Multithreaded Programming in Cilk

Matteo Frigo

CILK ARTS

Multicore challanges

Development time:

- Will you get your product out in time?
- Where will you find enough parallel-programming talent?
- Will you be forced to redesign your application?

Application performance

- How will you achieve superior processor utilization?
- Will your solution scale as the number of processor cores increases?

Software reliability

- How will you test your multicore applications?

Cilk Arts' solution

Development time:

- No application redesign required.
- Can be learned in days by average programmers.
- Seamless path forward (and backward).

Application performance

- Best-in-class performance.
- Linear scaling as cores are added.
- Minimal overhead on a single core.

Software reliability

- Multithreaded version as reliable as the original.

About Cilk Arts

Mission:

To provide the easiest, quickest, and most reliable way to optimize application performance on multicore processors.

About:

- Incorporated in 2006 to commercialize 15 years of research.
- Launched in March 2007.
- Headquartered in Lexington, MA.

Team:

- Duncan McCallum, CEO.
- Prof. Charles E. Leiserson, CTO.
- Matteo Frigo, Chief Scientist.
- Steve Lewin-Berlin, VP Engineering and Operations.

Cilk

What is Cilk?

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

Applications:

- Virus shell assembly.
- Graphics rendering.
- *n*-body simulation.
- Chess programs.
- Dense and sparse linear algebra.
- Lattice-Boltzmann methods.
- Chip power grid analysis.

Provably good runtime system:

Automatically manages low-level aspects of parallel execution, including protocols, load balancing, and scheduling.

Fibonacci

C elision

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

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;
  }
}
```

Cilk is a **faithful** extension of C/C++. The **serial elision** of a Cilk program is a valid implementation. Cilk provides no new data types.

Basic Cilk keywords

Fibonacci

```
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`:

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

Basic Cilk keywords

Fibonacci

```
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:

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

spawn:

The child procedure can be executed in parallel with the parent.

Basic Cilk keywords

Fibonacci

```
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`:

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

`spawn`:

The child procedure can be executed in parallel with the parent.

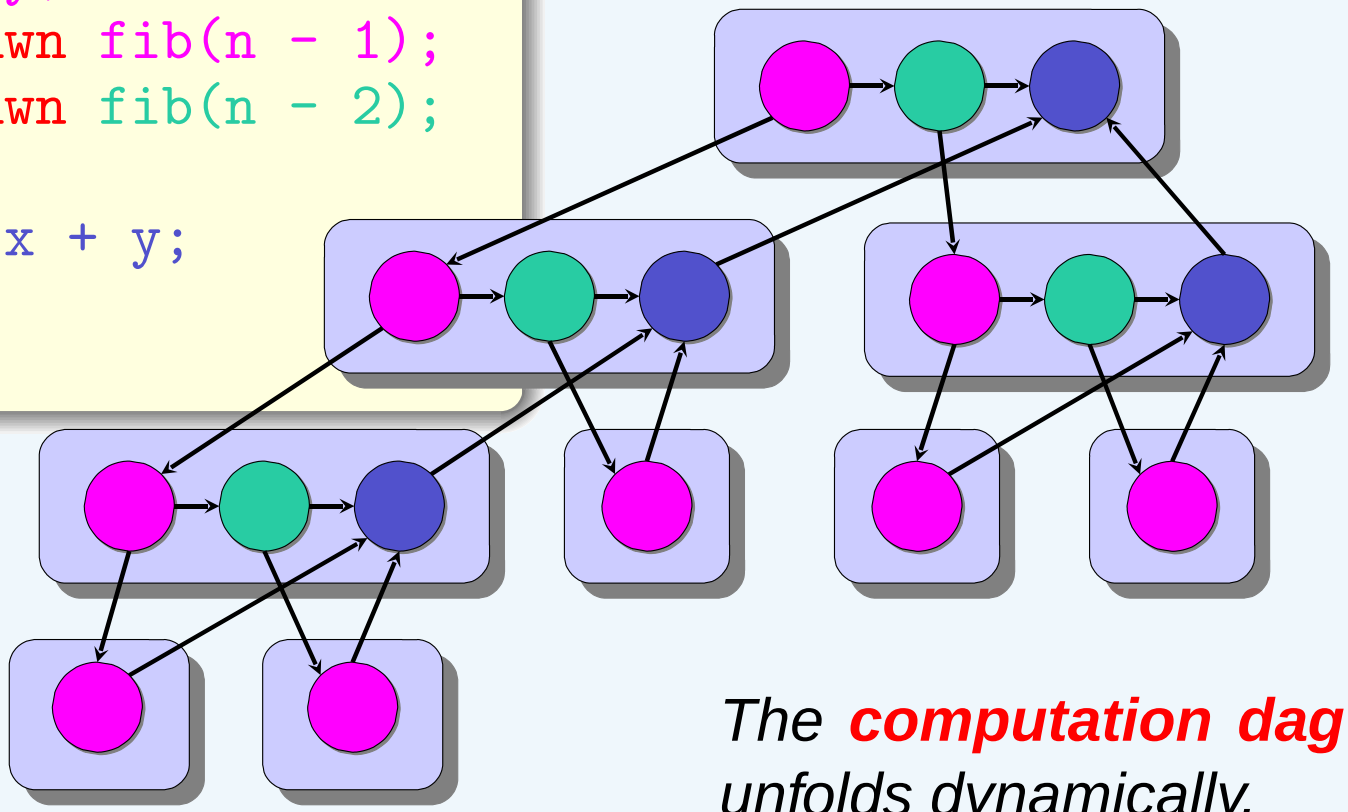
`sync`:

