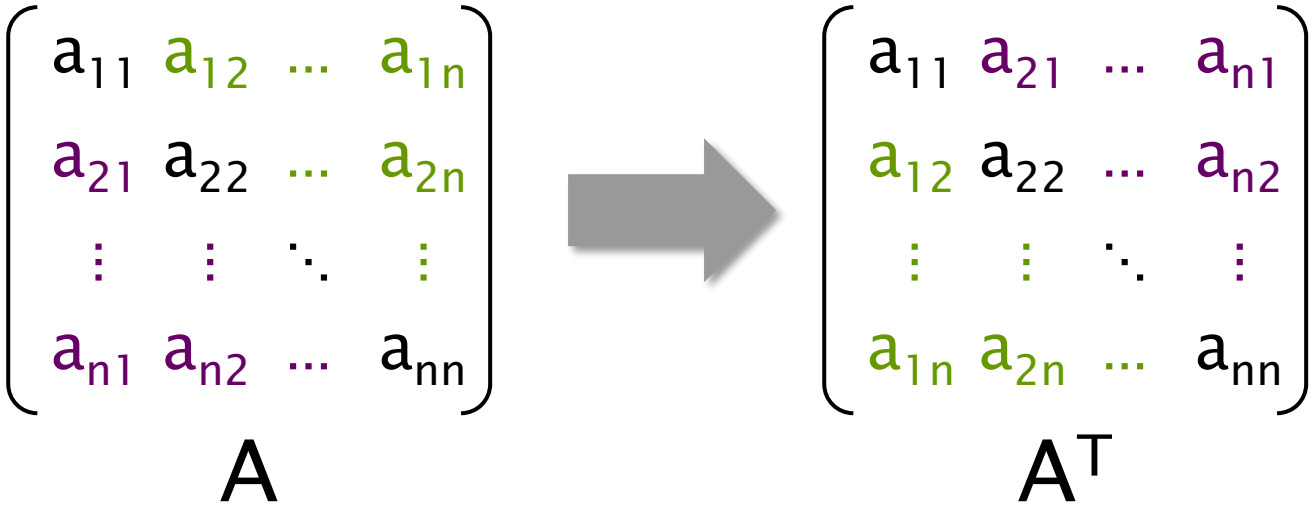
The named child function may execute in parallel with the parent caller.

Control cannot pass this point until all spawned children have returned.

Cilk keywords grant permission for parallel execution. They do not command parallel execution.

Loop Parallelism in Cilk

Example: In-place matrix transpose



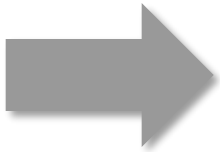
The iterations of a `cilk_for` loop execute in parallel.

```
// indices run from 0, not 1
cilk_for (int i=1; i<n; ++i) {
  for (int j=0; j<i; ++j) {
    double temp = A[i][j];
    A[i][j] = A[j][i];
    A[j][i] = temp;
  }
}
```

Serial Semantics

Cilk source

```
int64_t fib(int64_t n) {
  if (n < 2) {
    return n;
  } else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```



serial projection

```
int64_t fib(int64_t n) {
  if (n < 2) {
    return n;
  } else {
    int64_t x, y;
    x = fib(n-1);
    y = fib(n-2);

