Shared-memory Parallel Programming with Cilk Plus

Milind Chabbi

Nikhil Hegde

Department of Computer Science
IIT Dharwad

CS410 Lecture 10 26 January 2024

Outline for Today

- **Data race detection with CilkSan**
- **Assessing Cilk performance with CilkScale**

Concurrency Cautions

- Only limited guarantees between descendants or ancestors
  - —DAG precedence order maintained and nothing more
  - —don’t assume atomicity between different procedures!

Race Conditions

- Data race
  - —two parallel strands access the same data
  - —at least one access is a write
  - —no locks held in common
- General determinacy race
  - —two parallel strands access the same data
  - —at least one access is a write
  - —a common lock protects both accesses

CilkSan

- Detects and reports data races when program terminates
  - —finds all data races even those by third-party or system libraries
- Does not report determinacy races
  - —e.g. two concurrent strands use a lock to access a queue
    - – enqueue & dequeue operations could occur in different order

      potentially leads to different result

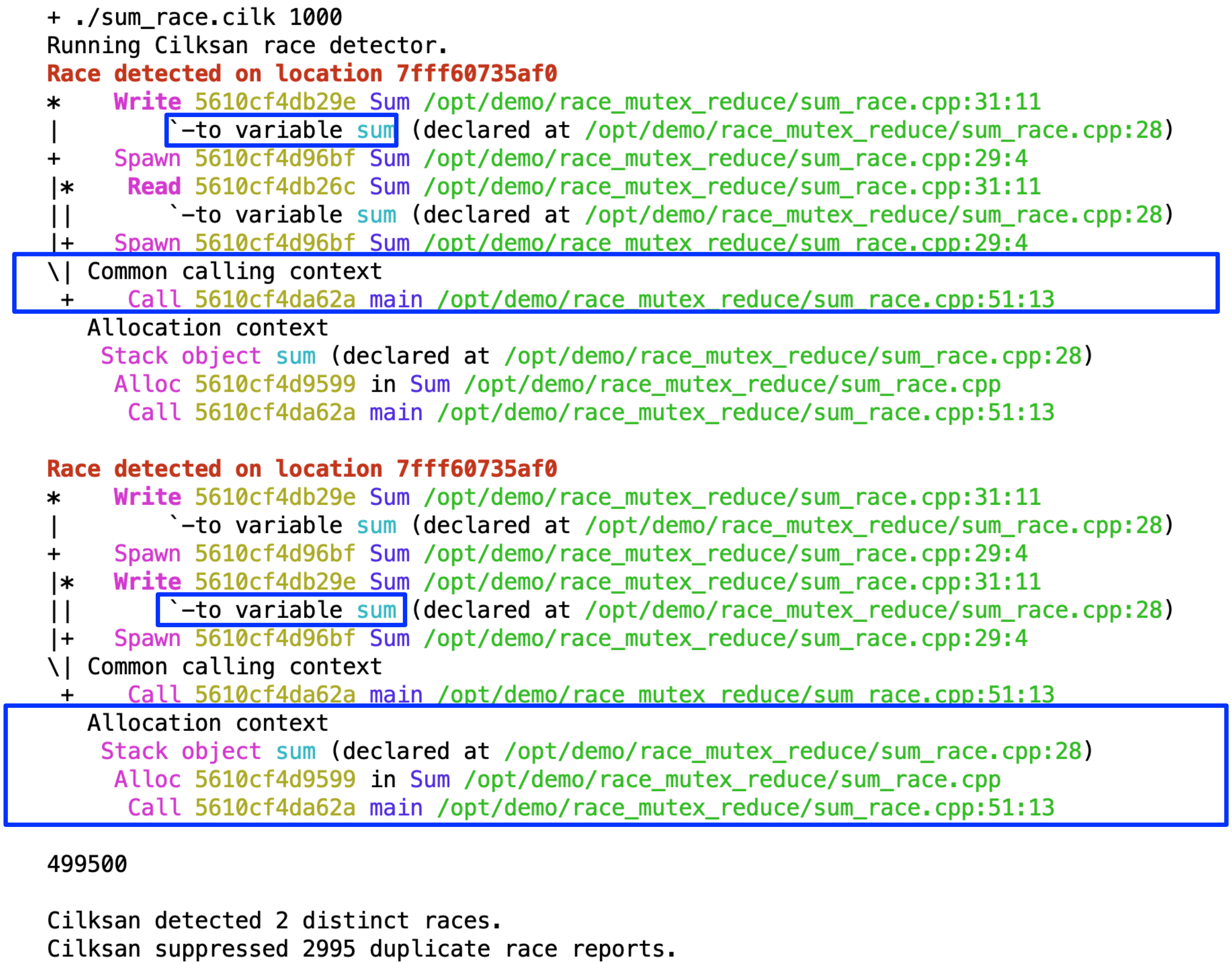
Race Detection Strategies in CilkSan

- Lock covers
  - —two conflicting accesses to a variable don’t race if some lock L is held while each of the accesses is performed by a strand
- Access precedence
  - —two conflicting accesses do not race if one must precede the other
    - access A is by a strand X, which precedes the cilk_spawn of strand Y which performs access B
    - access A is performed by strand X, which precedes a cilk_sync that is an ancestor of strand Y

CilkSan Limitations

- Only detects races between Cilk strands
  - —depends upon their strict fork/join paradigm
- Only detects races that occur given the input provided
  - —does not prove the absence of races for other inputs
  - —choose your testing inputs carefully!
- Runs serially, 15-30x slower
- Increases the memory footprint of an application
  - —could cause an error if memory demand is too large
