Introduction to CILK

The slides are based on:

"Multithreaded Programming in Cilk" by **Charles E. Leiserson**

http://gamma.cs.unc.edu/SC2007/Leiserson-SC07-Workshop.pdf

and some of the slides are mine (Guy Tel-Zur)

Version ~~24/5/2015~~, ~~23/5/2016~~, ~~20/12/2020~~, ~~9/5/2021~~, 18/12/2021

Cilk

A C language for programming dynamic multithreaded applications on shared-memory multiprocessors.

Example applications:

- virus shell assembly
- graphics rendering
- *n*-body simulation
- heuristic search
- dense and sparse matrix computations
- friction-stir welding simulation
- artificial evolution

Guy: http://software.intel.com/en-us/intel-cilk-plus

Intel® Developer Zone:

Intel® Cilk™ Plus

OVERVIEW | OBTAIN | LEARN | SUPPORT

A Quick, Easy and Reliable way to Improve Performance

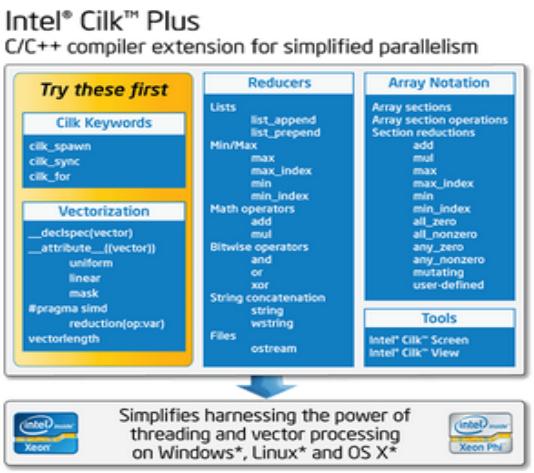
Intel® Cilk™ Plus is an extension to C and C++ that offers a quick and easy way to harness the power of both multicore and vector processing. The three Intel Cilk Plus keywords provide a simple yet surprisingly powerful model for parallel programming, while runtime and template libraries offer a well-tuned environment for building parallel applications.

Click here for sample code, contributed libraries, open specifications and other information from the Cilk Plus community.

Go to "**Intel® C++ Compiler Code Samples**" to see real-world applications that utilize the Intel® Cilk™ Plus.

Intel Cilk Plus allows you to:

- Write parallel programs using a simple model: With only three keywords to learn, C and C++ developers move quickly into the parallel programming domain.
- Identify data parallelism by using simple array notations that include elemental function capabilities.
- Leverage existing serial tools: The serial semantics of Intel Cilk Plus allows you to debug in a familiar serial debugger.
- Scale for the future: The runtime system operates smoothly on systems with hundreds of cores. Tools are available to analyze your application and predict how well it will scale.



Available in:

Intel® Parallel Studio XE
Intel® System Studio
Intel® Integrated Native Developer Experience 2015 Build Edition for OS X

OS Support

Windows*
Linux*
OS X*

Languages

C, C++

Related Content

Documentation
Tutorial
Forums
Blogs

Guy: http://cilkplus.org/

האתר הזה חדל מלהתקיים...

LEARN DOWNLOAD NEWS CONTRIBUTE LOGIN

Search -Any-

Intel ®
Cilk™ Plus

Why Use Intel® Cilk™ Plus?

Why Use it?

Intel® Cilk™ Plus is the easiest, quickest way to harness the power of both multicore and vector processing.

What is it?

Intel Cilk Plus is an extension to the C and C++ languages to support data and task parallelism.

Primary Features

High Performance:

- An efficient work-stealing scheduler provides nearly optimal scheduling of parallel tasks
- Vector support unlocks the performance that's been hiding in your processors
- Powerful hyperobjects allow for lock-free programming

Easy to Learn:

THE UNIVERSITY OF TEXAS AT AUSTIN

"I recently tried the updated Intel® Cilk™ Plus in the Intel® C++ Compiler. I liked the lower overhead from using Cilk Plus spawning compared to that of OpenMP* task. I'm looking forward to using Cilk Plus Array Notations. The concepts behind Cilk Plus – simplification of adding parallelism – is really great. Thanks for offering this easy-to-use capability!"

David Carver
Texas Advanced Computing Center

More Testimonials

Intel® Cilk™ Plus

C/C++ compiler extension for simplified parallelism

Try these first

Cilk Keywords

- cilk_spawn
- cilk_sync
- cilk_for

Vectorization

- __declspec(vector)
- __attribute__((vector))
  - uniform
  - linear
  - mask
- #pragma simd
  - reduction(op:var)
- vectorlength

Reducers

- Lists
  - list_append
  - list_prepend
- Min/Max
  - max
  - max_index
  - min
  - min_index
- Math operators
  - add
  - mul
- Bitwise operators
  - and
  - or
  - xor
- String concatenation
  - string
  - wstring
- Files
  - ostream

Array Notation

- Array sections
- Array section operations
- Section reductions
  - add
  - mul
  - max
  - max_index
  - min
  - min_index
  - all_zero
  - all_nonzero
  - any_zero
  - any_nonzero
  - mutating
  - user-defined

Tools

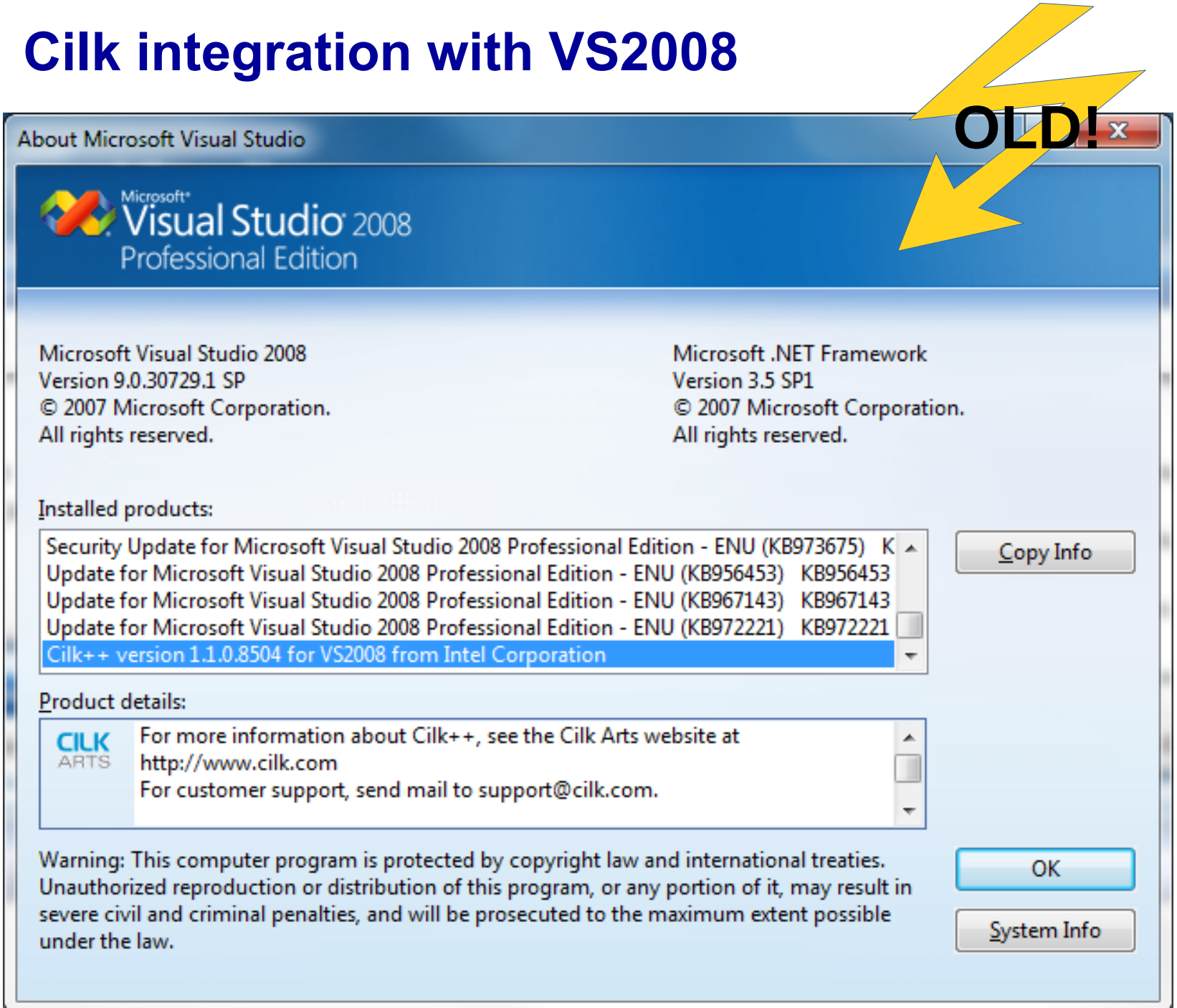
- Intel® Cilk™ Screen
- Intel® Cilk™ View

intel inside Xeon

Simplifies harnessing the power of threading and vector processing on Windows*, Linux* and OS X*

intel inside Xeon Phi

Cilk integration with VS2008



A quick, easy and reliable way to improve threaded performance

Intel® Cilk™ Plus

Intel® Cilk™ Plus is an extension to C and C++ that offers a quick, easy and reliable way to improve the performance of programs on multicore processors.

Intel® Cilk™ Plus is now available in open-source and for GCC 4.7!
Read more here.

Intel® Cilk™ Plus is an exte
quick, easy and reliable way to improve t
multicore processors. The three Intel Cilk
surprisingly powerful model for parallel pr
template libraries offer a well-tuned envir
applications. Intel Cilk Plus allows you to:

- Write parallel programs using a simple
learn, C and C++ developers move qui
domain.
- Utilize data parallelism by simple array notations that include elemental function capabilities.
- Leverage existing serial tools: The serial semantics of Intel Cilk Plus allows you to debug in a familiar serial debugger.
- Scale for the future: The runtime system operates smoothly on systems with hundreds of cores.

As multicore systems become prevalent on desktops, servers and even laptop systems, new performance leaps will come as the industry adopts parallel

17/8/2011

Learn

- Intel® Cilk™ Plus Evaluation Guide
- ntroduction to Intel® Cilk™ Plus
- arallel Composer 2011 Getting Started Guide
s Intel Cilk Plus tutorial)
- al functions: Writing data parallel code in
using Intel® Cilk™ Plus whitepaper
- k Plus User Forum
- Intel Cilk Plus Support Page

Obtain

The Intel Cilk Plus extension to C and C++ is now available in the "cilkplus" branch of GCC 4.7. Contributions are welcome! The Intel Cilk Plus runtime source kit is now available as well.

Intel Cilk Plus is available in Intel® Parallel Building Blocks which is supported by:

Intel Cilk Plus is being deprecated

2017

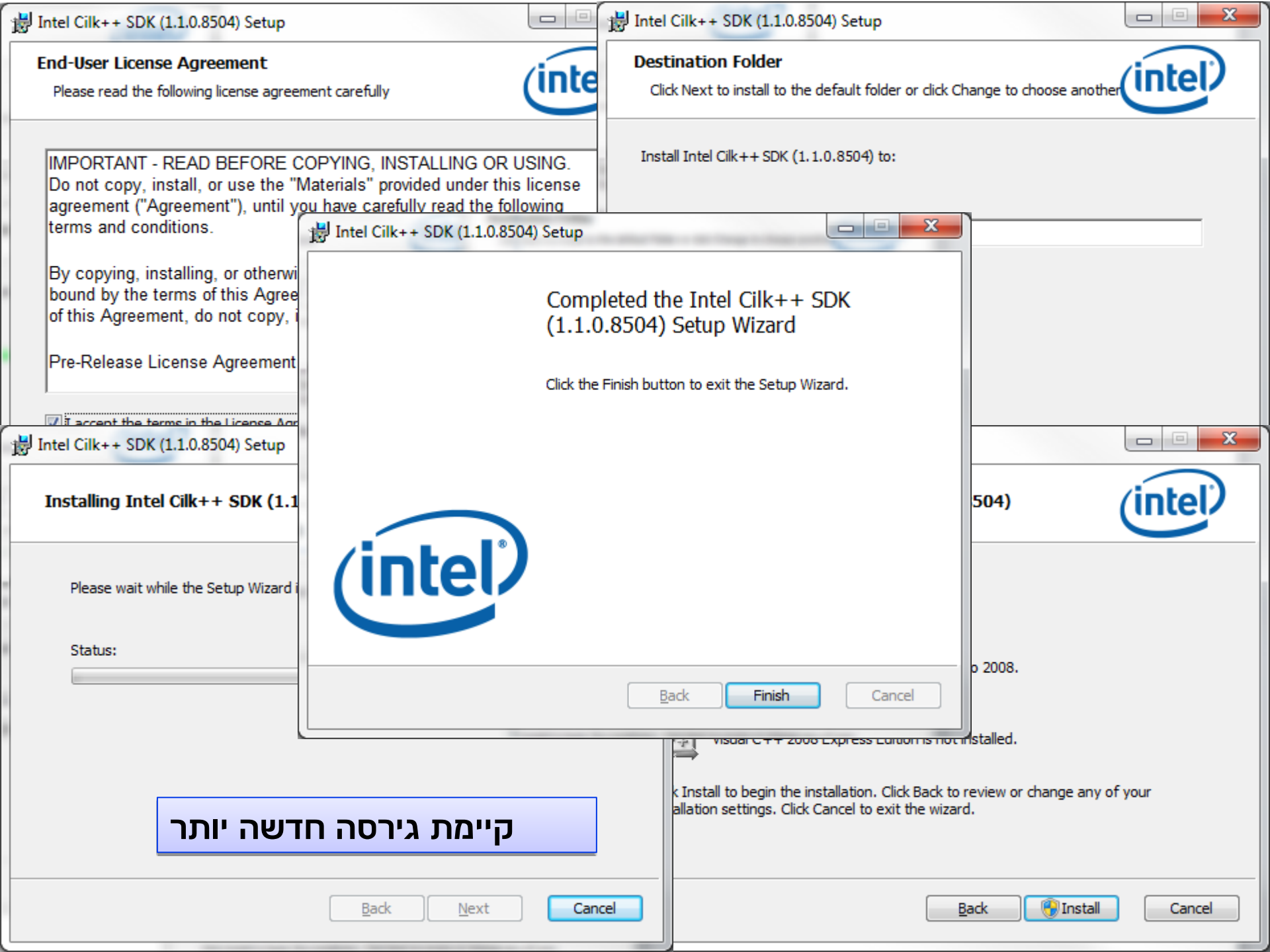
Hansang B. (Intel) Wed, 09/20/2017 - 08:03

Intel® Cilk™ Plus - an extension to the C and C++ languages to support data and task parallelism - is being deprecated in the 2018 release of Intel® Software Development Tools. It will remain in deprecation mode in the Intel® C++ Compiler for an extended period of two years. It is highly recommended that you start migratin
TBB). For more information see Migrate Yo
technology continues at MIT's Cilk Hub.

Announcements

Cilk Deprecation

Intel® Cilk™ Plus – an extension to the C and C++ languages to support data and task parallelism – is being deprecated in the 2018 release of Intel® Software Development Tools. It will remain in deprecation mode in the Intel® C++ Compiler for an extended period of two years. It is highly recommended that you start migrating to standard parallelization models such as OpenMP* and Threading Building Blocks (TBB). For more information see Migrate Your Application to use OpenMP* or TBB Instead of Intel® Cilk™ Plus. Research into Cilk technology continues at MIT's Cilk Hub.


Intel Cilk++ SDK (1.1.0.8504) Setup
End-User License Agreement
Please read the following license agreement carefully
IMPORTANT - READ BEFORE COPYING, INSTALLING OR USING.
Do not copy, install, or use the "Materials" provided under this license agreement ("Agreement"), until you have carefully read the following terms and conditions.
By copying, installing, or otherwi
bound by the terms of this Agree
of this Agreement, do not copy,
Pre-Release License Agreement
Intel Cilk++ SDK (1.1.0.8504) Setup
Destination Folder
Click Next to install to the default folder or click Change to choose another
Install Intel Cilk++ SDK (1.1.0.8504) to:
Intel Cilk++ SDK (1.1.0.8504) Setup
Completed the Intel Cilk++ SDK (1.1.0.8504) Setup Wizard
Click the Finish button to exit the Setup Wizard.
Back
Finish
Cancel
Intel Cilk++ SDK (1.1.0.8504) Setup
Installing Intel Cilk++ SDK (1.1
Please wait while the Setup Wizard i
Status:
Back
Next
Cancel
504)
o 2008.
Visual C++ 2008 Express Edition is not installed.
k Install to begin the installation. Click Back to review or change any of your
allation settings. Click Cancel to exit the wizard.
Back
Install
Cancel
קיימת גירסה חדשה יותר

http://cilk.mit.edu/

CILK HUB | DOWNLOAD | PROGRAMMING | COMPILING | COMPONENTS | PUBLICATIONS | BLOG

Cilk Hub

Welcome to Cilk Hub! Here you can find recent developments with the Cilk multithreaded programming technology.

What is Cilk?

Cilk aims to make parallel programming a simple extension of ordinary serial programming. Other concurrency platforms, such as Intel's Threading Building Blocks (TBB) and OpenMP, share similar goals of making parallel programming easier. But Cilk sets itself apart from other concurrency platforms through its simple design and implementation and its powerful suite of provably effective tools. These properties make Cilk well suited as a platform for next-generation multicore research.

The Cilk concurrency platform provides the following features:

Simple language extension

Cilk provides a simple linguistic extension to the C and C++ programming languages that allows programmers to parallelize their ordinary serial programs easily.

Compilation with Tapir/LLVM

The Tapir/LLVM compiler, which is based on Clang and LLVM, compiles Cilk programs more efficiently than existing compilers for parallel programming languages

New Blog Articles

Tapir · **Tapir/LLVM version 1.0-3**

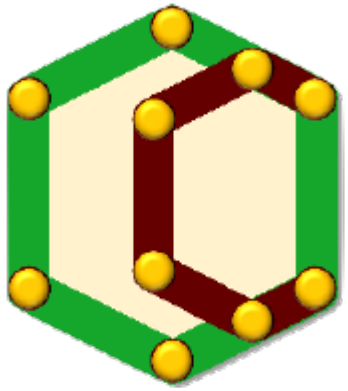
Cilk Hub · **Cilk Hub taking on Cilk development after Intel announcement**

Tapir · **Tapir wins best paper at PPoPP!**

More ›

Table of contents

OpenCilk



Home page: https://cilk.mit.edu/
Download from here:
https://github.com/OpenCilk/opencilk-project/releases/tag/opencilk/v1.0

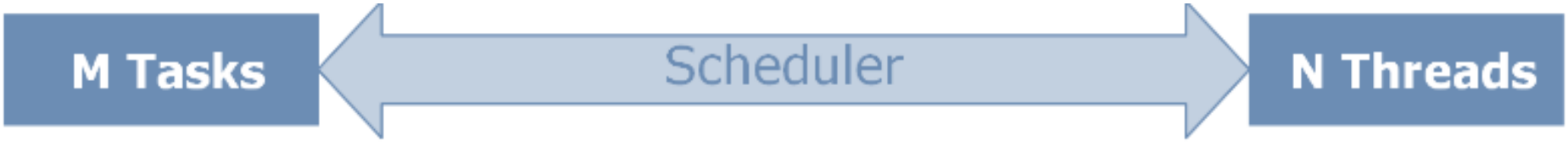
Installation directory on my laptop:
#/home/telzur/science/opt/OpenCilk-10.0.1-Linux

#For demos type:

```
$ export PATH=/home/telzur/science/opt/OpenCilk-10.0.1-Linux/bin:$PATH

$ clang -o fib ./fib.c -fopencilk
$ CILK_NWORKERS=4 time ./fib
```

An example with profiling:

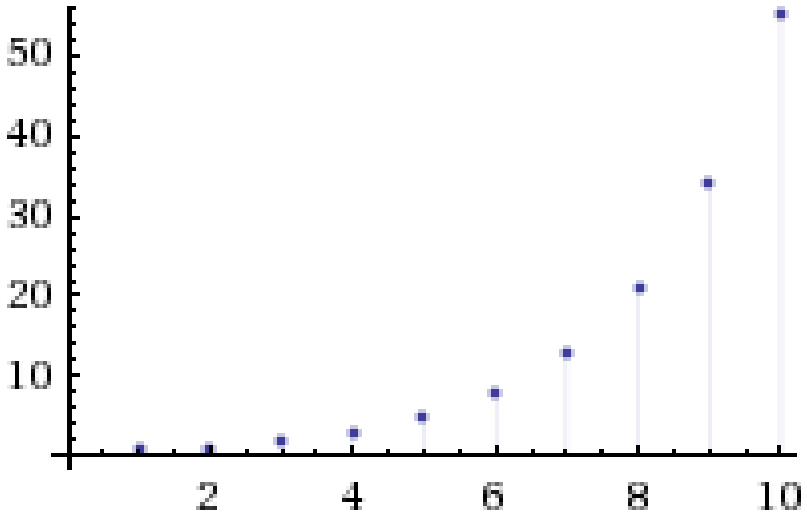
```
$ clang -o vector_add ./vector_add.cpp -fopencilk -fcilktool=cilkscale \
-L/usr/lib/gcc/x86_64-linux-gnu/9/ -lm
```