    return (x + y);
  }
}
```

The serial projection of a Cilk program is always a legal interpretation of the program's semantics.

Remember, Cilk keywords grant permission for parallel execution. They do not command parallel execution.

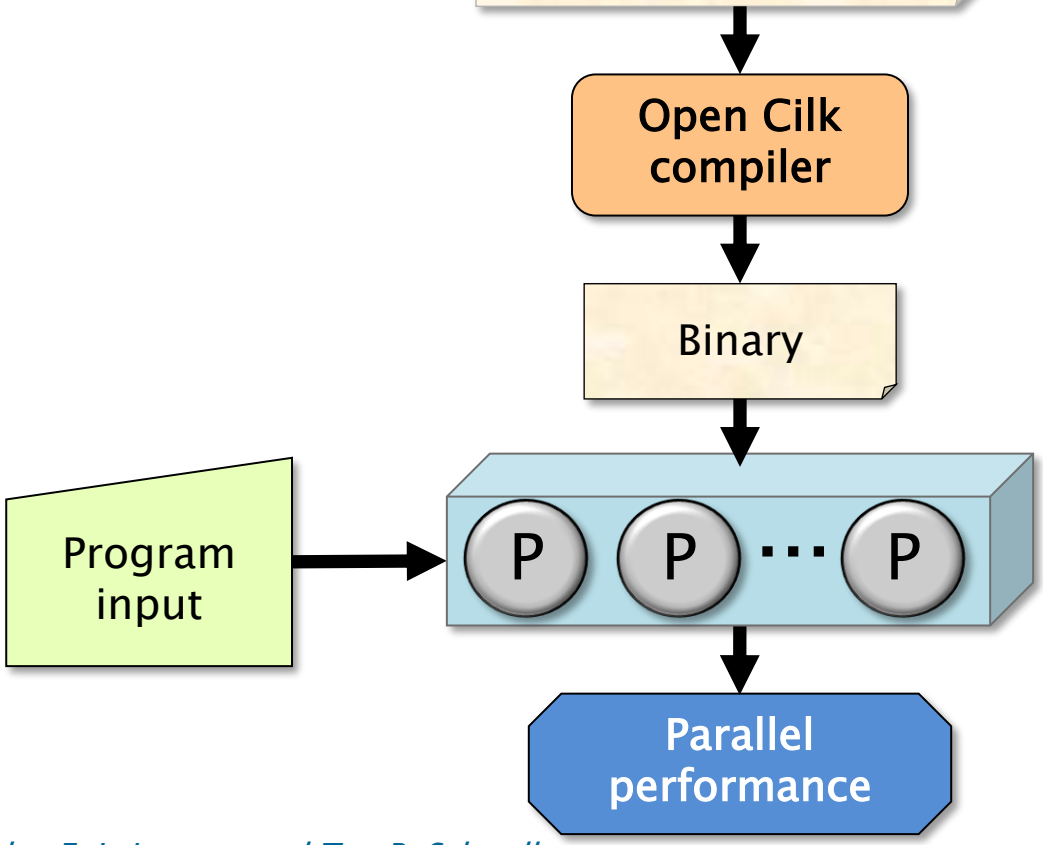
To obtain the serial projection:

```
#define cilk_for for
#define cilk_spawn
#define cilk_sync
```

Cilk Platform

```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

Cilk source



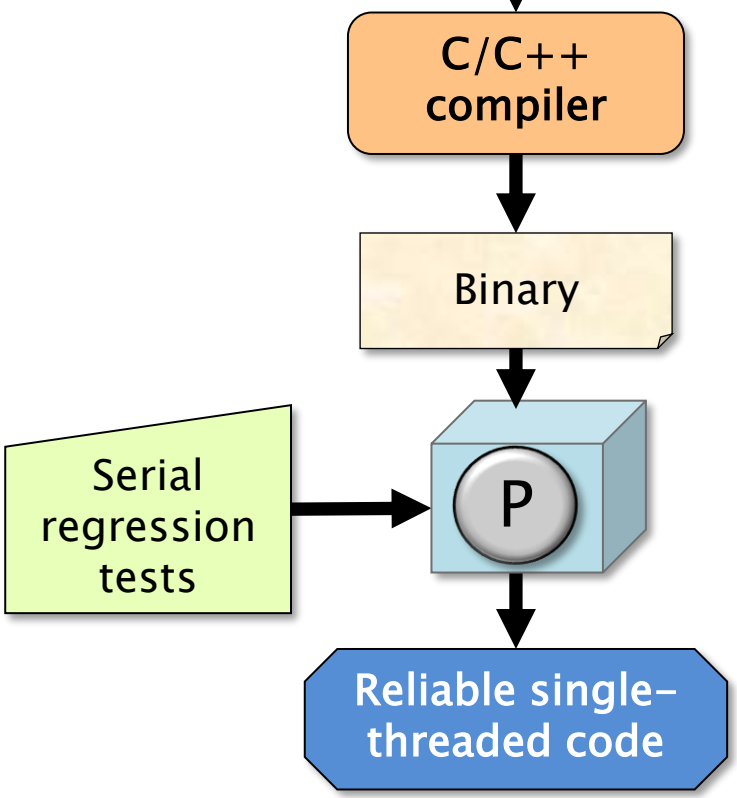
Serial Testing

```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

Cilk source