- If you build your program with debug information (compile with -g), CilkSan will associate races with source line numbers

CilkSan Output



```
+ ./sum_race.cilk 1000
Running Cilksan race detector.
Race detected on location 7fff60735af0
*    Write 5610cf4db29e Sum /opt/demo/race_mutex_reduce/sum_race.cpp:31:11
|        `-to variable sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
+    Spawn 5610cf4d96bf Sum /opt/demo/race_mutex_reduce/sum_race.cpp:29:4
|*    Read 5610cf4db26c Sum /opt/demo/race_mutex_reduce/sum_race.cpp:31:11
||       `-to variable sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
|+   Spawn 5610cf4d96bf Sum /opt/demo/race_mutex_reduce/sum_race.cpp:29:4
\| Common calling context
 +    Call 5610cf4da62a main /opt/demo/race_mutex_reduce/sum_race.cpp:51:13
   Allocation context
    Stack object sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
     Alloc 5610cf4d9599 in Sum /opt/demo/race_mutex_reduce/sum_race.cpp
      Call 5610cf4da62a main /opt/demo/race_mutex_reduce/sum_race.cpp:51:13

Race detected on location 7fff60735af0
*    Write 5610cf4db29e Sum /opt/demo/race_mutex_reduce/sum_race.cpp:31:11
|        `-to variable sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
+    Spawn 5610cf4d96bf Sum /opt/demo/race_mutex_reduce/sum_race.cpp:29:4
|*   Write 5610cf4db29e Sum /opt/demo/race_mutex_reduce/sum_race.cpp:31:11
||       `-to variable sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
|+   Spawn 5610cf4d96bf Sum /opt/demo/race_mutex_reduce/sum_race.cpp:29:4
\| Common calling context
 +    Call 5610cf4da62a main /opt/demo/race_mutex_reduce/sum_race.cpp:51:13
   Allocation context
    Stack object sum (declared at /opt/demo/race_mutex_reduce/sum_race.cpp:28)
     Alloc 5610cf4d9599 in Sum /opt/demo/race_mutex_reduce/sum_race.cpp
      Call 5610cf4da62a main /opt/demo/race_mutex_reduce/sum_race.cpp:51:13

499500

Cilksan detected 2 distinct races.
Cilksan suppressed 2995 duplicate race reports.
```