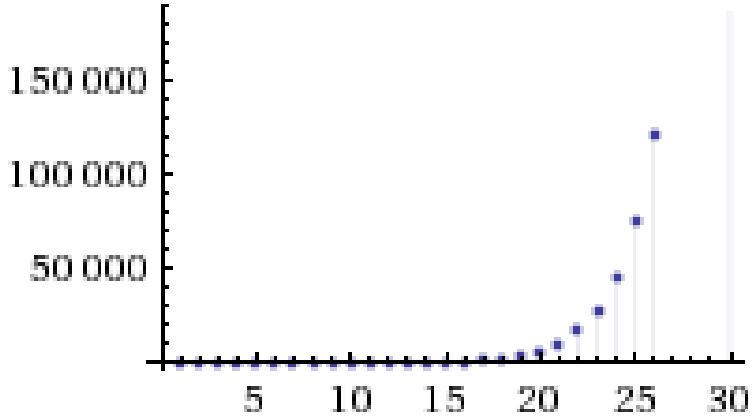
Source: http://wwwsfb.tpi.uni-jena.de/Events/PHSP11/slides/intel_cilk_tutorial.pdf

Fibonachi (Fibonacci)

Try:
http://www.wolframalpha.com/input/?i=fibonacci+number



(*n* from 2 to 10)



(*n* from 5 to 30)

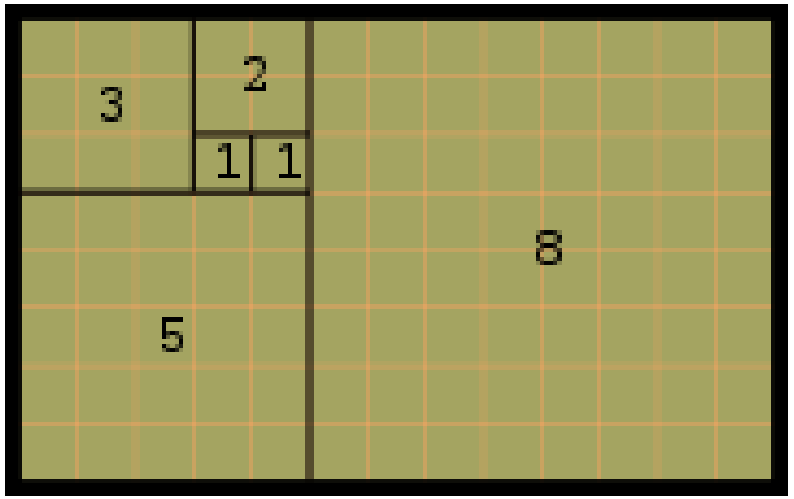
n	F_n
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55

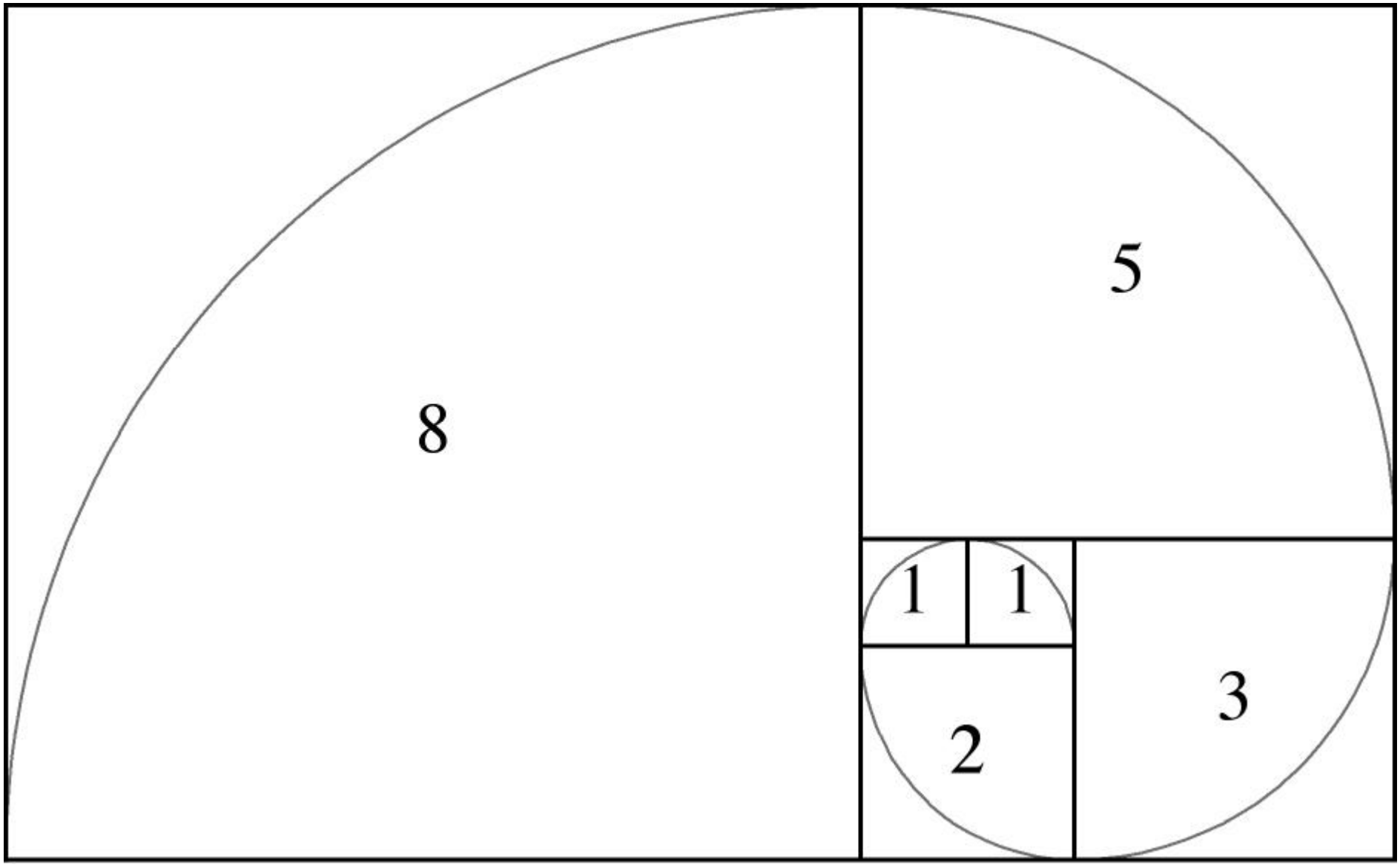
Fibonachi Numbers
serial version

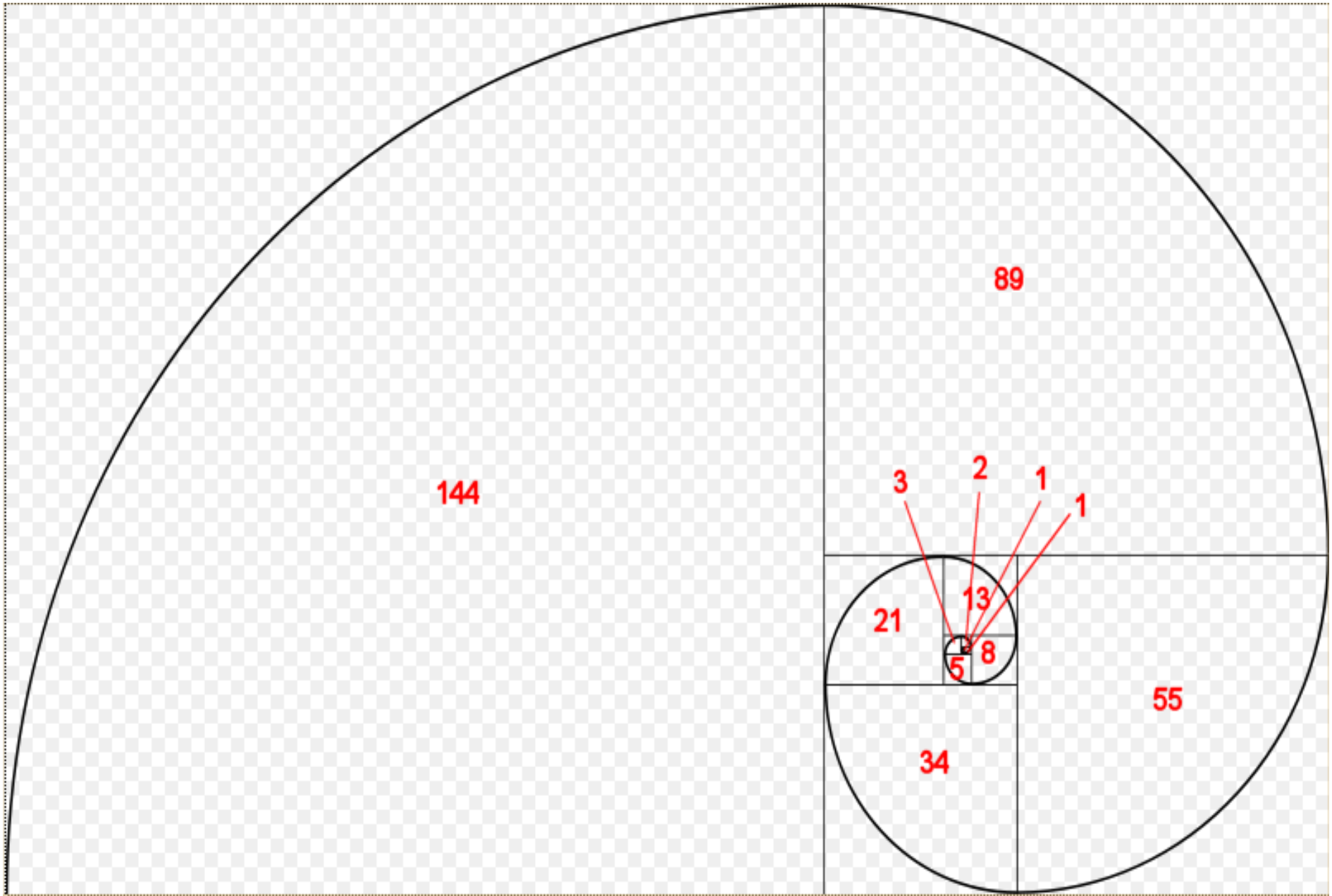
```
// 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
// Serial version
// Credit: http://myxman.org/dp/node/182

long fib_serial(long n) {

  if (n < 2) return n;

  return fib_serial(n-1) + fib_serial(n-2);

}
```

Cilk++ Fibonachi (Fibonacci)

```
#include <cilk.h>
#include <stdio.h>


long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    x = cilk_spawn fib_parallel(n-1);
    y = fib_parallel(n-2);
    cilk_sync;
    return (x+y);
}

int cilk_main()
{
    int N=50;
    long result;
    result = fib_parallel(N);
    printf("fib of %d is %d\n",N,result);
    return 0;
}
```




8
5
1
1
2
3


144
89
3
2
1
1
21
13
8
5
55
34

Cilk++

Simple, powerful expression of task parallelism:
***cilk_for* – Parallelize for loops**
***cilk_spawn* – Specify the start of parallel execution**
***cilk_sync* – Specify the end of parallel execution**

http://software.intel.com/en-us/articles/intel-cilk-plus/

Cilk_spawn

ADD PARALLELISM USING CILK_SPAWN

The cilk_spawn keyword indicates that a function (the *child*) may be executed in parallel with the code that follows the cilk_spawn statement (the *parent*). Note that the keyword *allows* but does not *require* parallel operation. The Cilk++ scheduler will dynamically determine what actually gets executed in parallel when multiple processors are available. The cilk_sync statement indicates that the function may not continue until all cilk_spawn requests in the same function have completed. cilk_sync does not affect parallel strands spawned in other functions.

Fibonacci Example: Creating Parallelism

```
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 elision

=הַשָּׁטָה

Cilk code

```
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);
  }
}
```

Cilk is a ***faithful*** extension of C. A Cilk program's ***serial elision*** is always a legal implementation of Cilk semantics. Cilk provides *no* new data types.

Basic Cilk Keywords

```
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);
  }
}
```

Identifies a function as a ***Cilk procedure***, capable of being spawned in parallel.

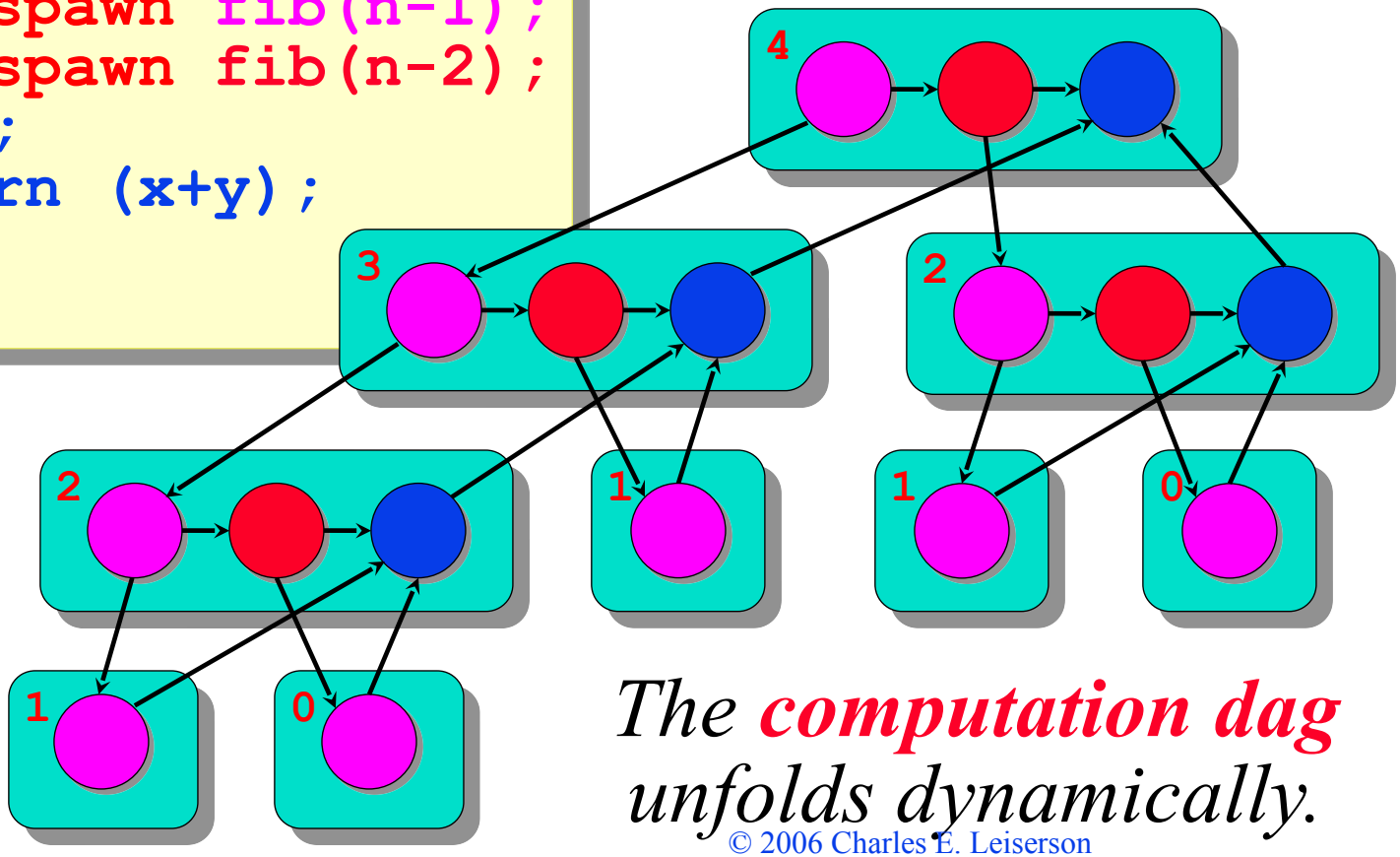
The named ***child*** Cilk procedure can execute in parallel with the ***parent*** caller.

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

Dynamic Multithreading

```
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);
  }
}
```

Example: `fib(4)`



processors are virtualized

The computation dag unfolds dynamically.

On the Hobbits

```
Demo: /users/agnon/misc/tel-zur/cilk
-bash-4.1$ gcc -o fib ./fib.c                 // serial version
-bash-4.1$ cilk++ -o fib_cilk ./fib_cilk.c //parallel version
-bash-4.1$  ./fib
Fibonacci of 45 is 1134903170
-bash-4.1$ export CILK_NWORKERS=4
 Bash-4.1$ #on other shells: setenv CILK_NWORKERS 4
-bash-4.1$  ./fib_cilk
Fibonacci of 45 is 1134903170
```

On the Hobbits (cont')

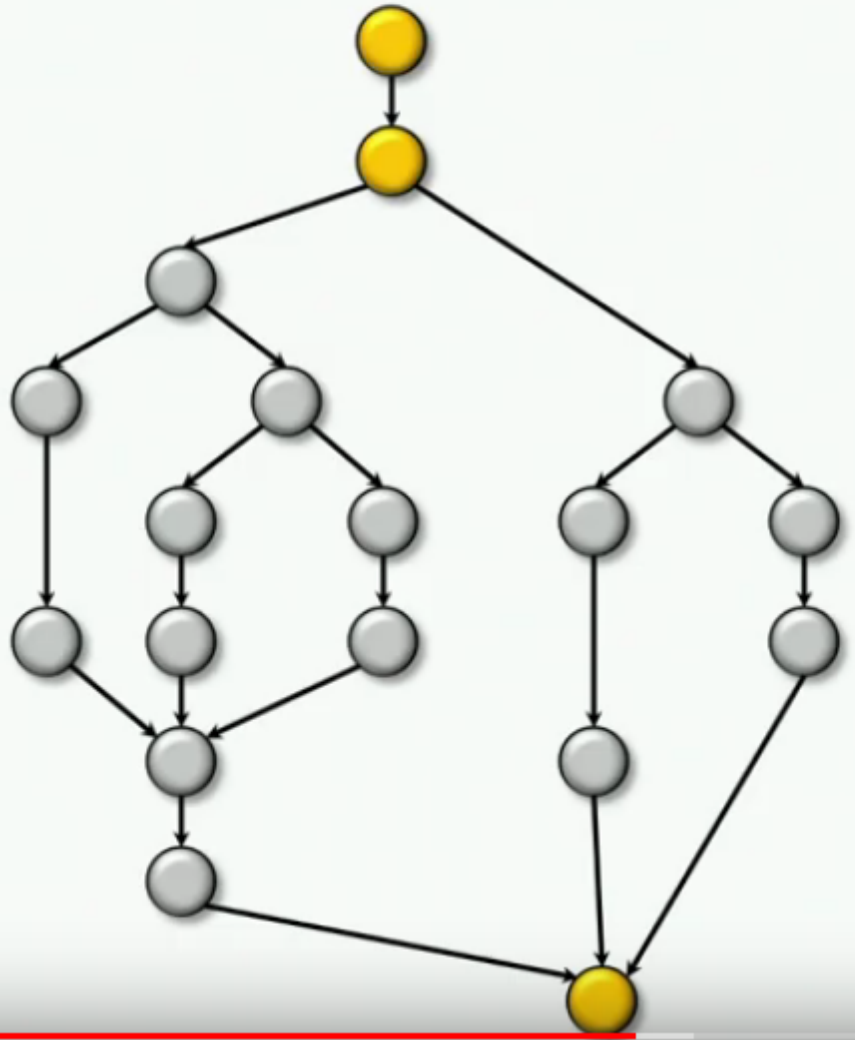
```
$ cilkscreen -a -r report.txt ./fib_cilk

$ more ./report.txt
Cilkscreen Race Detector V1.1.0, Build 8503
No errors found by Cilkscreen
```

On the Hobbits (cont')

```
cilkview -verbose -plot gnuplot ./fib_cilk
cilkview: CILK_NPROC=16 /usr/local/cilk/bin/../lib64/pinbin -ifeellucky -t
/usr/local/cilk/bin/../lib64/workspan.so -- ./fib_cilk
fibonacci of 15 is 610
Whole Program Statistics:

Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
   Work :                    3,938,736 instructions
   Span :                    3,690,448 instructions
   Burdened span :               3,793,650 instructions
   Parallelism :                 1.07
   Burdened parallelism :            1.04
   Number of spawns/syncs:           986
   Average instructions / strand :       1,331
   Strands along span :              29
   Average instructions / strand on span :   127,256
   Total number of atomic instructions :     10
   Frame count :                 2962
2) Speedup Estimate
   2 processors:     0.76 - 1.07
   4 processors:     0.68 - 1.07
   8 processors:     0.64 - 1.07
   16 processors:    0.63 - 1.07
   32 processors:    0.62 - 1.07
```

Quantifying Parallelism

What is the parallelism of this computation?

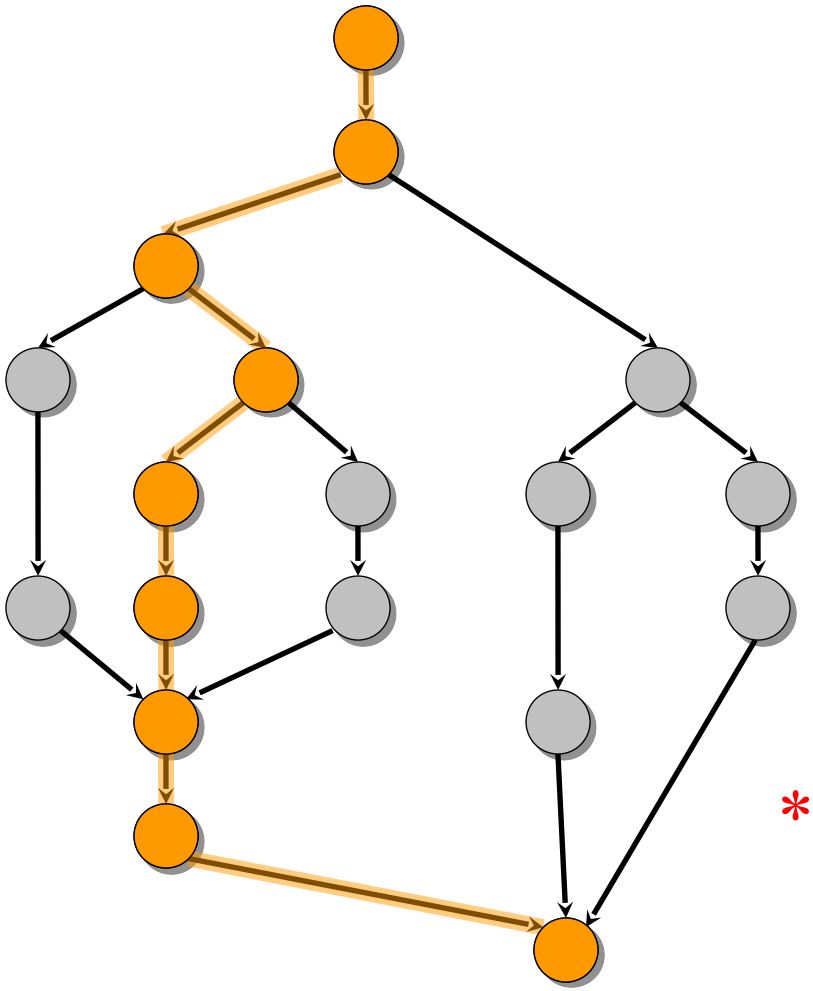


Amdahl's Law says that since the serial fraction is 3/18 = 1/6, the speedup is upper-bounded by 6.

מסתבר שהחסם על ההאצה על-פי חוק אמדהל הוא די גס.
הנבוי כאן הוא אינו מספיק טוב כפי שתכף נראה

Algorithmic Complexity Measures

T_P = execution time on P processors



T_1 = ***work***

T_∞ = ***span****

LOWER BOUNDS

- $T_P \geq T_1/P$
- $T_P \geq T_\infty$

*Also called ***critical-path length*** or ***computational depth***.

For p processors:

$1*T_1 = work_1$

$p*T_p=Work_p$

$p*T_p \geq work_1=T_1$

היזכרו בהגדרת העבודה (שיעור מס' 1)

חוק העבודה: $T_p \geq T_1/p$

חוק הספאן

$T_p \geq T_\infty$

Speedup

Definition: T_1/T_P = ***speedup*** on P processors.

If $T_1/T_P = O(P) \le P$, we have ***linear speedup***;

$= P$, we have ***perfect linear speedup***;

$> P$, we have ***superlinear speedup***, which is not possible in our model, because of the lower bound $T_P \ge T_1/P$.

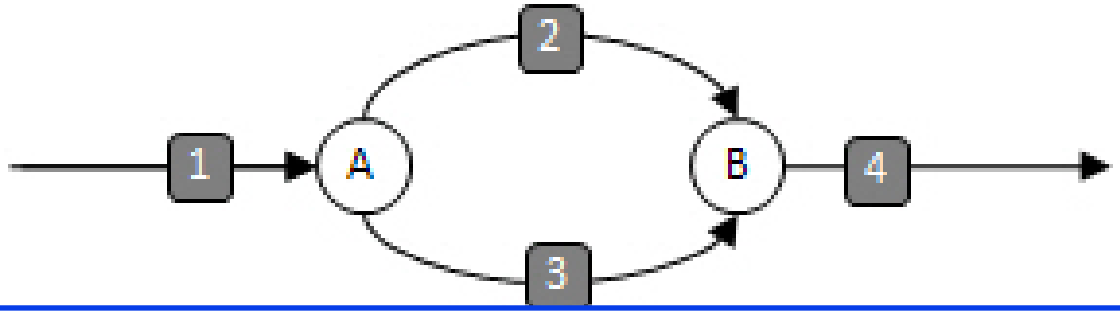
T_1/T_∞ = ***parallelism***

= *the average amount of work per step along the span.*

כמה עבודה עודפת יש לנו מעבר לנתיב הקריטי?

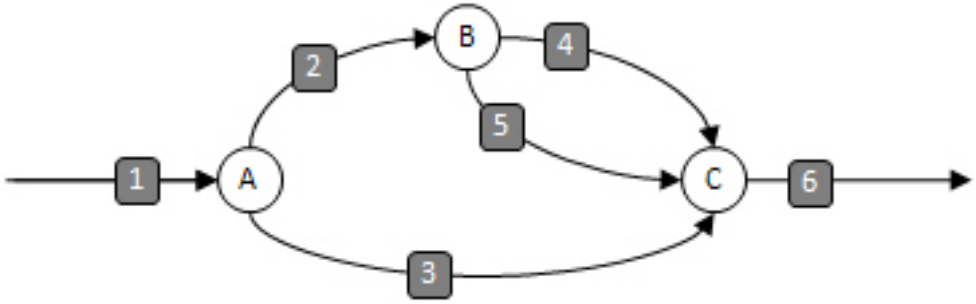
Strands and Knots

A Cilk++ program fragments

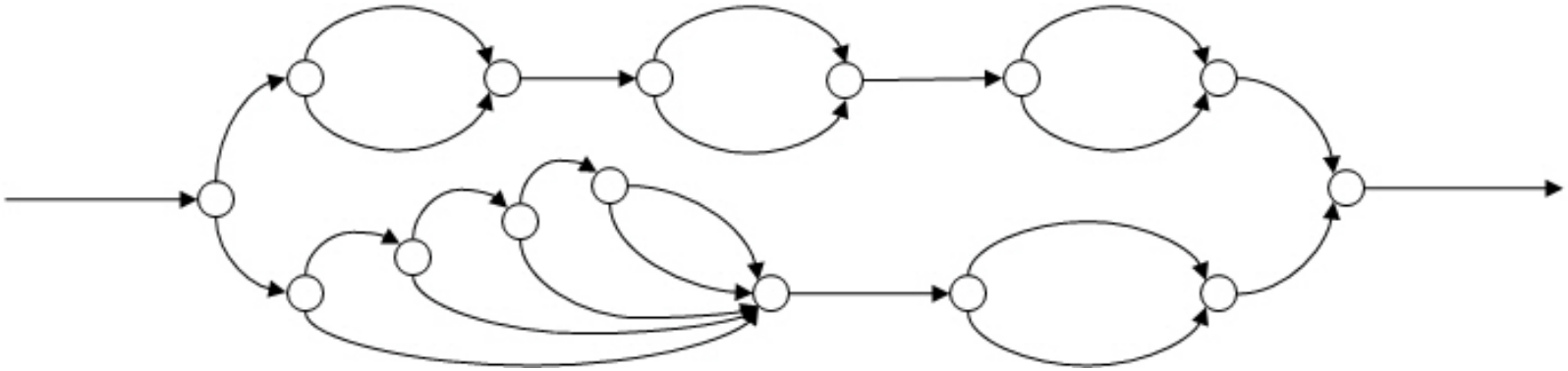



```
...
do_stuff_1(); // execute strand 1
cilk_spawn func_3(); // spawn strand 3 at knot A
do_stuff_2(); // execute strand 2
cilk_sync; // sync at knot B
do_stuff_4(); // execute strand 4 ...
```

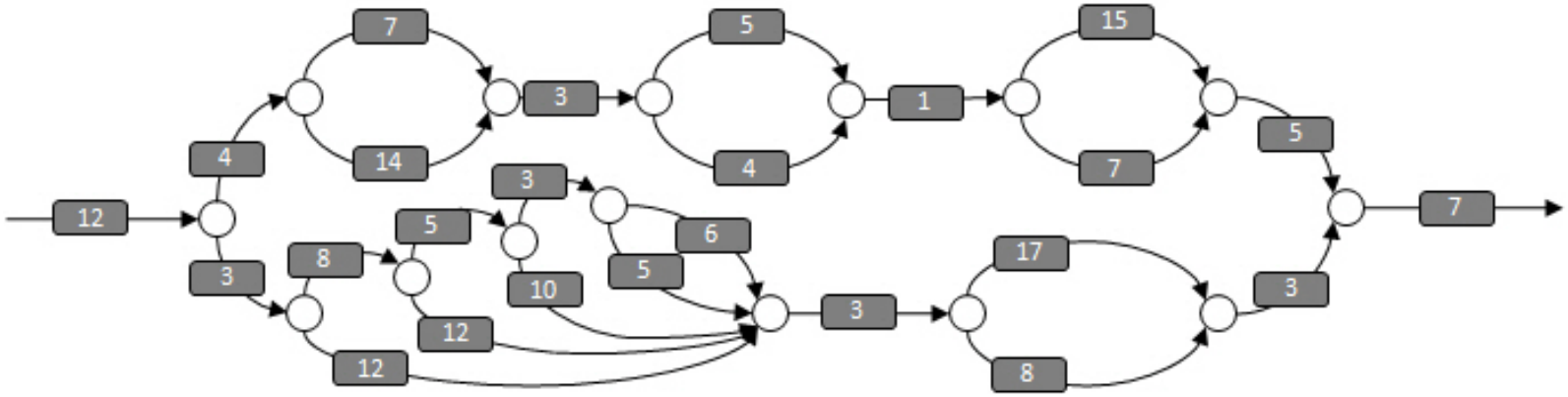
DAG with two spawns (labeled A and B) and one sync (labeled C)



a more complex Cilk++ program (DAG):



Let's add labels to the strands to indicate the number of milliseconds it takes to execute each strand



In ideal circumstances (e.g., if there is no scheduling overhead) then, if an unlimited number of processors are available, this program should run for 68 milliseconds.

Work and Span

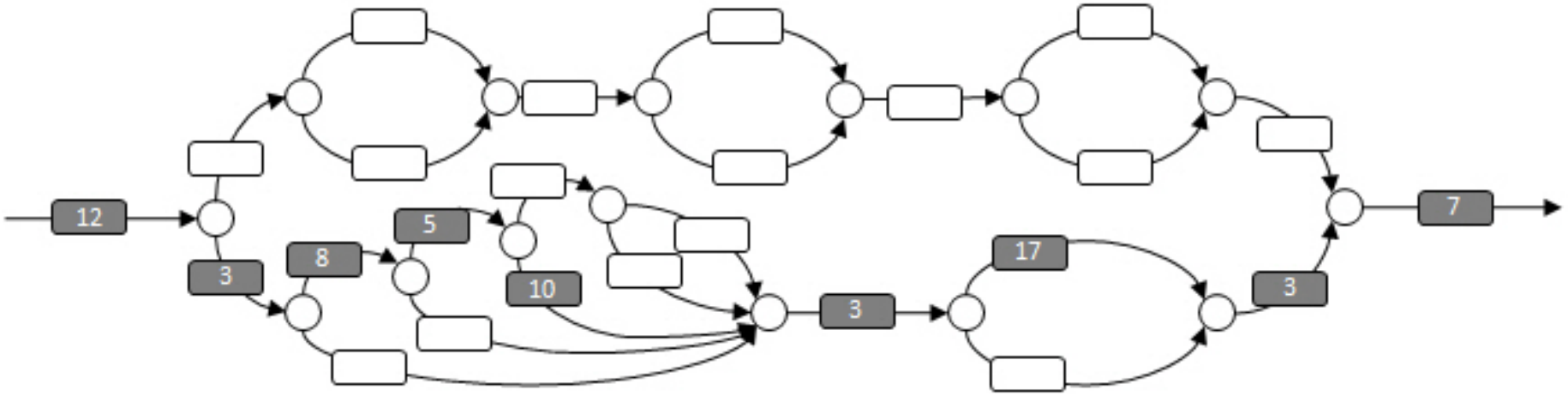
Work

The total amount of processor time required to complete the program is the sum of all the numbers. We call this the *work*.

In this DAG, the work is 181 milliseconds for the 25 strands shown, and if the program is run on a single processor, the program should run for 181 milliseconds.

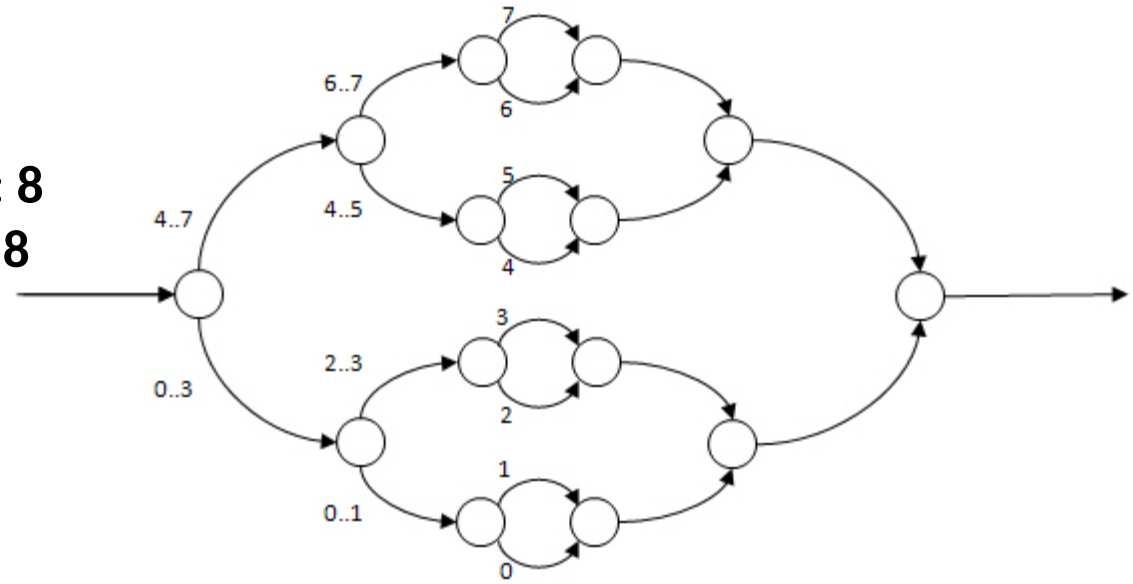
Span

Another useful concept is the *span*, sometimes called the *critical path length*. The span is the *most expensive* path that goes from the beginning to the end of the program. In this DAG, the span is 68

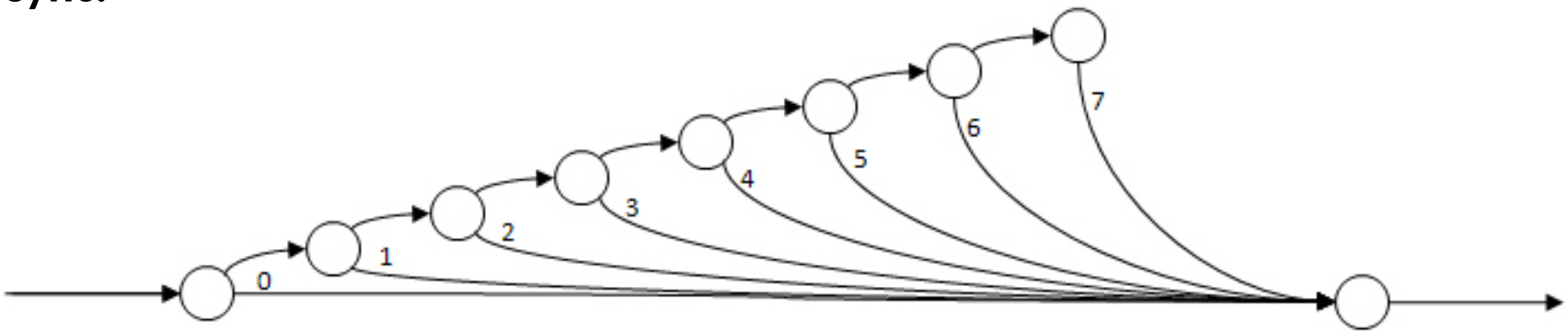


divide-and-conquer strategy

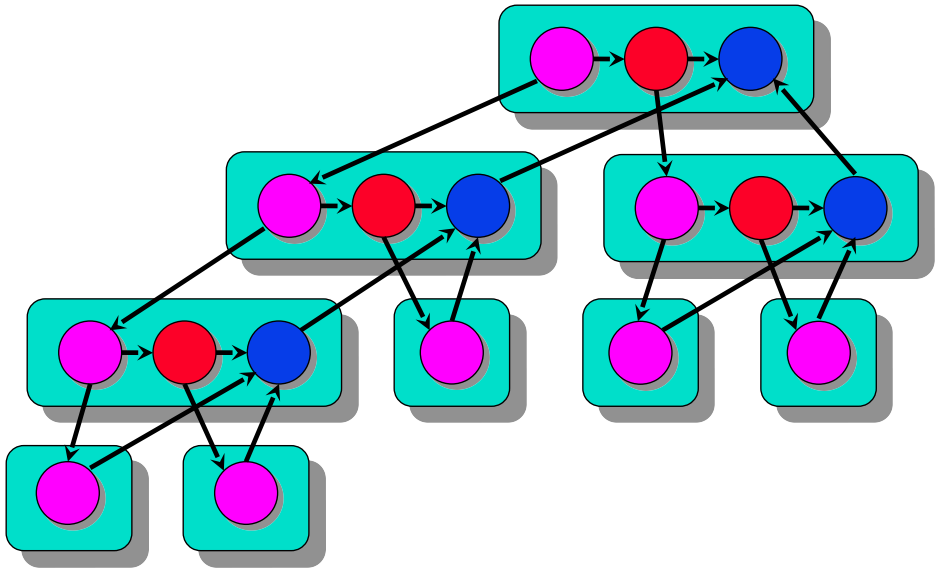
cilk_for
Shown here: 8 threads and 8 iterations



Here is the DAG for a serial loop that spawns each iteration. In this case, the work is not well balanced, because each child does the work of only one iteration before incurring the scheduling overhead inherent in entering a sync.



Example: `fib(4)`



Assume for simplicity that each Cilk thread in `fib()` *takes unit time to execute.*

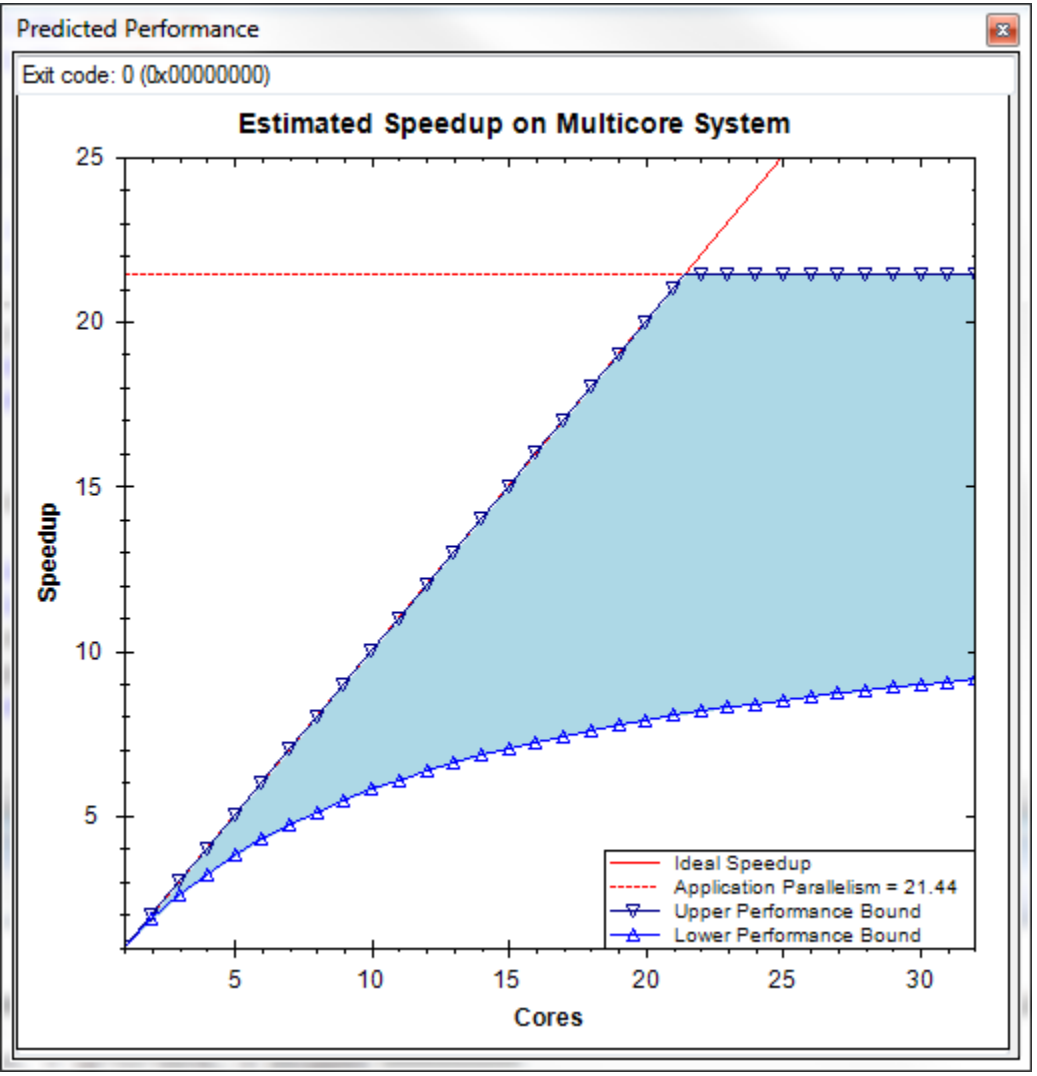
Work: $T_1 = 17$

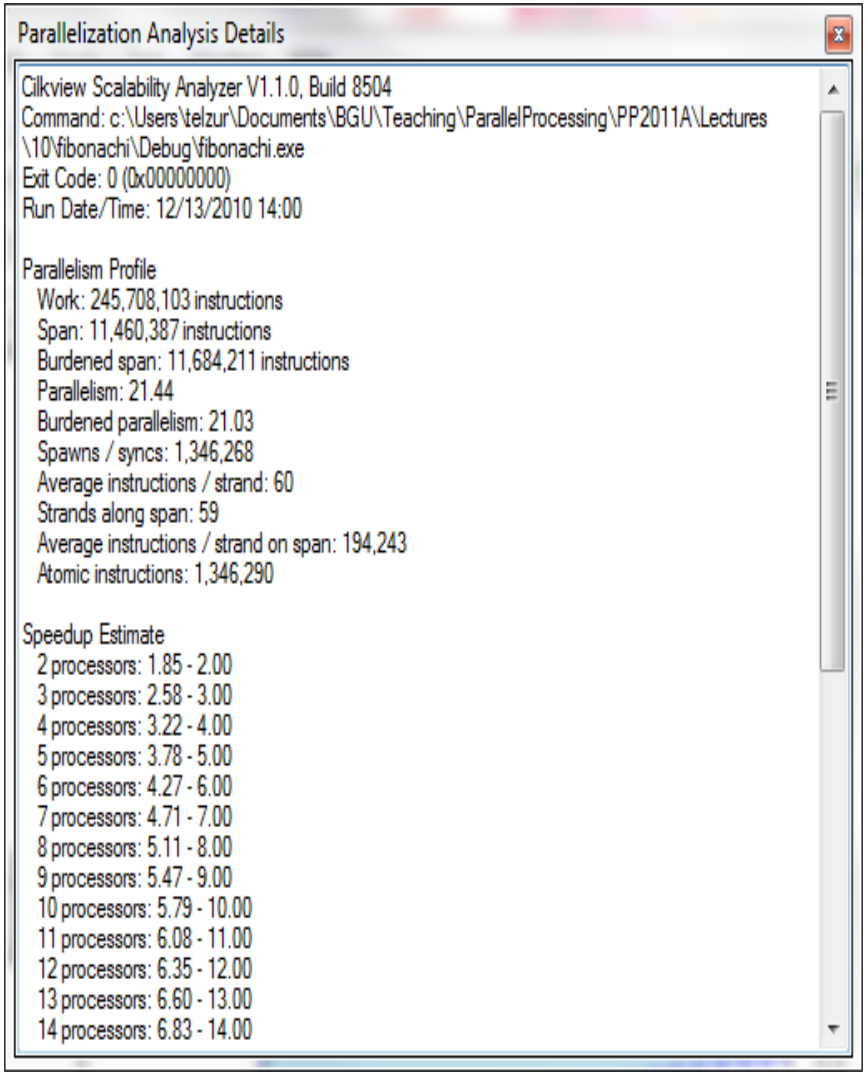
Span: $T_\infty = 8$

Parallelism: $T_1/T_\infty = 2.125$

Using many more than 2 processors makes little sense.

Cilkview Fn(30)





~~Show demo in VisualStudio 2008~~
~~C:\Users\telzur\Documents\BGU\Teaching\ParallelProcessing\PP2011A\Lectures\10\fibonachi\fibonachi\fibonachi.vcproj~~

התיישן...

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

C