```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x = fib(n-1);
    int64_t y = fib(n-2);
    return (x + y);
  }
}
```

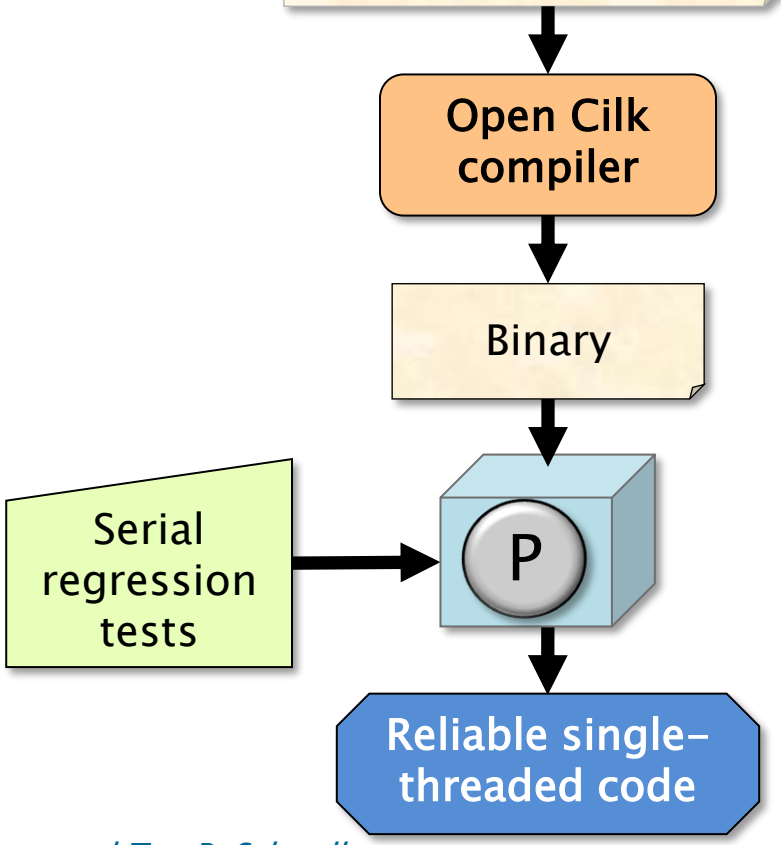
Serial projection



One-Processor Testing