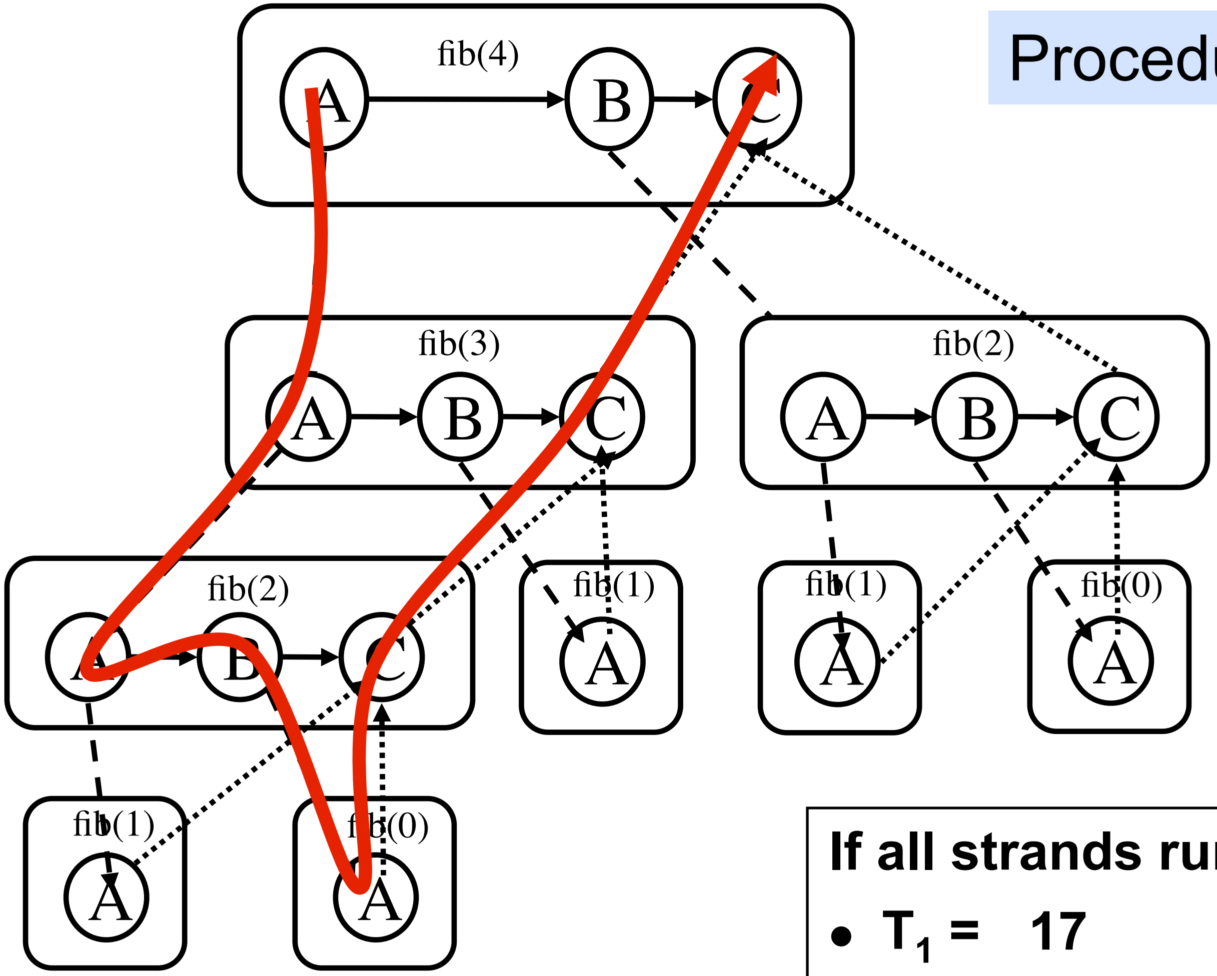
CilkSan Demo

- **Explore CilkSan race detection**
  - `cd ~/software/demo/race_mutex_reduce`
  - `cilk_race.sh:`
    - `Uses -fopencilk -fsanitize=cilk`
    - `Run with CILK_NWORKERS=1`

Performance Measures

- T_s = serial execution time
- T_1 = execution time on 1 processor (total work), $T_1 \geq T_s$
- T_p = execution time on P processors
- T_∞ = execution time on infinite number of processors
  - longest path in DAG
    - length reflects the cost of computation at nodes along the path
  - known as "critical path length"