```
void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    vadd (A, B, n/2);
    vadd (A+n/2, B+n/2, n-n/2);

  }
}
```

Parallelization strategy:

1. Convert loops to recursion.

Demo (28/12/20): vector_add demo under Intel environment (see readme_guy.txt)

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

Cilk

```
cilk void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    spawn vadd (A, B, n/2);
    spawn vadd (A+n/2, B+n/2, n-n/2);
    sync;
  }
}
```

Parallelization strategy:

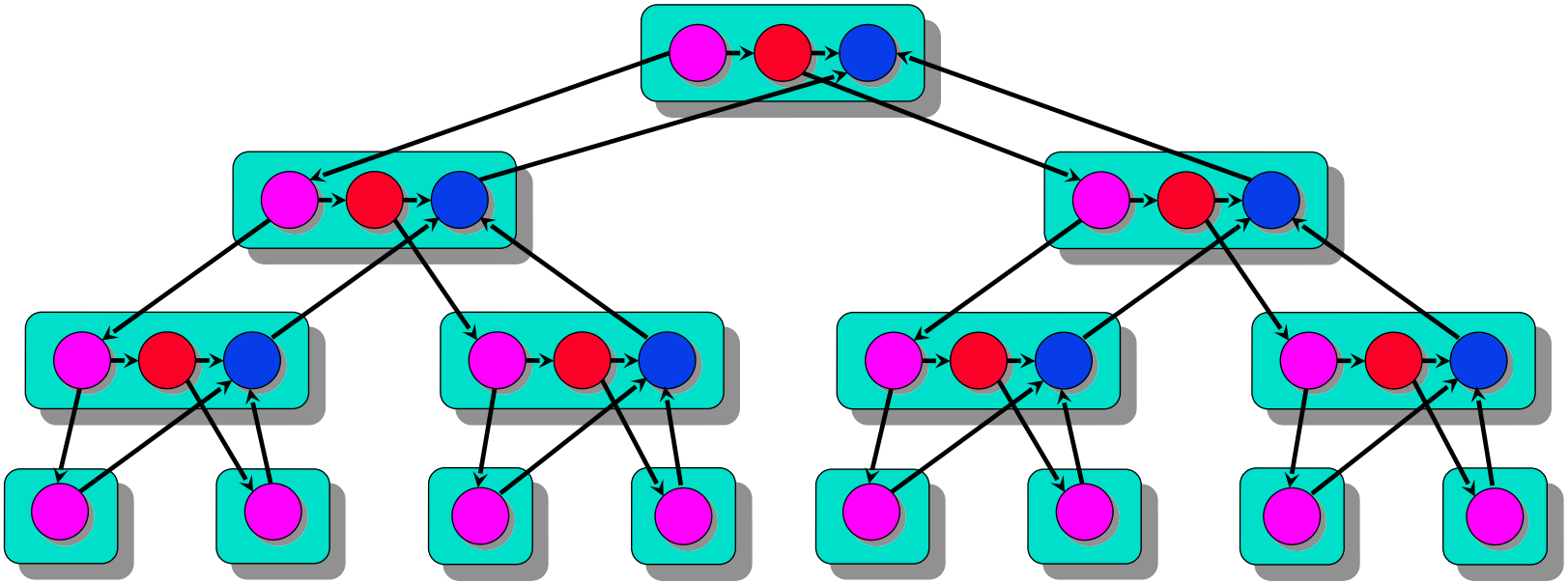
1. Convert loops to recursion.
2. Insert Cilk keywords.

Side benefit:
D&C is generally good for caches!

D&C = Divide and Conquer

Vector Addition

```
cilk void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    spawn vadd (A, B, n/2);
    spawn vadd (A+n/2, B+n/2, n-n/2);
    sync;
  }
}
```



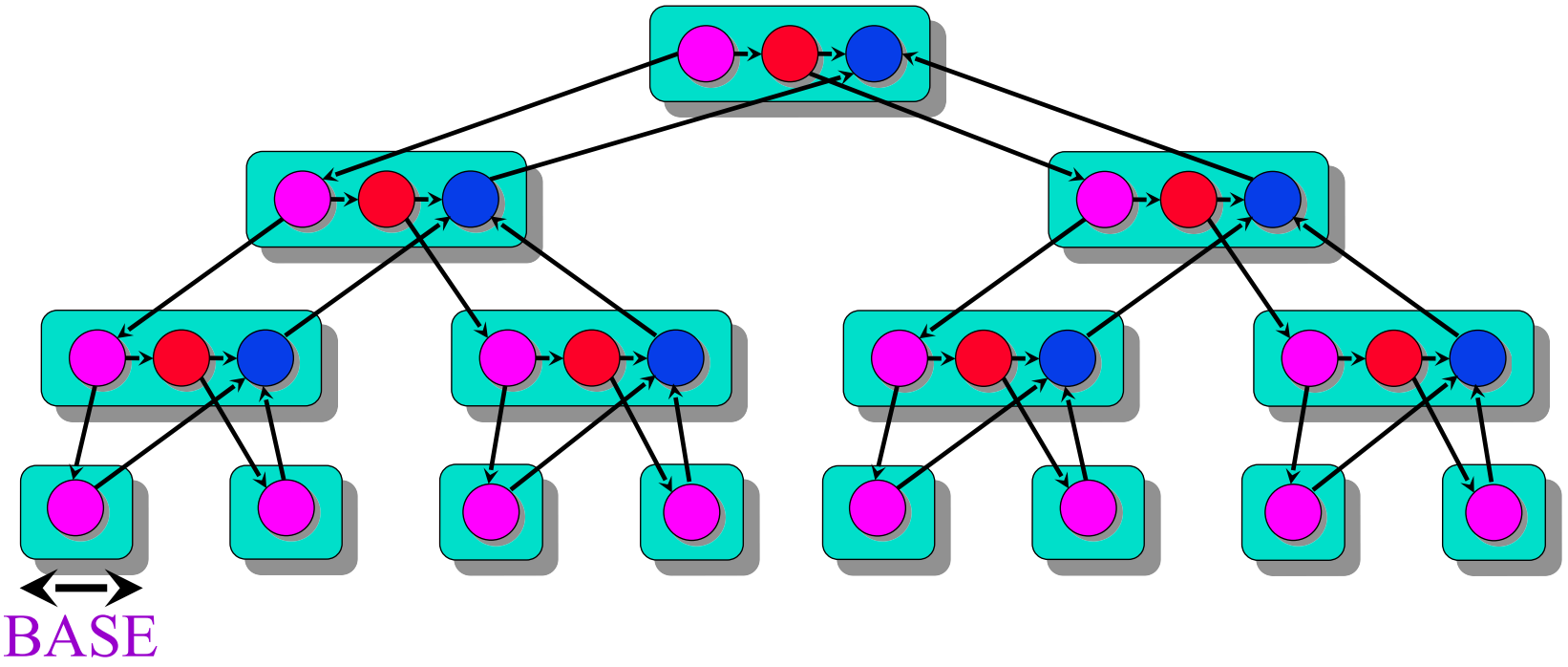
Vector Addition Analysis

To add two vectors of length n, where BASE = O(1):

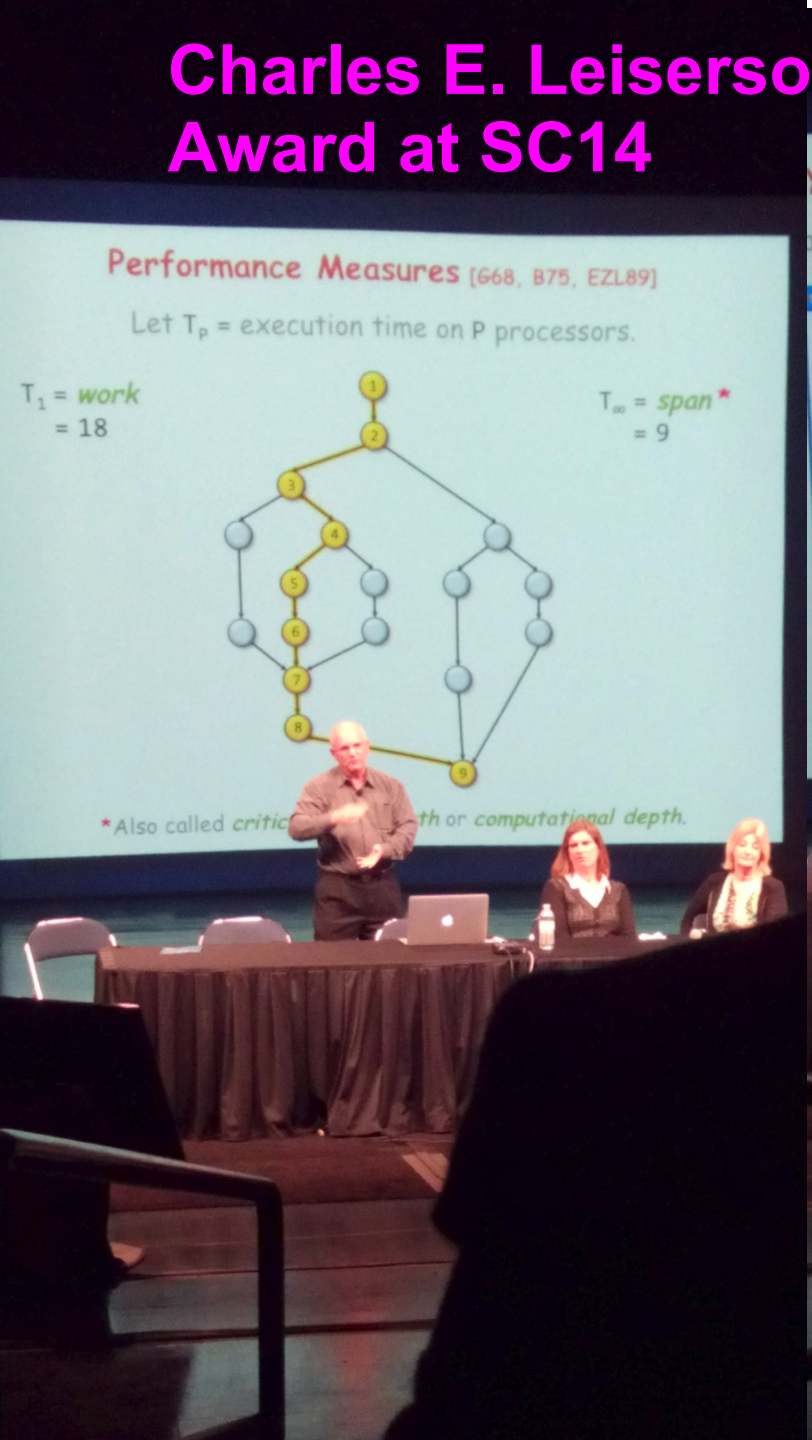
Work: $T_1 = \mathrm{O}(n)$

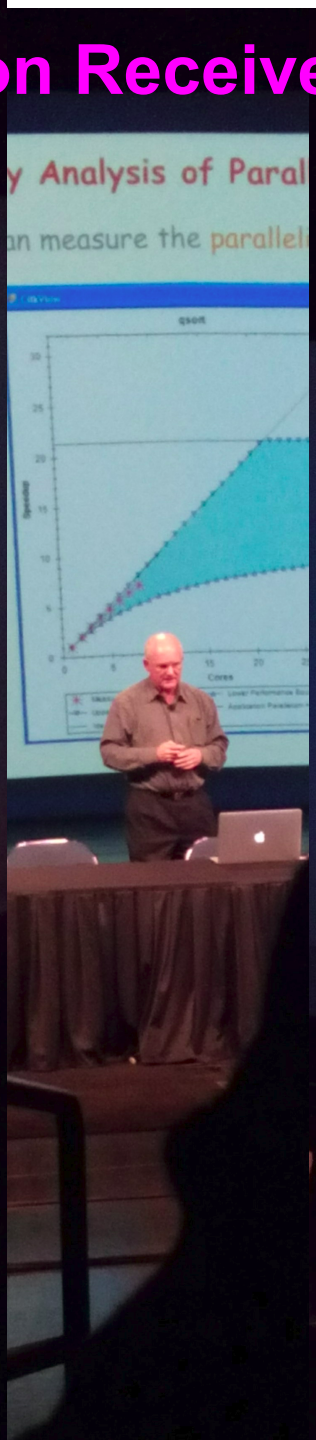
Span: $T_\infty = \mathrm{O}(\lg n)$

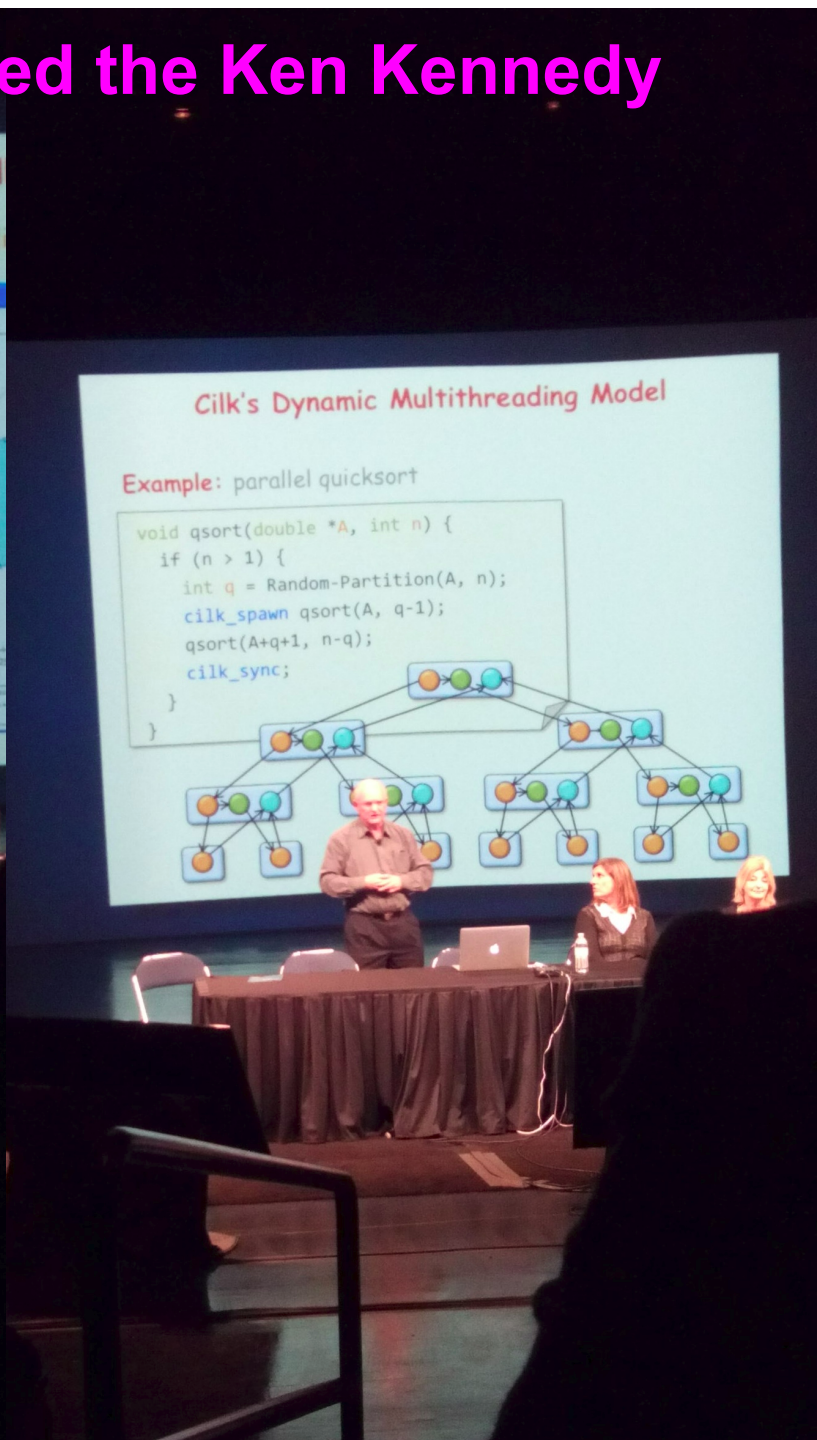
Parallelism: $T_1/T_\infty = \mathrm{O}(n/\lg n)$



Charles E. Leiserson Received the Ken Kennedy Award at SC14







On my **LIFEBOOK** laptop
Folder:
/home/telzur/Documents/Teaching/BGU/PP/PP2016B/lectures/10/code/cilk

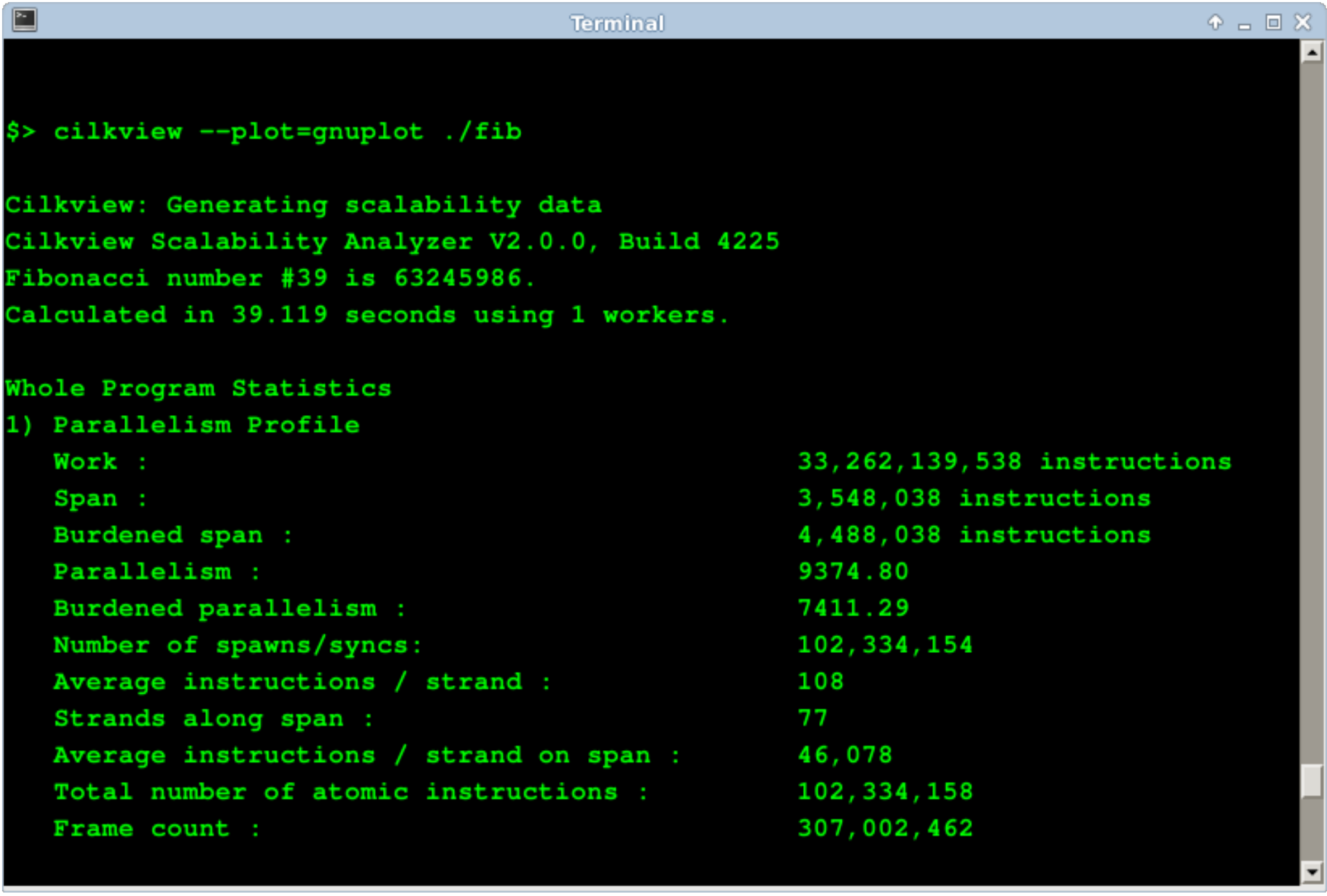
Compilation with GNU:

```
$CILK/bin/g++ -o fib -fcilkplus -L
$CILK/lib64 -lcilkrts -lstdc++ -I
$CILK/include/cilk ./fib.cpp
```

Compilation with Intel

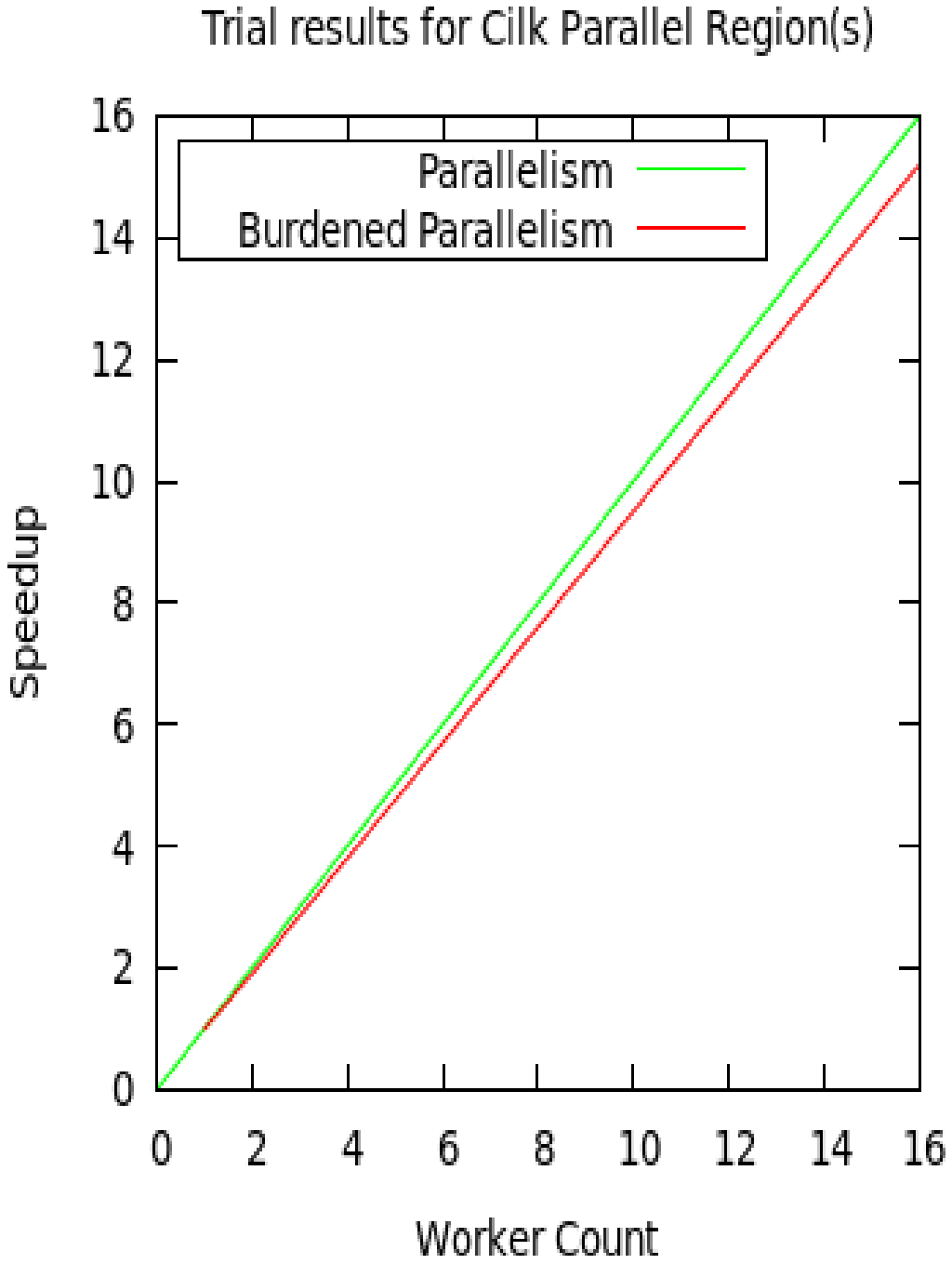
```
/opt/intel/bin/icc  -o
fib_intel ./fib_intel.cpp
```

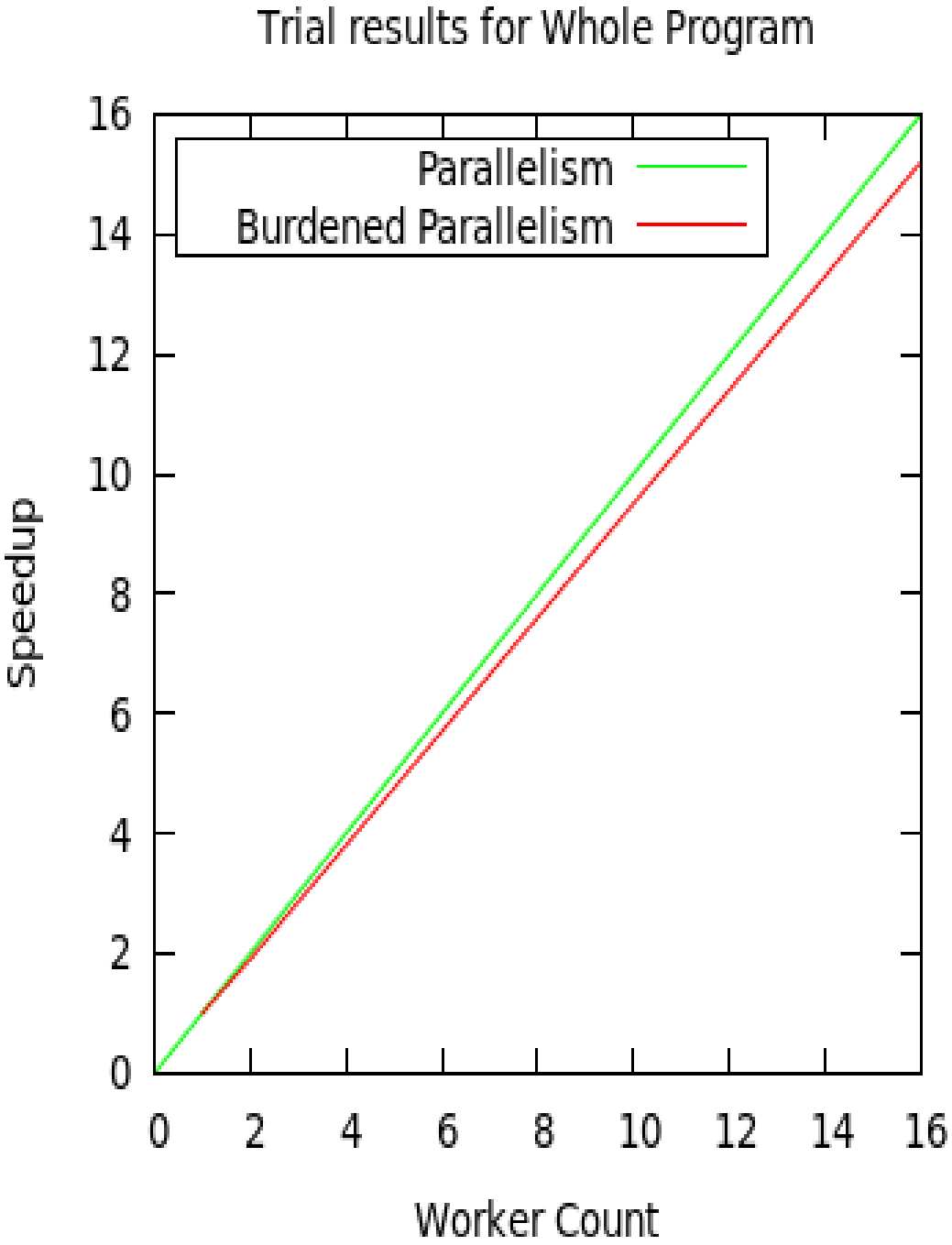
```
cilkview --plot=gnuplot ./fib
```


Terminal
$> cilkview --plot=gnuplot ./fib
Cilkview: Generating scalability data
Cilkview Scalability Analyzer V2.0.0, Build 4225
Fibonacci number #39 is 63245986.
Calculated in 39.119 seconds using 1 workers.
Whole Program Statistics
1) Parallelism Profile
Work : 33,262,139,538 instructions
Span : 3,548,038 instructions
Burdened span : 4,488,038 instructions
Parallelism : 9374.80
Burdened parallelism : 7411.29
Number of spawns/syncs: 102,334,154
Average instructions / strand : 108
Strands along span : 77
Average instructions / strand on span : 46,078
Total number of atomic instructions : 102,334,158
Frame count : 307,002,462

Terminal

```
2) Speedup Estimate
     2 processors:        1.90 - 2.00
     4 processors:        3.80 - 4.00
     8 processors:        7.60 - 8.00
    16 processors:       15.20 - 16.00
    32 processors:       30.40 - 32.00
    64 processors:       60.80 - 64.00
   128 processors:      121.60 - 128.00
   256 processors:      241.85 - 256.00

Cilk Parallel Region(s) Statistics - Elapsed time: 38.964 seconds
1) Parallelism Profile
   Work :                                    33,258,600,915 instructions
   Span :                                    9,415 instructions
   Burdened span :                           949,415 instructions
   Parallelism :                             3532512.05
   Burdened parallelism :                    35030.63
   Number of spawns/syncs:                   102,334,154
   Average instructions / strand :           108
   Strands along span :                      38
   Average instructions / strand on span :   247
   Total number of atomic instructions :     102,334,158
```


Trial results for Cilk Parallel Region(s)
Parallelism
Burdened Parallelism
Speedup
Worker Count
0
2
4
6
8
10
12
14
16


Trial results for Whole Program
Parallelism
Burdened Parallelism
Speedup
Worker Count
0
2
4
6
8
10
12
14
16

Examples on my laptop (currently ASUS ROG, Linux mint20)

Directory: ~/lectures/10/cilk-fib

Examples:

```
g++-5 -o vec_add -fcilkplus ./vector_add.cpp
```

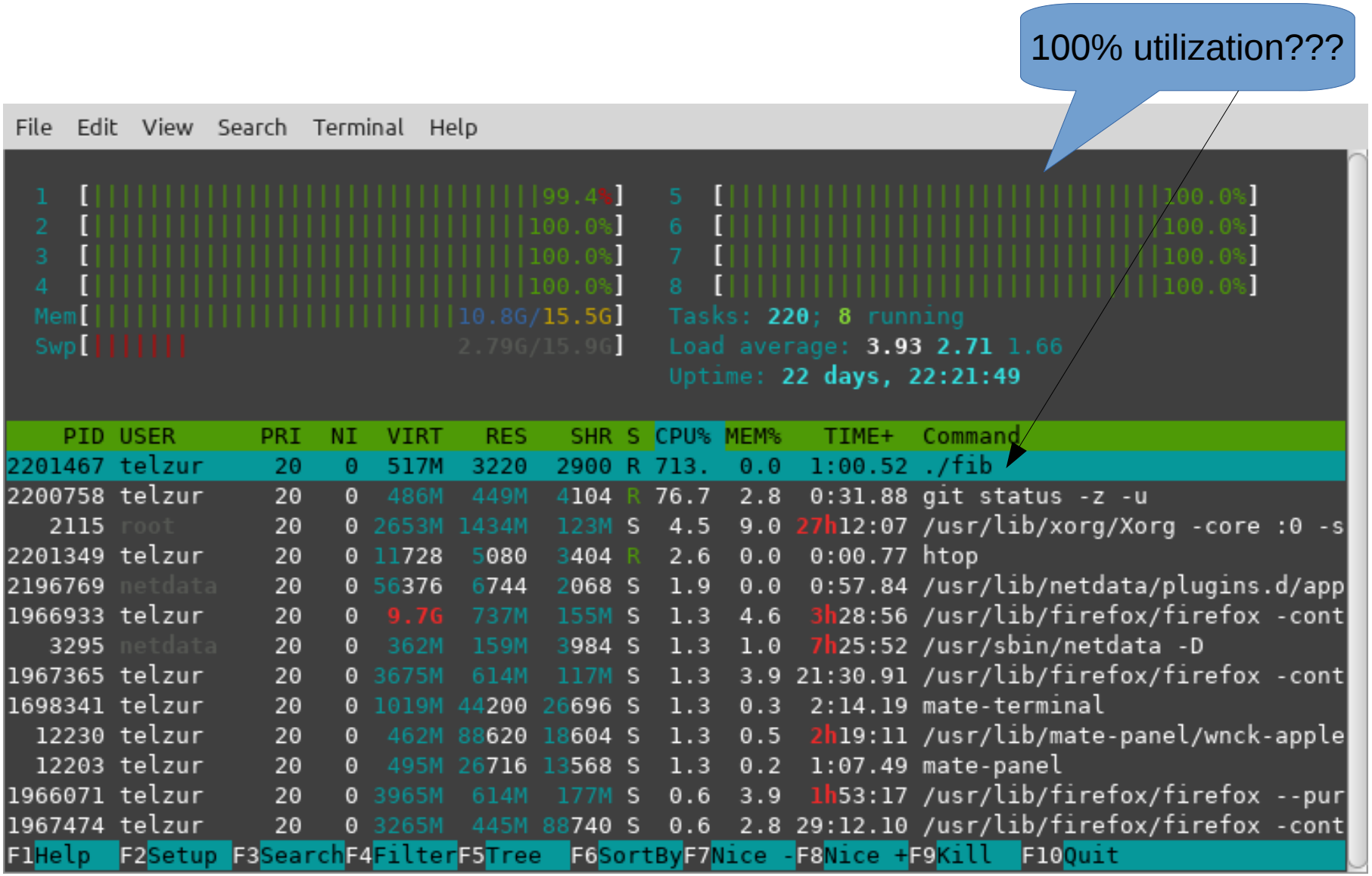
and:

```
g++-5 -o fib -fcilkplus ./fib.cpp
```

(see next slide)

```
# OpenCilk with Clang:
$ ~/science/opt/OpenCilk-10.0.1-Linux/bin/clang
-fopencilk ./fib_clang.c -o fib_clang

export CILK_NWORKERS=4
./fib
```

Performance issues with Cilk Fibonacci

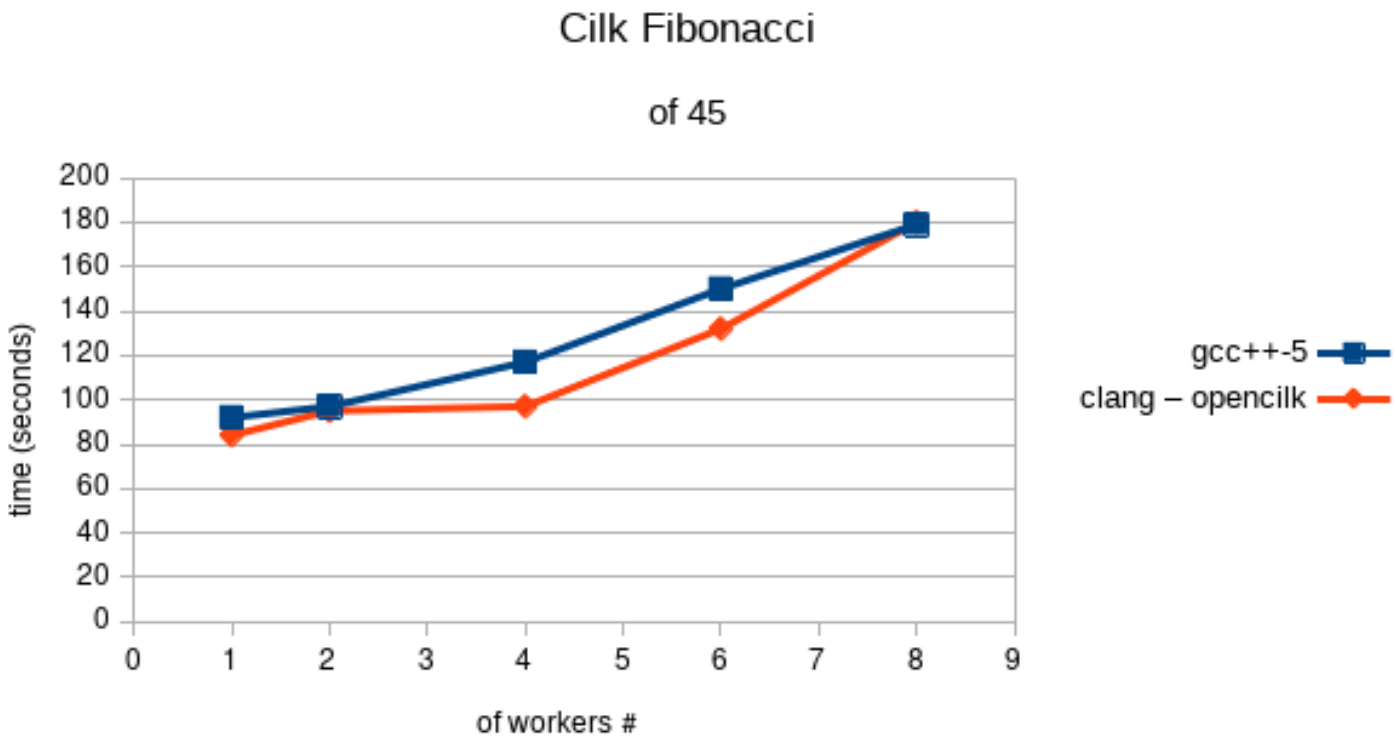


Cilk Fibonacci results

# workers	g++-5 time
1	92
2	97
4	117
6	150
8	179

מסקנה:
דוגמה יפה מבחינה פדגוגית
אבל ביצועים גרועים – ראו גם השקף הבא!

ההסבר:
הרבה זמן מתבזבז על יצירת המשימות אך כל משימה בפני עצמה דלה מאוד בחישוב...



https://cilk.mit.edu/programming/

Spawn and sync

Let us first examine the task-parallel keywords `cilk_spawn` and `cilk_sync`. Consider the following example code for a `fib` routine, which uses these keywords to parallelize the computation of the nth Fibonacci number.

class="pygments highlight" style="background: #f8f8f8;"> CILK

```
1  int64_t fib(int64_t n) {
2    if (n < 2) return n;
3    int x, y;
4    x = cilk_spawn fib(n - 1);
5    y = fib(n - 2);
6    cilk_sync;
7    return x + y;
8  }
```

Note to the algorithms police

The example `fib` routine is a terribly inefficient code for computing Fibonacci numbers. This `fib` routine computes the \(n\)th Fibonacci number using \(\Theta(\phi^n)\) work, where \(\phi\) denotes the golden ratio, while in fact this number can be computed using \(\Theta(\lg n)\) work. We use this example `fib` code simply for didactic purposes.

cilkscale

compile:

```
~/science/opt/OpenCilk-10.0.1-Linux/bin/clang -fopencilk
./fib_clang.c -o fib_clang -fcilktool=cilkscale -L/usr/lib/x86_64-
linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/9
```

```
~/….../cilk-fib $ ./fib_clang
Fibonacci number #45 is 1134903170.
Calculated in 2543.843 seconds.
tag,work (seconds),span (seconds),parallelism,burdened_span
(seconds),burdened_parallelism
,1679.12,0.0928454,18085.1,0.0930579,18043.8
```

so far I couldn't use cilkscale is not working with graphics