```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

Cilk source

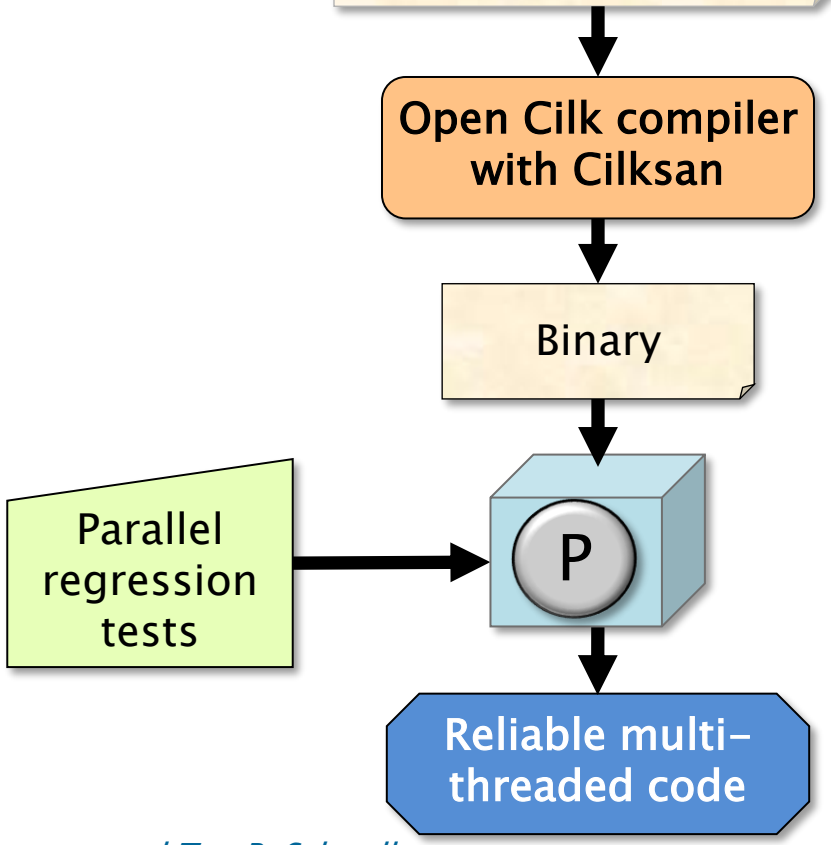


The parallel program executing on one core should behave exactly the same as the execution of the serial projection.

Parallel Testing

```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

Cilk source



Cilksan finds and localizes race bugs.

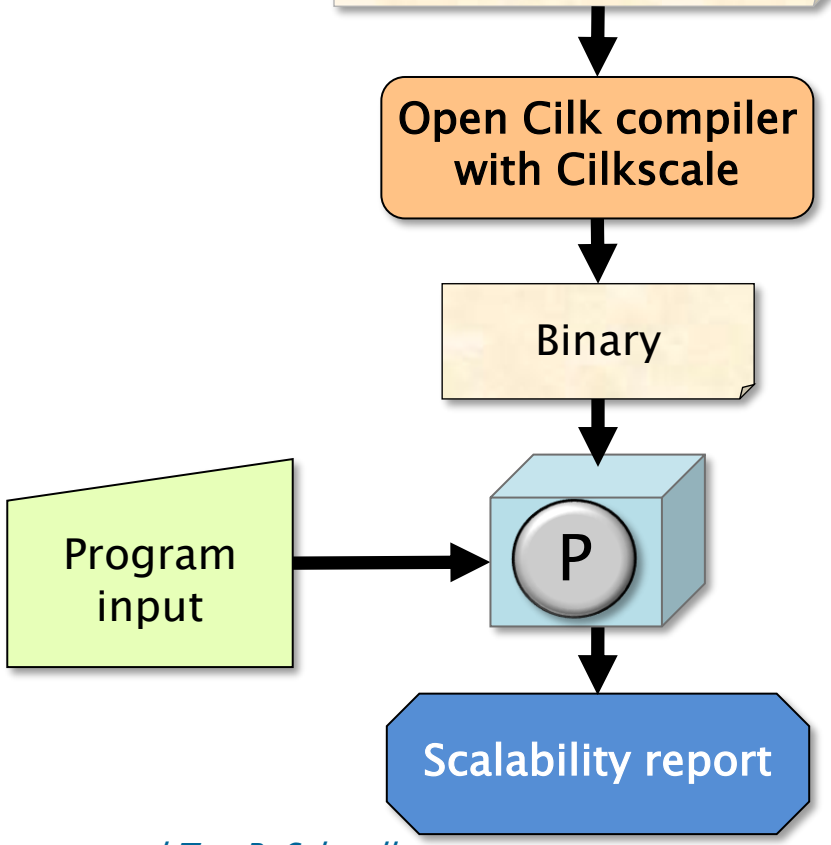
Preview: Cilksan Output