Work and Critical Path Example



Procedure oriented view

If all strands run in unit time

- **T_1 = 17**
- **T_∞ = 8 (critical path length)**

CilkScale - Scalability Analysis Tool

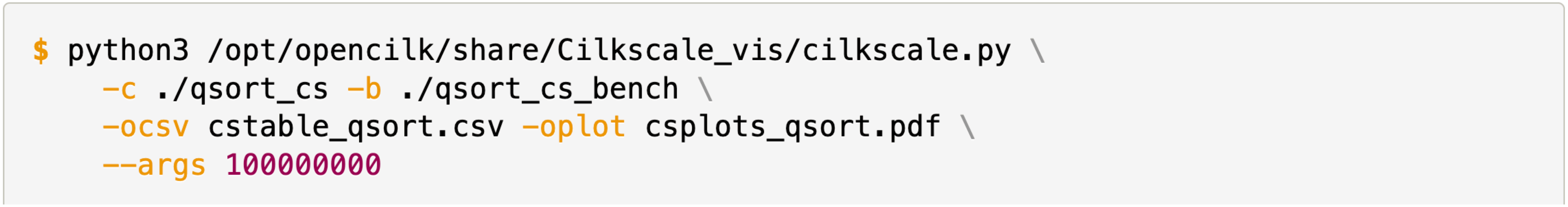
- Collects stats of parallel performance
- Measures work, span, parallelism
- Automatically benchmarks program in different number of processors
- Produce tables and graphical plots with the above performance and scalability measurements
- Predicts speedup on various numbers of processors based on work and span

CilkScale - Scalability Analysis Tool

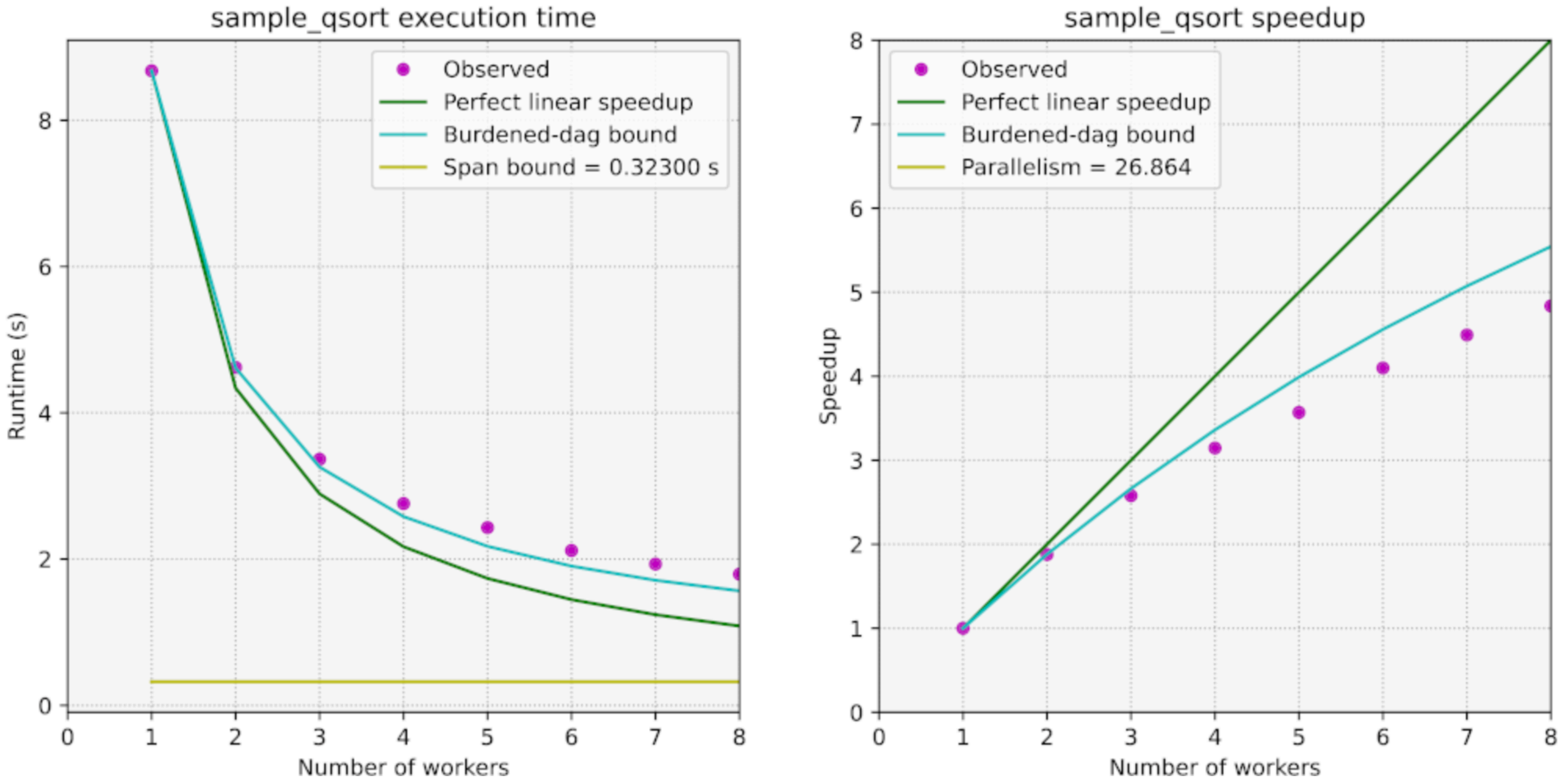
- Instruments Cilk program to collect performance measurements during its execution
- Cilkscale instrumentation operates in one of two modes
  - *Benchmarking* mode: Cilkscale measures the wall-clock execution time of your program
    - compile with -fopencilk -fcilktool=cilkscale-benchmark
  - *Work/span analysis* mode: Cilkscale measures the *work*, *span*, and *parallelism* of your program
    - compile with -fopencilk -fcilktool=cilkscale
- By default, Cilkscale only reports performance results for whole-program execution
  - fine-grained analyses of specific sub-computations is possible