```
export
LD_LIBRARY_PATH=/home/telzur/science/opt/OpenCilk-10.0.1-Linux/lib/
clang/10.0.1/lib/x86_64-unknown-linux-gnu:$LD_LIBRARY_PATH
 ~/science/opt/OpenCilk-10.0.1-Linux/bin/clang  -fopencilk ./fib_clang.c -o
fib_clang -fcilktool=cilkscale -fcilktool=cilkscale-benchmark -L/usr/lib/x86_64-
linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/9  -lclang_rt.cilkscale
-L/home/telzur/science/opt/OpenCilk-10.0.1-Linux/lib/clang/10.0.1/lib/x86_64-
unknown-linux-gnu
```

למידע נוסף:

https://cilk.scripts.mit.edu/pact21/opencilk-pact-2021.pdf

https://www.youtube.com/watch?v=a_R_DpsENfk&t=34s
MIT 6.172 Performance Engineering of Software Systems, Fall 2018
Instructor: Julian Shun
View the complete course: https://ocw.mit.edu/6-172F18
YouTube Playlist: https://www.youtube.com/playlist?...list

Professor Shun discusses races and parallelism, how cilkscale can analyze computation and detect .determinancy races, and types of schedulers

Introduction to CILK

The slides are based on:

"Multithreaded Programming in Cilk" by **Charles E. Leiserson**

http://gamma.cs.unc.edu/SC2007/Leiserson-SC07-Workshop.pdf

and some of the slides are mine (Guy Tel-Zur)

Version ~~24/5/2015~~, ~~23/5/2016~~, ~~20/12/2020~~, ~~9/5/2021~~, 18/12/2021

Cilk

A C language for programming dynamic multithreaded applications on shared-memory multiprocessors.

Example applications:

- virus shell assembly
- graphics rendering
- *n*-body simulation
- heuristic search
- dense and sparse matrix computations
- friction-stir welding simulation
- artificial evolution

Guy: http://software.intel.com/en-us/intel-cilk-plus

Intel® Developer Zone:

Intel® Cilk™ Plus

OVERVIEW | OBTAIN | LEARN | SUPPORT

A Quick, Easy and Reliable way to Improve Performance

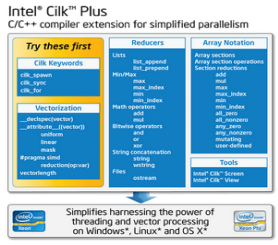
Intel® Cilk™ Plus is an extension to C and C++ that offers a quick and easy way to harness the power of both multicore and vector processing. The three Intel Cilk Plus keywords provide a simple yet surprisingly powerful model for parallel programming, while runtime and template libraries offer a well-tuned environment for building parallel applications.

Click here for sample code, contributed libraries, open specifications and other information from the Cilk Plus community.

Go to "**Intel® C++ Compiler Code Samples**" to see real-world applications that utilize the Intel® Cilk™ Plus.

Intel Cilk Plus allows you to:

- Write parallel programs using a simple model: With only three keywords to learn, C and C++ developers move quickly into the parallel programming domain.
- Identify data parallelism by using simple array notations that include elemental function capabilities.
- Leverage existing serial tools: The serial semantics of Intel Cilk Plus allows you to debug in a familiar serial debugger.
- Scale for the future: The runtime system operates smoothly on systems with hundreds of cores. Tools are available to analyze your application and predict how well it will scale.



Available in:

Intel® Parallel Studio XE
Intel® System Studio
Intel® Integrated Native Developer Experience 2015 Build Edition for OS X

OS Support

Windows*
Linux*
OS X*

Languages

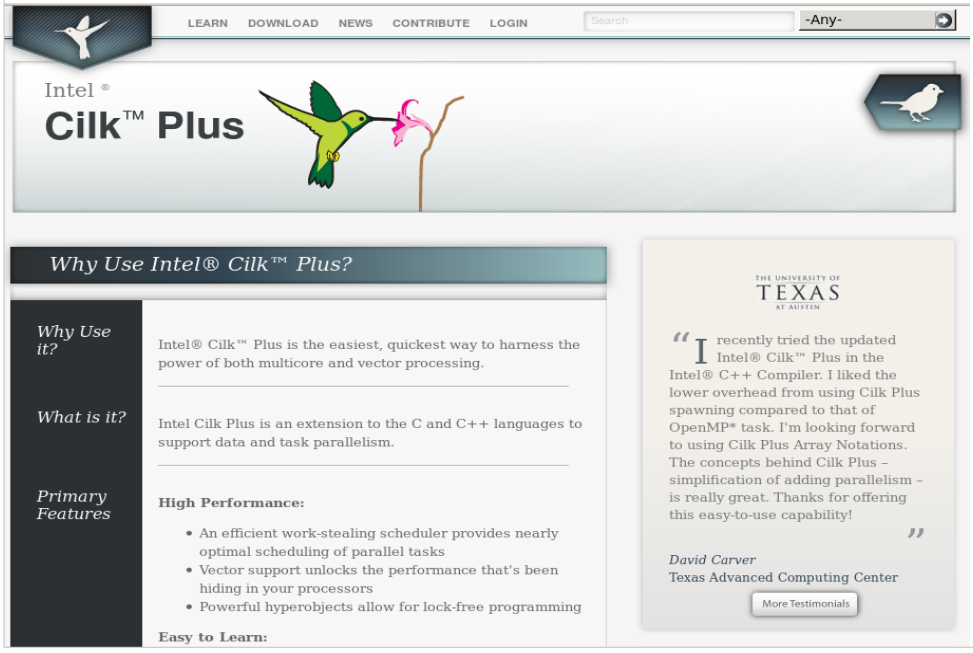
C, C++

Related Content

Documentation
Tutorial
Forums
Blogs

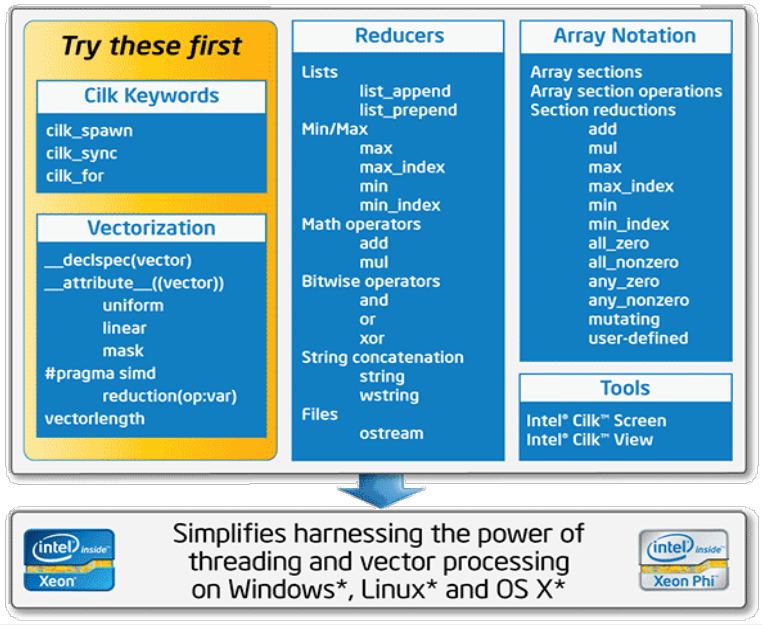
Guy: http://cilkplus.org/

האתר הזה חדל מלהתקיים...

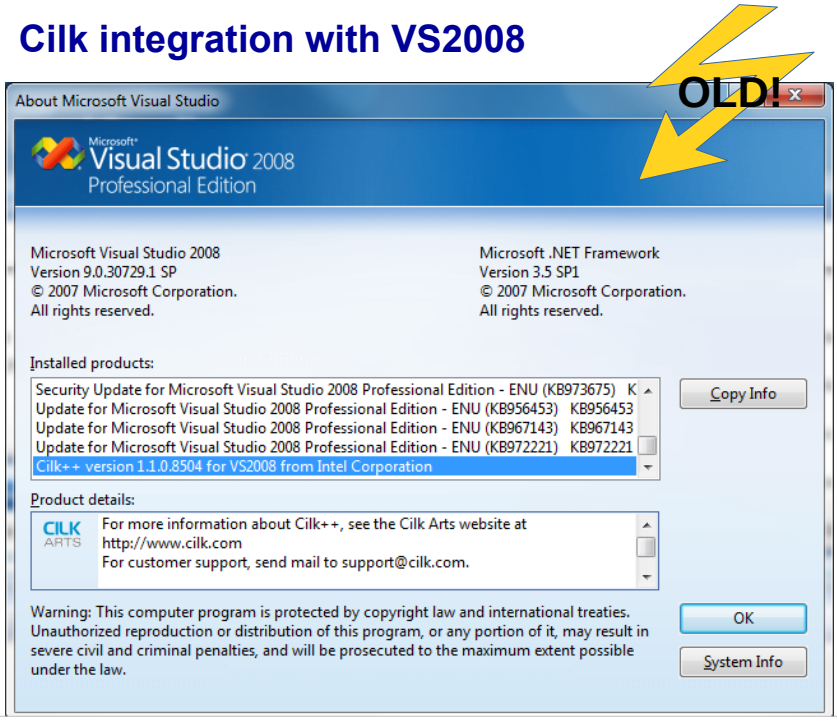


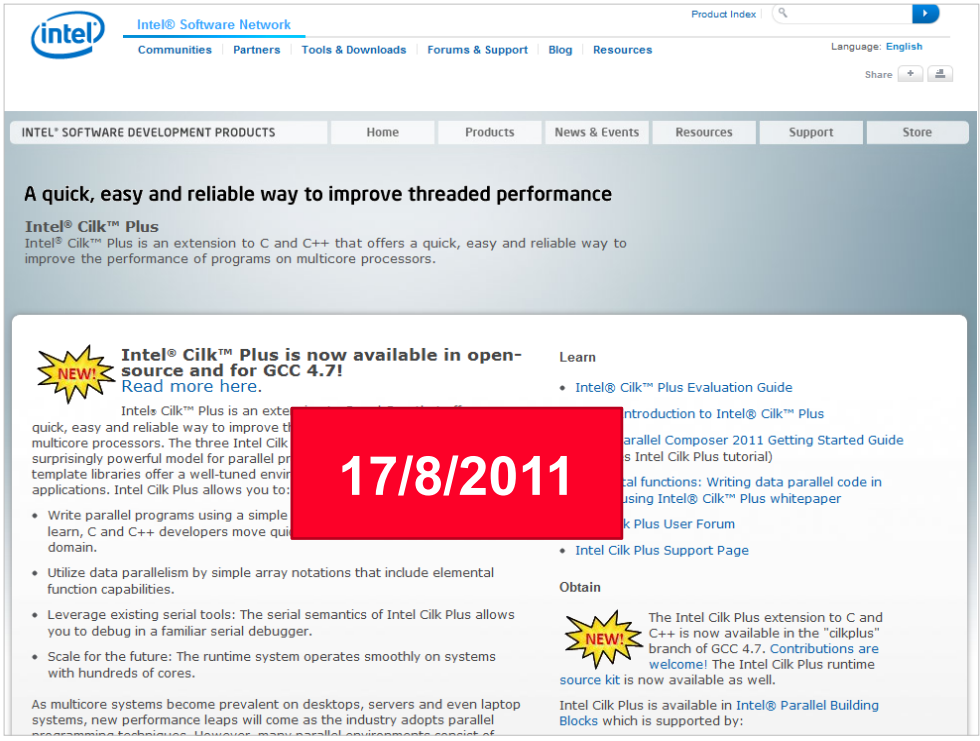
Intel® Cilk™ Plus

C/C++ compiler extension for simplified parallelism

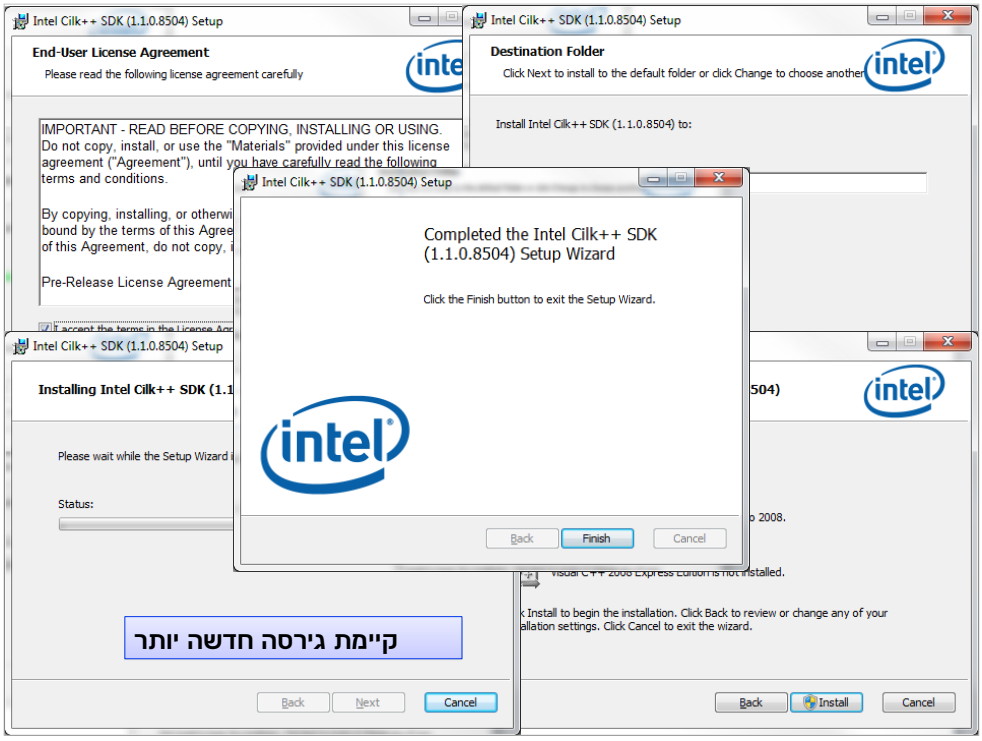


Cilk integration with VS2008



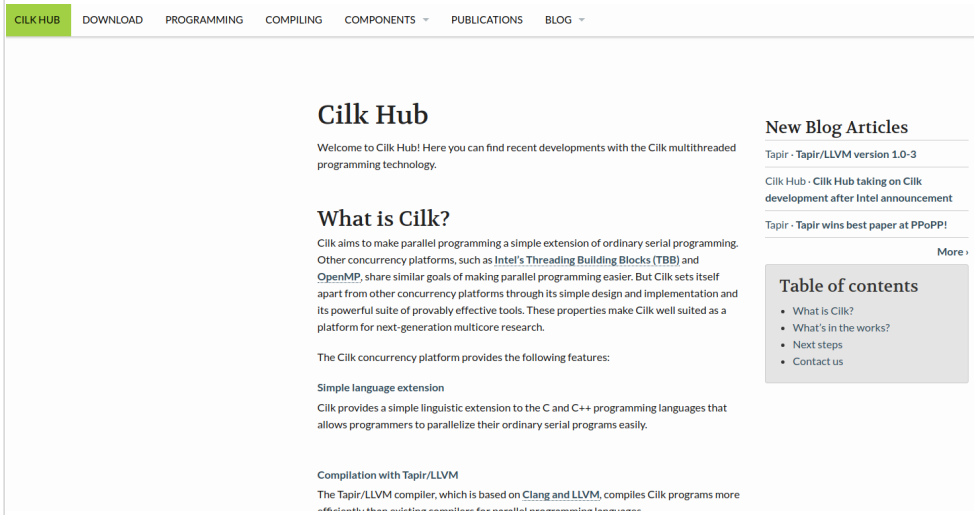

Intel® Software Network
Communities | Partners | Tools & Downloads | Forums & Support | Blog | Resources
Product Index
Language: English
Share
INTEL® SOFTWARE DEVELOPMENT PRODUCTS
Home
Products
News & Events
Resources
Support
Store
A quick, easy and reliable way to improve threaded performance
Intel® Cilk™ Plus
Intel® Cilk™ Plus is an extension to C and C++ that offers a quick, easy and reliable way to improve the performance of programs on multicore processors.
NEW!
Intel® Cilk™ Plus is now available in open-source and for GCC 4.7!
Read more here.
17/8/2011
Learn
Intel® Cilk™ Plus Evaluation Guide
Obtain
NEW!
The Intel Cilk Plus extension to C and C++ is now available in the "cilkplus" branch of GCC 4.7. Contributions are welcome! The Intel Cilk Plus runtime source kit is now available as well.
Intel Cilk Plus is available in Intel® Parallel Building Blocks which is supported by:


intel Developer Zone
Join Today
Log in
Development
Tools
Resources
Search our content library...
Home › Forums › Intel® Software Development Products › Intel® Cilk™ Plus
2017
Intel Cilk Plus is being deprecated
Hansang B. (Intel) Wed, 09/20/2017 - 08:03
Intel® Cilk™ Plus - an extension to the C and C++ languages to support data and task parallelism - is being deprecated in the 2018 release
highly recommended that you start migratin
TBB). For more information see Migrate Yo
technology continues at MIT's Cilk Hub.
Intel ®
Cilk™ Plus
Announcements
Cilk Deprecation
Intel® Cilk™ Plus – an extension to the C and C++ languages to support data and task parallelism – is being deprecated in the 2018 release of Intel® Software Development Tools. It will remain in deprecation mode in the Intel® C++ Compiler for an extended period of two years. It is highly recommended that you start migrating to standard parallelization models such as OpenMP* and Threading Building Blocks (TBB). For more information see Migrate Your Application to use OpenMP* or TBB Instead of Intel® Cilk™ Plus. Research into Cilk technology continues at MIT's Cilk Hub.

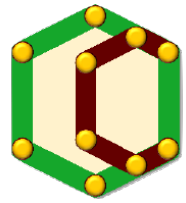


קיימת גירסה חדשה יותר

http://cilk.mit.edu/



OpenCilk



Home page: https://cilk.mit.edu/
Download from here:
https://github.com/OpenCilk/opencilk-project/releases/tag/opencilk/v1.0

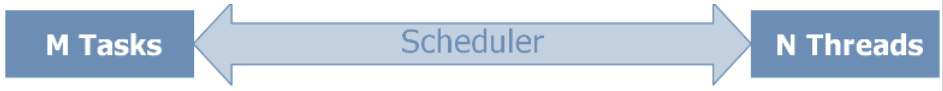
Installation directory on my laptop:
#/home/telzur/science/opt/OpenCilk-10.0.1-Linux

#For demos type:

```
$ export PATH=/home/telzur/science/opt/OpenCilk-10.0.1-Linux/bin:$PATH

$ clang -o fib ./fib.c -fopencilk
$ CILK_NWORKERS=4 time ./fib
```

An example with profiling:

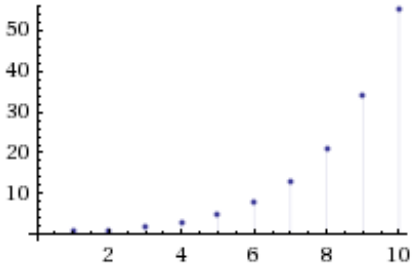
```
$ clang -o vector_add ./vector_add.cpp -fopencilk -fcilktool=cilkscale \
-L/usr/lib/gcc/x86_64-linux-gnu/9/ -lm
```



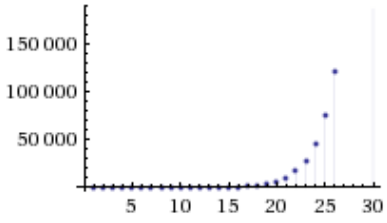
Source: http://wwwsfb.tpi.uni-jena.de/Events/PHSP11/slides/intel_cilk_tutorial.pdf

Fibonachi (Fibonacci)

Try:
http://www.wolframalpha.com/input/?i=fibonacci+number



(*n* from 2 to 10)



(*n* from 5 to 30)

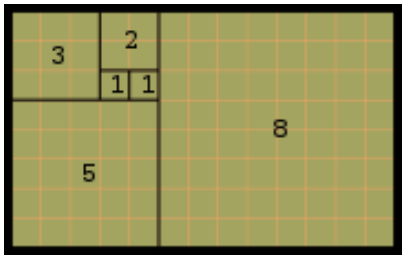
n	F_n
0	0
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55

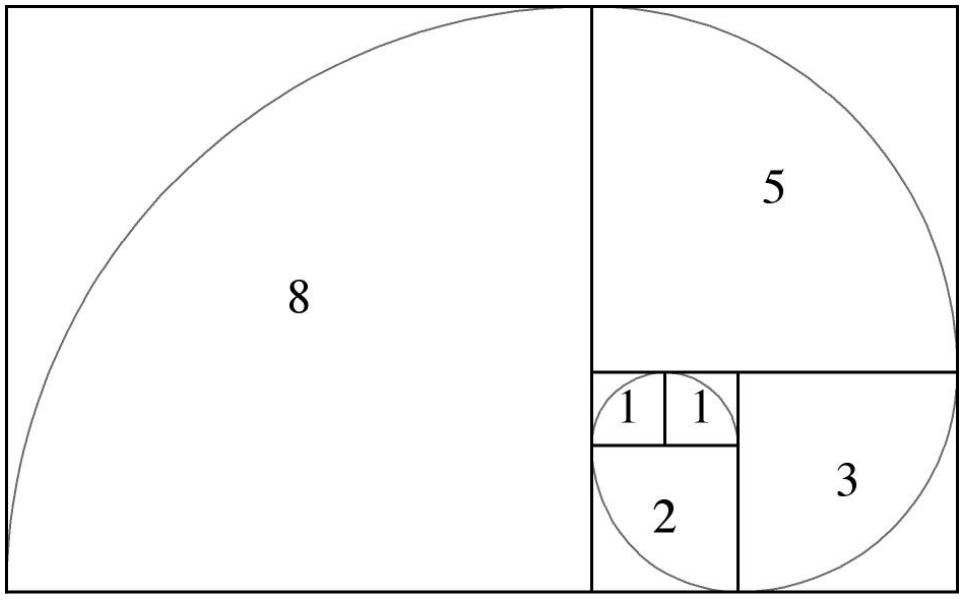
Fibonachi Numbers
serial version

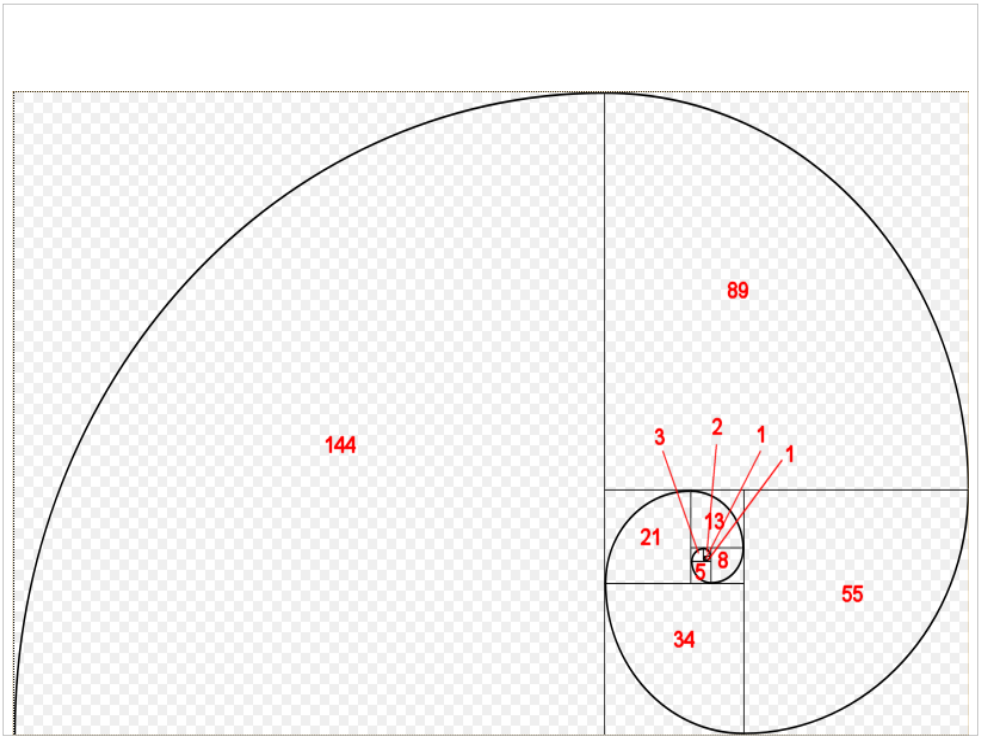
```
// 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
// Serial version
// Credit: http://myxman.org/dp/node/182

long fib_serial(long n) {

  if (n < 2) return n;

  return fib_serial(n-1) + fib_serial(n-2);