```
$ cilksan ./mm_dac
…
Race detected at address 0x65c070
  Write access to C (declared at mm/mm_dac.c:27)
             from 0x400ed0 mm_base mm/mm_dac.c:34:15
      Called from 0x401868 mm_dac mm/mm_dac.c:57:5
      Called from 0x4025d0 mm_dac mm/mm_dac.c:63:5
     Spawned from 0x401548 mm_dac mm/mm_dac.c:63:5
      Called from 0x4025d0 mm_dac mm/mm_dac.c:63:5
     Spawned from 0x401548 mm_dac mm/mm_dac.c:63:5
  Read access to  C (declared at mm/mm_dac.c:27)
             from 0x400e27 mm_base mm/mm_dac.c:34:15
      Called from 0x401868 mm_dac mm/mm_dac.c:57:5
  Common calling context
      Called from 0x401c02 main mm/mm_dac.c:85:3

0.686637

Race detector detected total of 47 races.
Race detector suppressed 147409 duplicate error messages
```

Scalability Analysis

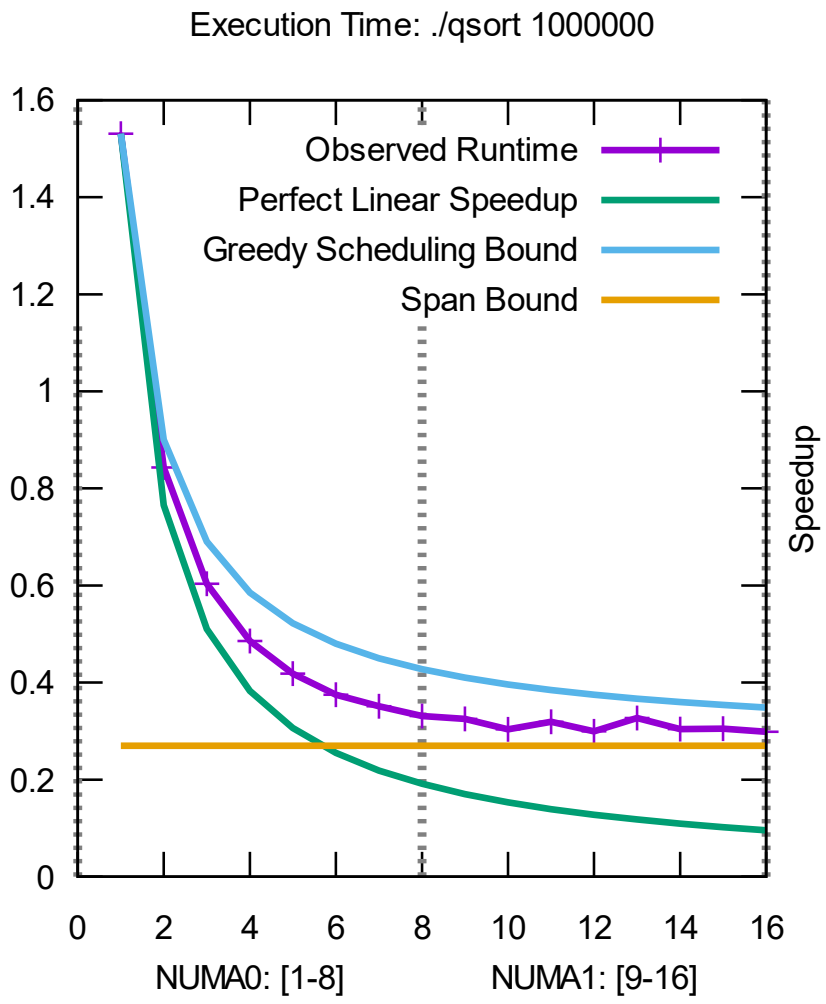
```
int64_t fib(int64_t n) {
  if (n < 2) { return n; }
  else {
    int64_t x, y;
    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return (x + y);
  }
}
```

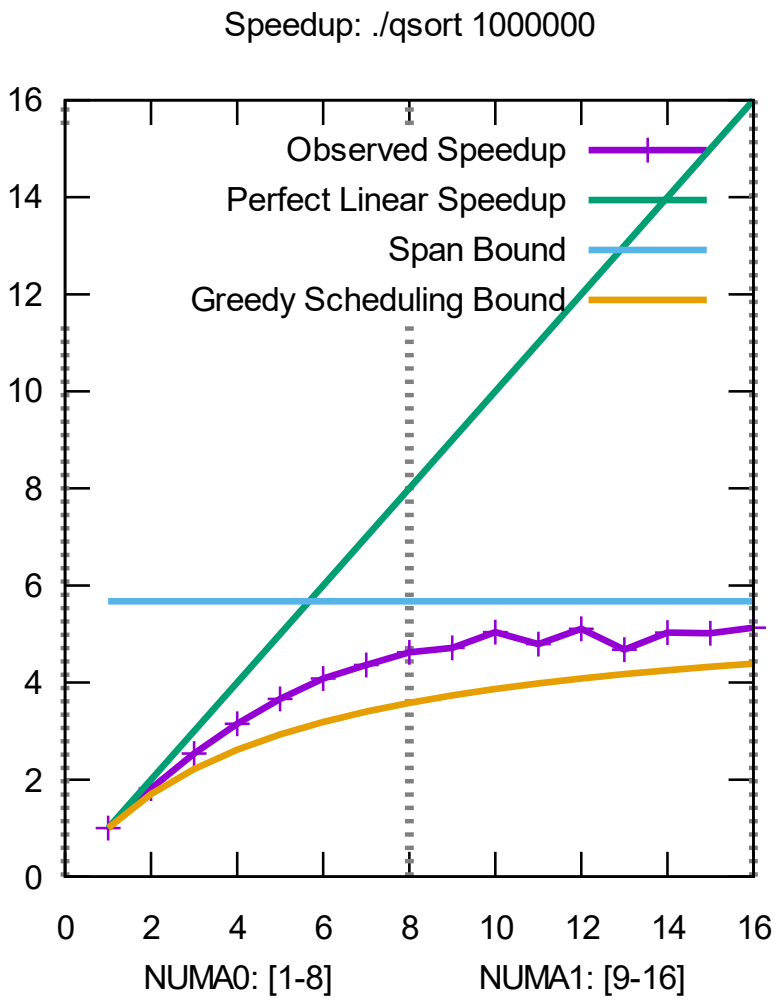
Cilk source



Cilkscale analyzes how well your program will scale to larger machines.

Preview: Cilkscale Output





Hands-On: Cilk Programming

Setup

1. Find a buddy to do pair programming.
2. Start up a Jupyter notebook, and start a new Terminal in that notebook.
3. At the terminal, add the "tapir-6.0.0" module:

```
$ module add ./tapir-6.0.0
```

4. Compile and run the serial quicksort code in the `qsort` directory:

```
$ cd qsort
$ clang -O3 qsort.c -o qsort
$ ./qsort 1000000
```

Exercise: Getting Started with Cilk

5. Open `qsort/qsort.c` and parallelize the code using `cilk_spawn` and `cilk_sync` as follows:

```
static void quicksort(int64_t *begin, int64_t *end) {
  …
  cilk_spawn quicksort(middle + 1, end);
  quicksort(begin, middle);
  cilk_sync;
}
```

6. Add "`#include <cilk/cilk.h>`" to the top of `qsort.c`.
7. Recompile and rerun `qsort`. The program should now run faster.

```
$ clang -O3 -fcilkplus qsort.c -o qsort
$ ./qsort 1000000
```

Compiler flag to enable Cilk.