Automatic benchmarks and visualization

ref: https://www.opencilk.org/doc/users-guide/cilkscale/



```
$ python3 /opt/opencilk/share/Cilkscale_vis/cilkscale.py \
    -c ./qsort_cs -b ./qsort_cs_bench \
    -ocsv cstable_qsort.csv -oplot csplots_qsort.pdf \
    --args 100000000
```



Rule of Thumb

As a rule of thumb, the parallelism of a computation is deemed sufficient if it is about 10x larger (or more) than the number of available processors. On the other hand, if the parallelism is too high — say, several orders of magnitude higher than the number of processors — then the overhead for spawning tasks that are too fine-grained may become substantial

Burdened Dag Bound:

Slowed execution if every continuation incurs a steal (incurs overhead of parallelism)

CilkScale Demo

- **Explore CilkScale for performance analysis using fib example**
  - `cilkscale_run.sh fib 47`
  - `cilkscale_run.sh fib_cutoff 47 2`
  - `cilkscale_run.sh fib_cutoff 47 5`
  - `cilkscale_run.sh fib_cutoff 47 10`
  - `cilkscale_run.sh fib_cutoff 47 20`

References - I

- **Matteo Frigo, Charles Leiserson, and Keith Randall. The implementation of the Cilk-5 multithreaded language. In PLDI (Montreal, Quebec, Canada, June 17 - 19, 1998), 212-223.**

- **Mingdong Feng and Charles E. Leiserson. 1997. Efficient detection of determinacy races in Cilk programs. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures* (SPAA '97). ACM, New York, NY, USA, 1-11.**

- **Guang-Ien Cheng, Mingdong Feng, Charles E. Leiserson, Keith H. Randall, and Andrew F. Stark. 1998. Detecting data races in Cilk programs that use locks. In *Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures* (SPAA '98). ACM, New York, NY, USA, 298-309.**

References - II

- Yuxiong He, Charles E. Leiserson, and William M. Leiserson. 2010. The Cilkview scalability analyzer. In Proc. of the 22nd annual ACM symposium on Parallelism in algorithms and architectures (SPAA '10). ACM, New York, NY.
- Charles E. Leiserson. Cilk LECTURE 1. Supercomputing Technologies Research Group. Computer Science and Artificial Intelligence Laboratory. http://bit.ly/mit-cilk-lec1
- Matteo Frigo, Pablo Halpern, Charles E. Leiserson, and Stephen Lewin-Berlin. Reducers and other Cilk++ hyperobjects. SPAA '09, 79-90. Talk Slides. April 11, 2009. http://bit.ly/reducers
- Charles Leiserson, Bradley Kuzmaul, Michael Bender, and Hua-wen Jing. MIT 6.895 lecture notes - Theory of Parallel Systems. http://bit.ly/mit-6895-fall03
- Intel Cilk++ Programmer’s Guide. Document # 322581-001US.