}
```

Cilk++ Fibonachi (Fibonacci)

```
#include <cilk.h>
#include <stdio.h>


long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    x = cilk_spawn fib_parallel(n-1);
    y = fib_parallel(n-2);
    cilk_sync;
    return (x+y);
}

int cilk_main()
{
    int N=50;
    long result;
    result = fib_parallel(N);
    printf("fib of %d is %d\n",N,result);
    return 0;
}
```




8
5
1
1
2
3


144
89
3
2
1
1
13
21
8
5
55
34

Cilk++

Simple, powerful expression of task parallelism:
***cilk_for* – Parallelize for loops**
***cilk_spawn* – Specify the start of parallel execution**
***cilk_sync* – Specify the end of parallel execution**

http://software.intel.com/en-us/articles/intel-cilk-plus/

Cilk_spawn

ADD PARALLELISM USING CILK_SPAWN

The cilk_spawn keyword indicates that a function (the *child*) may be executed in parallel with the code that follows the cilk_spawn statement (the *parent*). Note that the keyword *allows* but does not *require* parallel operation. The Cilk++ scheduler will dynamically determine what actually gets executed in parallel when multiple processors are available. The cilk_sync statement indicates that the function may not continue until all cilk_spawn requests in the same function have completed. cilk_sync does not affect parallel strands spawned in other functions.

Fibonacci Example: Creating Parallelism

```
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 elision

=הַשׁמָטָה

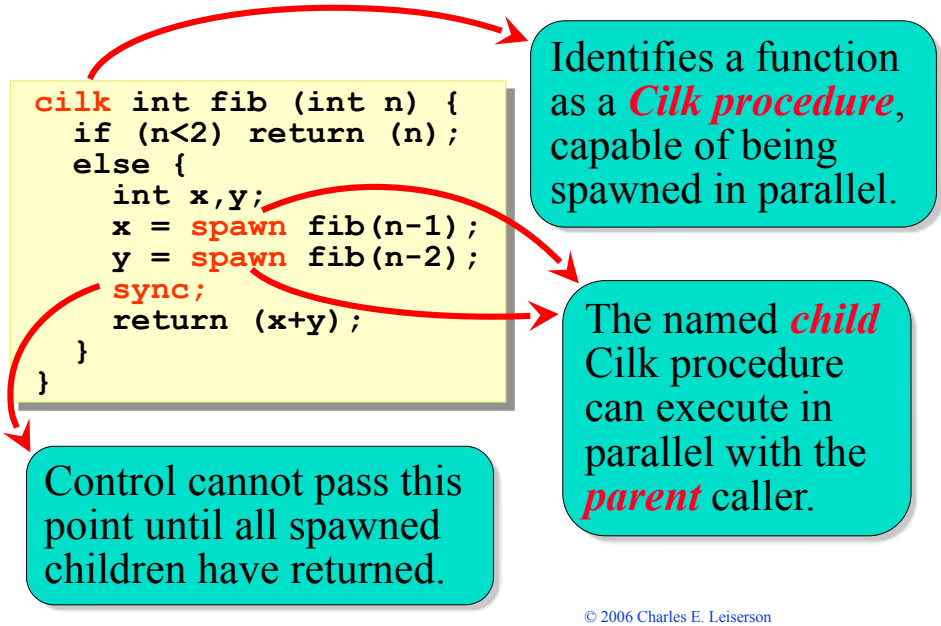
Cilk code

```
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);
  }
}
```

Cilk is a ***faithful*** extension of C. A Cilk program's ***serial elision*** is always a legal implementation of Cilk semantics. Cilk provides *no* new data types.

Elision = השמטה

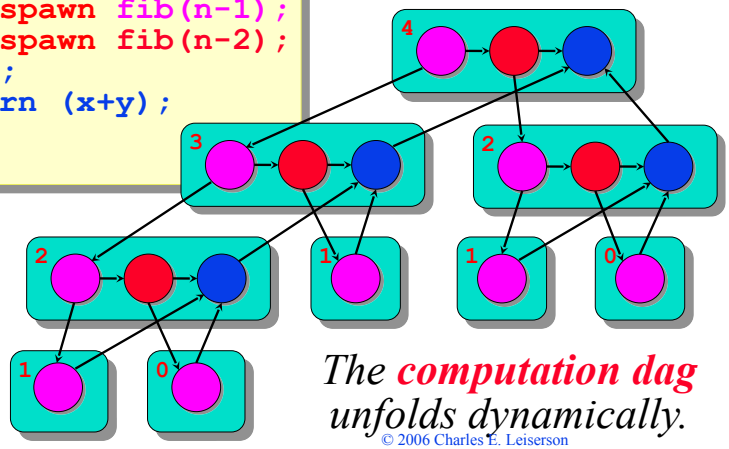
Basic Cilk Keywords



Dynamic Multithreading

```
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);
  }
}
```

Example: `fib(4)`



processors are virtualized

The computation dag unfolds dynamically.

On the Hobbits

```
Demo: /users/agnon/misc/tel-zur/cilk
-bash-4.1$ gcc -o fib ./fib.c              // serial version
-bash-4.1$ cilk++ -o fib_cilk ./fib_cilk.c //parallel version
-bash-4.1$  ./fib
Fibonacci of 45 is 1134903170
-bash-4.1$ export CILK_NWORKERS=4
 Bash-4.1$ #on other shells: setenv CILK_NWORKERS 4
-bash-4.1$  ./fib_cilk
Fibonacci of 45 is 1134903170
```

On the Hobbits (cont')

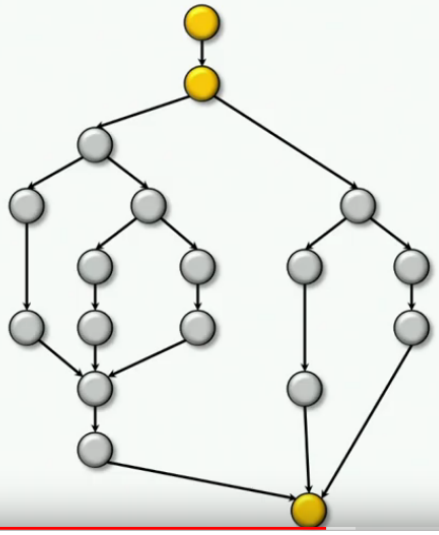
```
$ cilkscreen -a -r report.txt ./fib_cilk

$ more ./report.txt
Cilkscreen Race Detector V1.1.0, Build 8503
No errors found by Cilkscreen
```

On the Hobbits (cont')

```
cilkview -verbose -plot gnuplot ./fib_cilk
cilkview: CILK_NPROC=16 /usr/local/cilk/bin/../lib64/pinbin -ifeellucky -t
/usr/local/cilk/bin/../lib64/workspan.so -- ./fib_cilk
fibonacci of 15 is 610
Whole Program Statistics:

Cilkview Scalability Analyzer V1.1.0, Build 8503
1) Parallelism Profile
   Work :                    3,938,736 instructions
   Span :                    3,690,448 instructions
   Burdened span :               3,793,650 instructions
   Parallelism :                 1.07
   Burdened parallelism :            1.04
   Number of spawns/syncs:           986
   Average instructions / strand :       1,331
   Strands along span :              29
   Average instructions / strand on span :   127,256
   Total number of atomic instructions :     10
   Frame count :                 2962
2) Speedup Estimate
   2 processors:     0.76 - 1.07
   4 processors:     0.68 - 1.07
   8 processors:     0.64 - 1.07
   16 processors:    0.63 - 1.07
   32 processors:    0.62 - 1.07
```

Quantifying Parallelism

What is the parallelism of this computation?

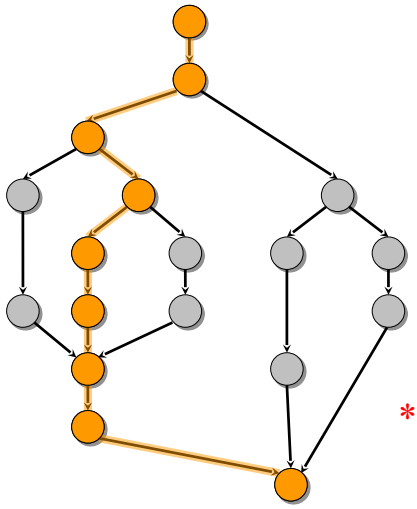


Amdahl's Law says that since the serial fraction is 3/18 = 1/6, the speedup is upper-bounded by 6.

מסתבר שהחסם על ההאצה על-פי חוק אמדהל הוא די גס.
הנבוי כאן הוא אינו מספיק טוב כפי שתכף נראה

Algorithmic Complexity Measures

T_P = execution time on P processors



T_1 = ***work***

T_∞ = ***span****

LOWER BOUNDS

- $T_P \geq T_1/P$
- $T_P \geq T_\infty$

*Also called ***critical-path length*** or ***computational depth***.

Tinfinity = time with infinite number of processors

For p processors:

$1*T_1 = work_1$

$p*T_p=Work_p$

$p*T_p \geq work_1=T_1$

היזכרו בהגדרת העבודה (שיעור מס' 1)

חוק העבודה: $T_p \geq T_1/p$

חוק הספאן

$T_p \geq T_\infty$

Speedup

Definition: T_1/T_P = ***speedup*** on P processors.

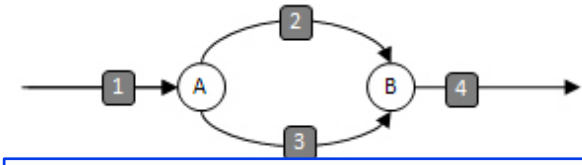
If $T_1/T_P = O(P) \le P$, we have ***linear speedup***;
$= P$, we have ***perfect linear speedup***;
$> P$, we have ***superlinear speedup***, which is not possible in our model, because of the lower bound $T_P \ge T_1/P$.

T_1/T_∞ = ***parallelism***
= *the average amount of work per step along the span.*

כמה עבודה עודפת יש לנו מעבר לנתיב הקריטי?

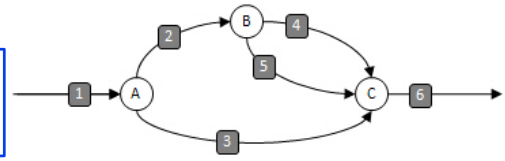
Strands and Knots

A Cilk++ program fragments



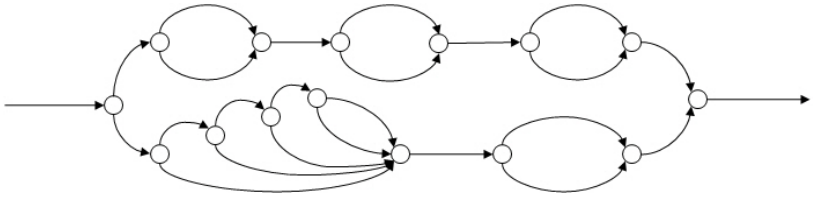

```
...
do_stuff_1(); // execute strand 1
cilk_spawn func_3(); // spawn strand 3 at knot A
do_stuff_2(); // execute strand 2
cilk_sync; // sync at knot B
do_stuff_4(); // execute strand 4 ...
```

DAG with two spawns (labeled A and B) and one sync (labeled C)

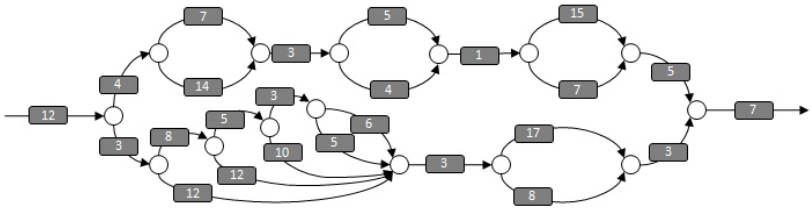


Strand = רצועה

a more complex Cilk++ program (DAG):



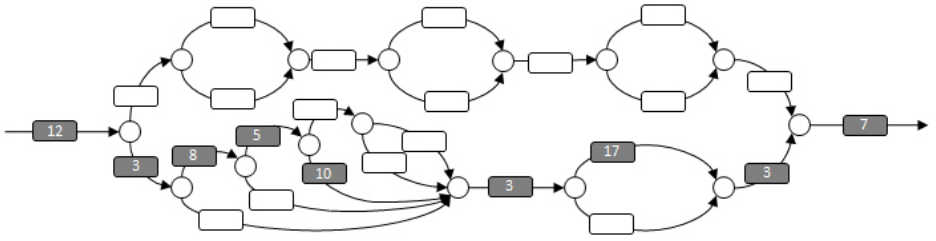
Let's add labels to the strands to indicate the number of milliseconds it takes to execute each strand



In ideal circumstances (e.g., if there is no scheduling overhead) then, if an unlimited number of processors are available, this program should run for 68 milliseconds.

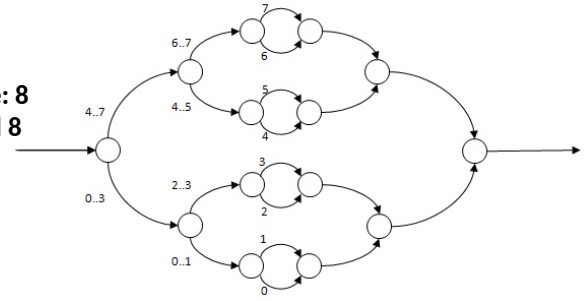
Work and Span

Work
The total amount of processor time required to complete the program is the sum of all the numbers. We call this the *work*.
In this DAG, the work is 181 milliseconds for the 25 strands shown, and if the program is run on a single processor, the program should run for 181 milliseconds.
Span
Another useful concept is the *span*, sometimes called the *critical path length*. The span is the *most expensive* path that goes from the beginning to the end of the program. In this DAG, the span is 68

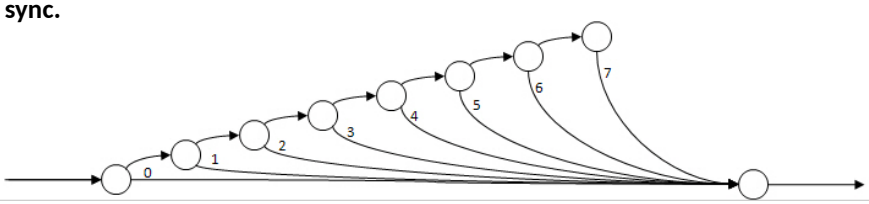


divide-and-conquer strategy

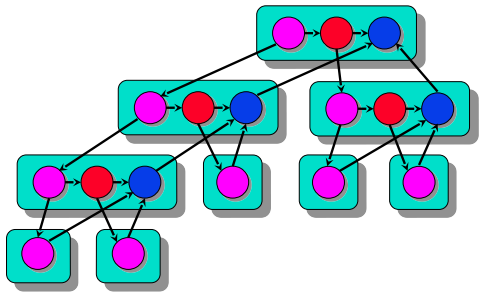
cilk_for
Shown here: 8 threads and 8 iterations



Here is the DAG for a serial loop that spawns each iteration. In this case, the work is not well balanced, because each child does the work of only one iteration before incurring the scheduling overhead inherent in entering a sync.



Example: `fib(4)`

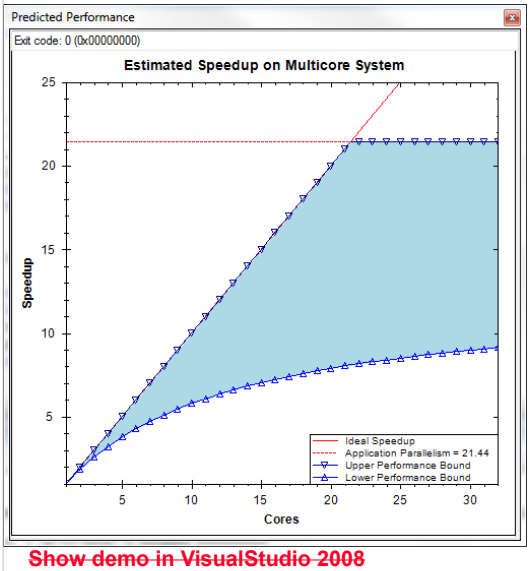


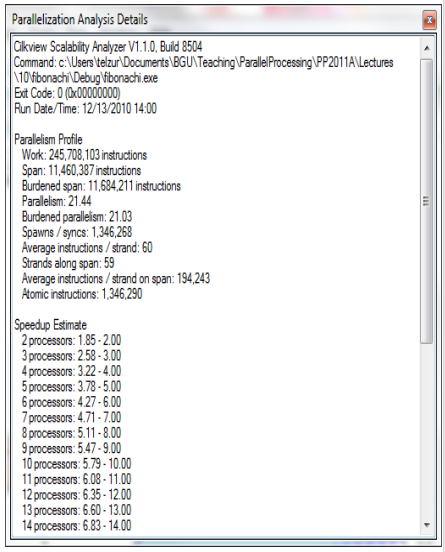
Assume for simplicity that each Cilk thread in `fib()` *takes unit time to execute.*

Work: $T_1 = 17$
Span: $T_\infty = 8$
Parallelism: $T_1/T_\infty = 2.125$

Using many more than 2 processors makes little sense.

Cilkview Fn(30)





~~Show demo in VisualStudio 2008~~ ...התייש|

~~C:\Users\telzur\Documents\BGU\Teaching\ParallelProcessing\PP2011A\Lectures\10\fibonachi\fibonachi\fibonachi.vcproj~~

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

C

```
void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    vadd (A, B, n/2);
    vadd (A+n/2, B+n/2, n-n/2);

  }
}
```

Parallelization strategy:

1. Convert loops to recursion.

Demo (28/12/20): vector_add demo under Intel environment (see readme_guy.txt)

Parallelizing Vector Addition

C

```
void vadd (real *A, real *B, int n){
  int i; for (i=0; i<n; i++) A[i]+=B[i];
}
```

Cilk

```
cilk void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    spawn vadd (A, B, n/2);
    spawn vadd (A+n/2, B+n/2, n-n/2);
    sync;
  }
}
```

Parallelization strategy:

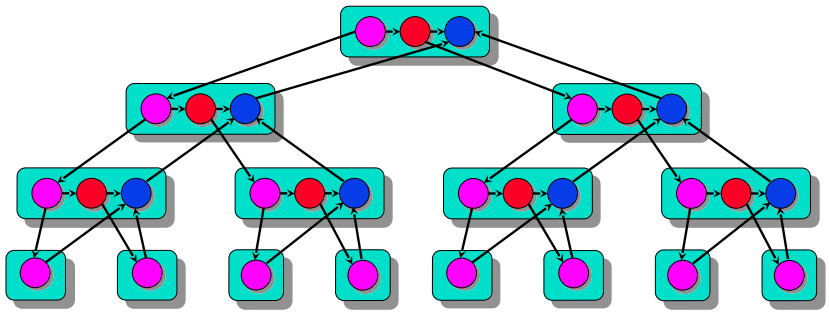
1. Convert loops to recursion.
2. Insert Cilk keywords.

Side benefit: D&C is generally good for caches!

D&C = Divide and Conquer

Vector Addition

```
cilk void vadd (real *A, real *B, int n){
  if (n<=BASE) {
    int i; for (i=0; i<n; i++) A[i]+=B[i];
  } else {
    spawn vadd (A, B, n/2);
    spawn vadd (A+n/2, B+n/2, n-n/2);
    sync;
  }
}
```



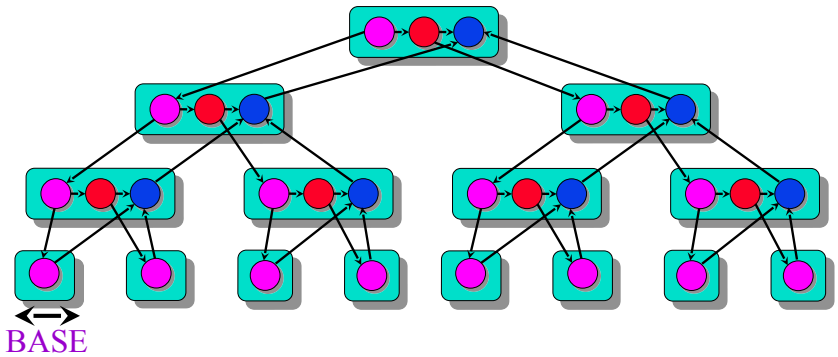
Vector Addition Analysis

To add two vectors of length n, where BASE = O(1):

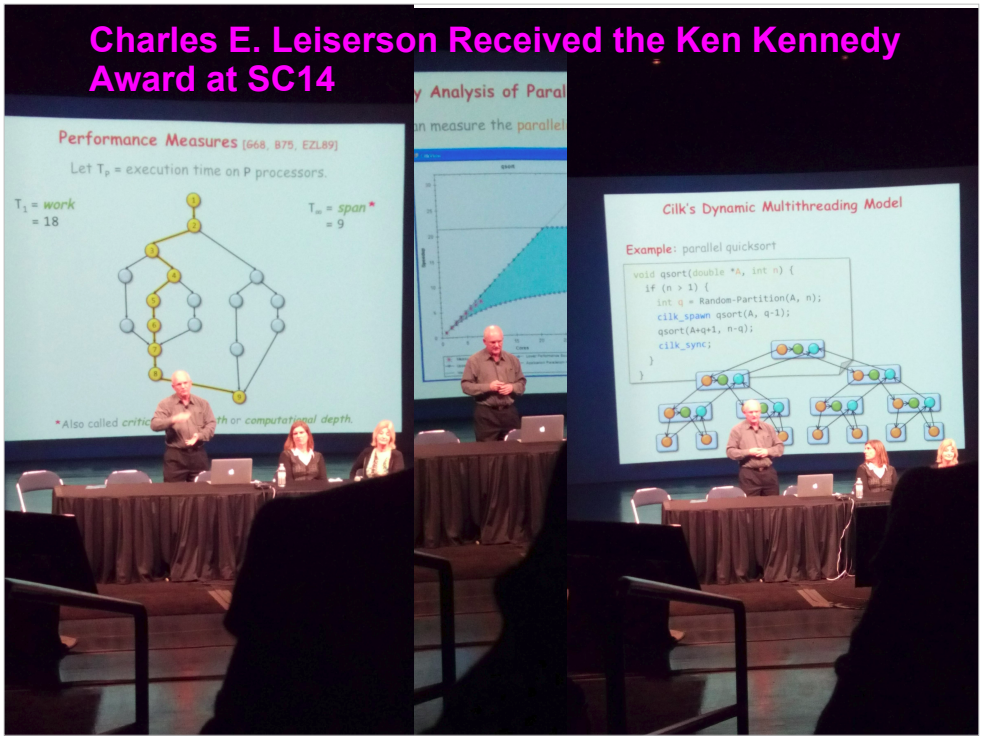
Work: $T_1 = \mathrm{O}(n)$

Span: $T_\infty = \mathrm{O}(\lg n)$

Parallelism: $T_1/T_\infty = \mathrm{O}(n/\lg n)$



Charles E. Leiserson Received the Ken Kennedy Award at SC14



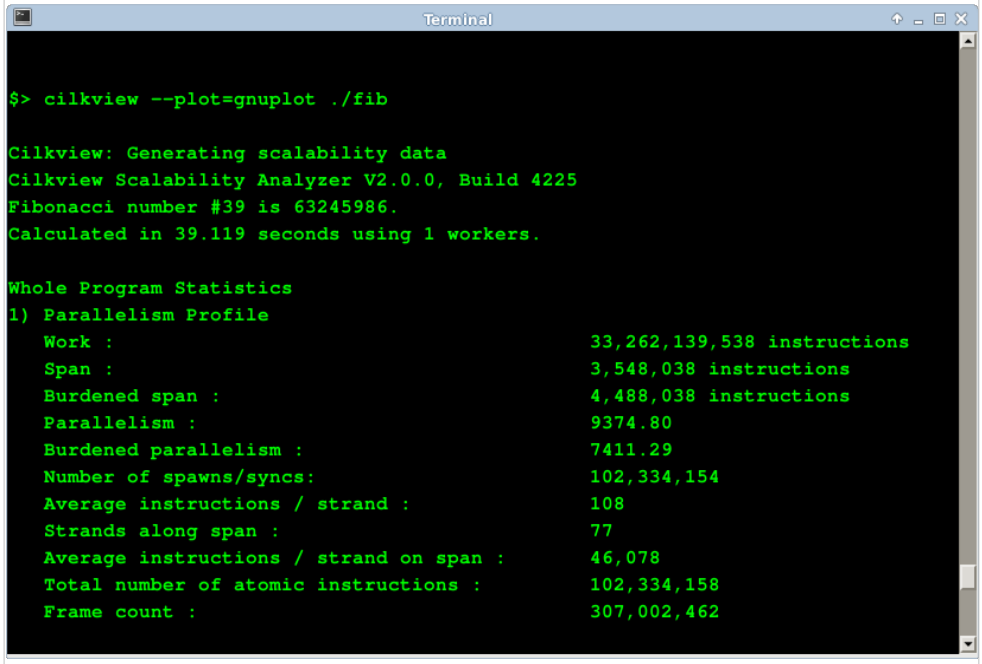
On my **LIFEBOOK** laptop
Folder:
/home/telzur/Documents/Teaching/BGU/PP/PP2016B/lectures/10/code/cilk

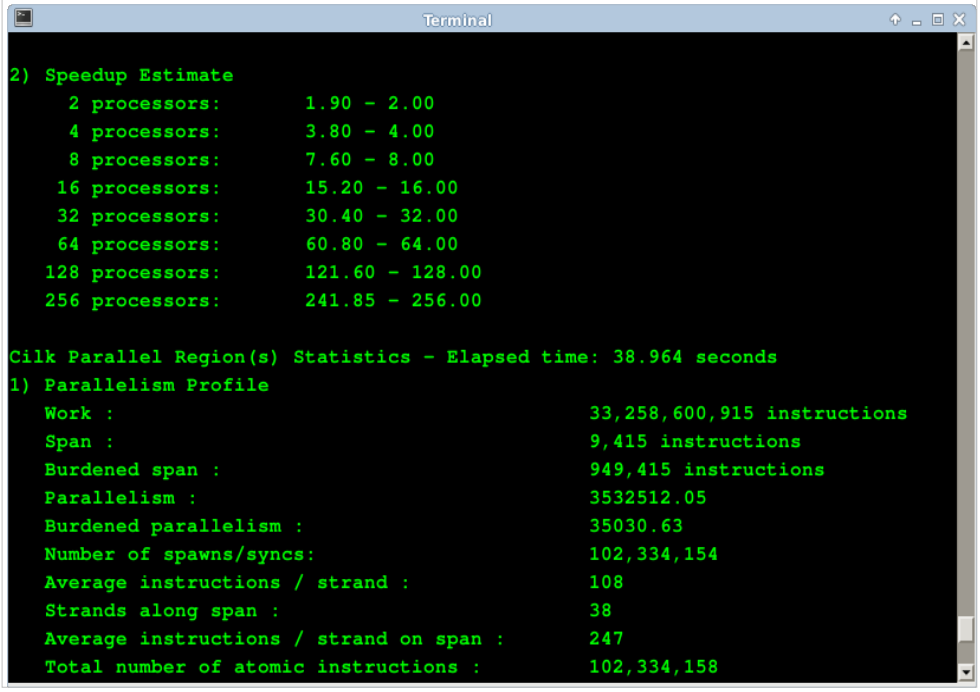
Compilation with GNU:

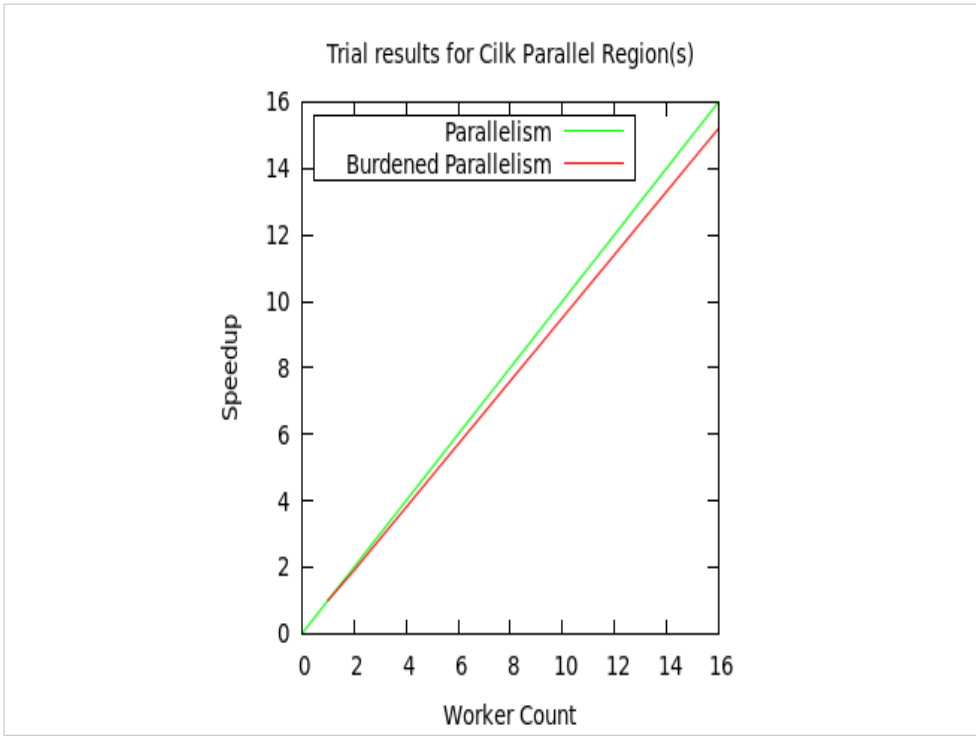
```
$CILK/bin/g++ -o fib -fcilkplus -L
$CILK/lib64 -lcilkrts -lstdc++ -I
$CILK/include/cilk ./fib.cpp
```

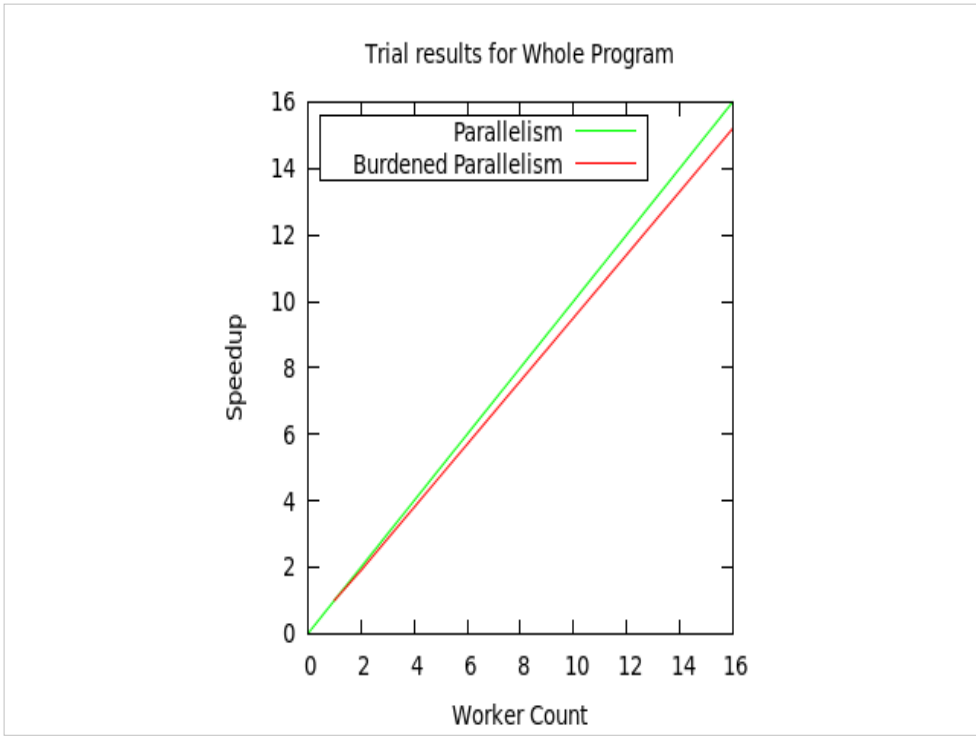
Compilation with Intel

```
/opt/intel/bin/icc  -o
fib_intel ./fib_intel.cpp

cilkview --plot=gnuplot ./fib
```


Terminal
$> cilkview --plot=gnuplot ./fib
Cilkview: Generating scalability data
Cilkview Scalability Analyzer V2.0.0, Build 4225
Fibonacci number #39 is 63245986.
Calculated in 39.119 seconds using 1 workers.
Whole Program Statistics
1) Parallelism Profile
Work : 33,262,139,538 instructions
Span : 3,548,038 instructions
Burdened span : 4,488,038 instructions
Parallelism : 9374.80
Burdened parallelism : 7411.29
Number of spawns/syncs: 102,334,154
Average instructions / strand : 108
Strands along span : 77
Average instructions / strand on span : 46,078
Total number of atomic instructions : 102,334,158
Frame count : 307,002,462


Terminal
2) Speedup Estimate
2 processors: 1.90 - 2.00
4 processors: 3.80 - 4.00
8 processors: 7.60 - 8.00
16 processors: 15.20 - 16.00
32 processors: 30.40 - 32.00
64 processors: 60.80 - 64.00
128 processors: 121.60 - 128.00
256 processors: 241.85 - 256.00
Cilk Parallel Region(s) Statistics - Elapsed time: 38.964 seconds
1) Parallelism Profile
Work : 33,258,600,915 instructions
Span : 9,415 instructions
Burdened span : 949,415 instructions
Parallelism : 3532512.05
Burdened parallelism : 35030.63
Number of spawns/syncs: 102,334,154
Average instructions / strand : 108
Strands along span : 38
Average instructions / strand on span : 247
Total number of atomic instructions : 102,334,158


Trial results for Cilk Parallel Region(s)
Parallelism
Burdened Parallelism
Speedup
Worker Count
0
2
4
6
8
10
12
14
16


Trial results for Whole Program
Parallelism
Burdened Parallelism
Speedup
Worker Count
0
2
4
6
8
10
12
14
16

Examples on my laptop (currently ASUS ROG, Linux mint20)

Directory: ~/lectures/10/cilk-fib

Examples:

```
g++-5 -o vec_add -fcilkplus ./vector_add.cpp
```

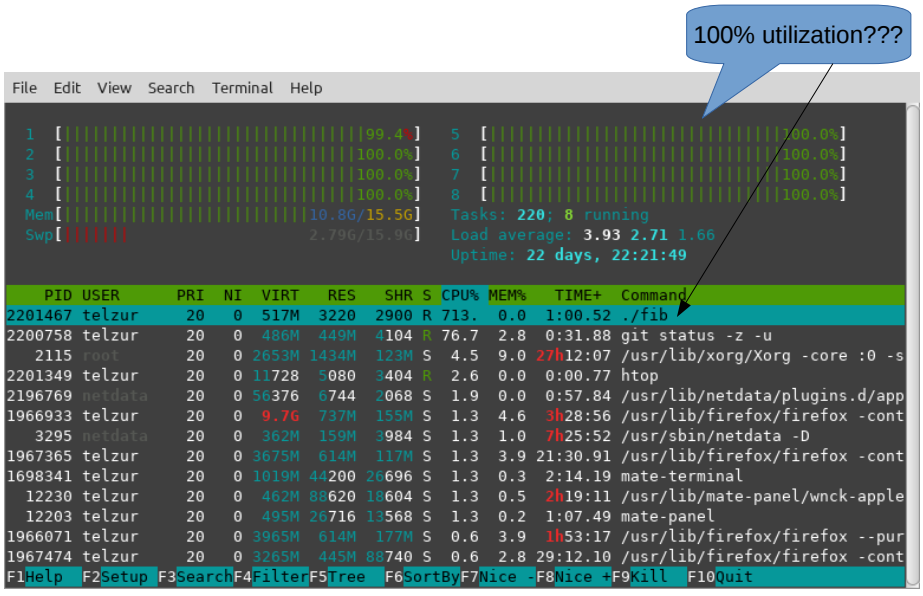
and:

```
g++-5 -o fib -fcilkplus ./fib.cpp
```

(see next slide)

```
# OpenCilk with Clang:
$ ~/science/opt/OpenCilk-10.0.1-Linux/bin/clang
-fopencilk ./fib_clang.c -o fib_clang

export CILK_NWORKERS=4
./fib
```

Performance issues with Cilk Fibonacci

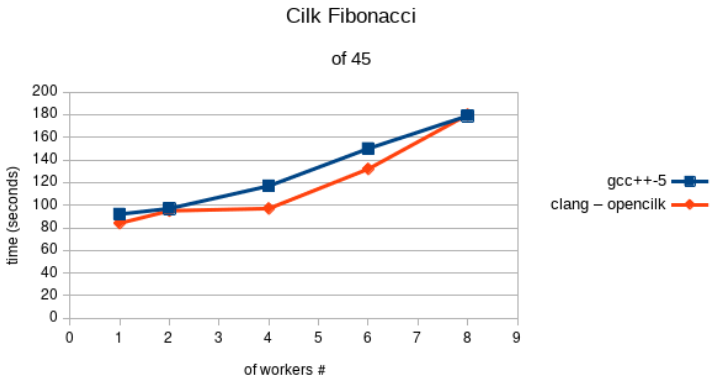


Cilk Fibonacci results

# workers	g++-5 time
1	92
2	97
4	117
6	150
8	179

מסקנה:
דוגמה יפה מבחינה פדגוגית
אבל ביצועים גרועים – ראו גם השקף הבא!

ההסבר:
הרבה זמן מתבזבז על יצירת המשימות אך כל משימה בפני עצמה דלה מאוד בחישוב...



https://cilk.mit.edu/programming/

Spawn and sync

Let us first examine the task-parallel keywords `cilk_spawn` and `cilk_sync`. Consider the following example code for a `fib` routine, which uses these keywords to parallelize the computation of the nth Fibonacci number.

class="pygments highlight" style="background: #f8f8f8;"> CILK

```
1  int64_t fib(int64_t n) {
2    if (n < 2) return n;
3    int x, y;
4    x = cilk_spawn fib(n - 1);
5    y = fib(n - 2);
6    cilk_sync;
7    return x + y;
8  }
```

Note to the algorithms police

The example `fib` routine is a terribly inefficient code for computing Fibonacci numbers. This `fib` routine computes the \(n\)th Fibonacci number using \(\Theta(\phi^n)\) work, where \(\phi\) denotes the golden ratio, while in fact this number can be computed using \(\Theta(\lg n)\) work. We use this example `fib` code simply for didactic purposes.

cilkscale

compile:

```
~/science/opt/OpenCilk-10.0.1-Linux/bin/clang -fopencilk
./fib_clang.c -o fib_clang -fcilktool=cilkscale -L/usr/lib/x86_64-
linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/9
```

```
~/….cilk-fib $ ./fib_clang
Fibonacci number #45 is 1134903170.
Calculated in 2543.843 seconds.
tag,work (seconds),span (seconds),parallelism,burdened_span
(seconds),burdened_parallelism
,1679.12,0.0928454,18085.1,0.0930579,18043.8
```

so far I couldn't use cilkscale is not working with graphics

```
export
LD_LIBRARY_PATH=/home/telzur/science/opt/OpenCilk-10.0.1-Linux/lib/
clang/10.0.1/lib/x86_64-unknown-linux-gnu:$LD_LIBRARY_PATH
 ~/science/opt/OpenCilk-10.0.1-Linux/bin/clang  -fopencilk ./fib_clang.c -o
fib_clang -fcilktool=cilkscale -fcilktool=cilkscale-benchmark -L/usr/lib/x86_64-
linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/9  -lclang_rt.cilkscale
-L/home/telzur/science/opt/OpenCilk-10.0.1-Linux/lib/clang/10.0.1/lib/x86_64-
unknown-linux-gnu
```

למידע נוסף:
https://cilk.scripts.mit.edu/pact21/opencilk-pact-2021.pdf

https://www.youtube.com/watch?v=a_R_DpsENfk&t=34s
MIT 6.172 Performance Engineering of Software Systems, Fall 2018
Instructor: Julian Shun
View the complete course: https://ocw.mit.edu/6-172F18
YouTube Playlist: https://www.youtube.com/playlist?...list

Professor Shun discusses races and parallelism, how cilkscale can analyze computation and detect .determinancy races, and types of schedulers