Cannot be passed until all spawned children have returned.

Dynamic multithreading

Example: fib(4)

```
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;
  }
}
```



*The **computation dag** unfolds dynamically.*

Cactus stack

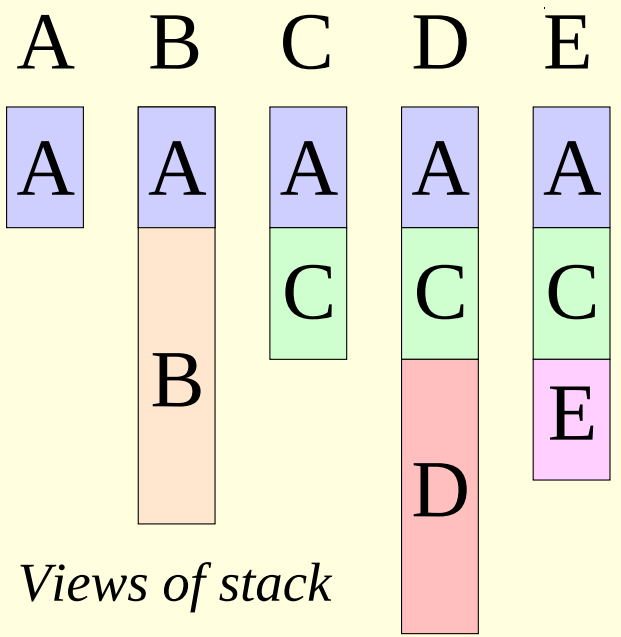
Cilk supports C's rule for pointers:

A pointer to stack space can be passed from parent to child, but not from child to parent. (Cilk also supports `malloc`/`new`.)

Cilk's cactus stack supports several stack views in parallel.

```
cilk void A(void)
{
   spawn B();
   spawn C();
}

cilk void C(void)
{
   spawn D();
   spawn E();
}
```



Views of stack

Cilk's thread scheduler

Randomized work stealing

Load-balances the computation at run-time.

Scales down

Slowdown of Cilk program on 1 processor vs. the serial elision is negligible.

Scales up

A mathematical proof guarantees near-perfect linear speed-up on applications with sufficient parallelism, as long as the architecture has sufficient memory bandwidth.

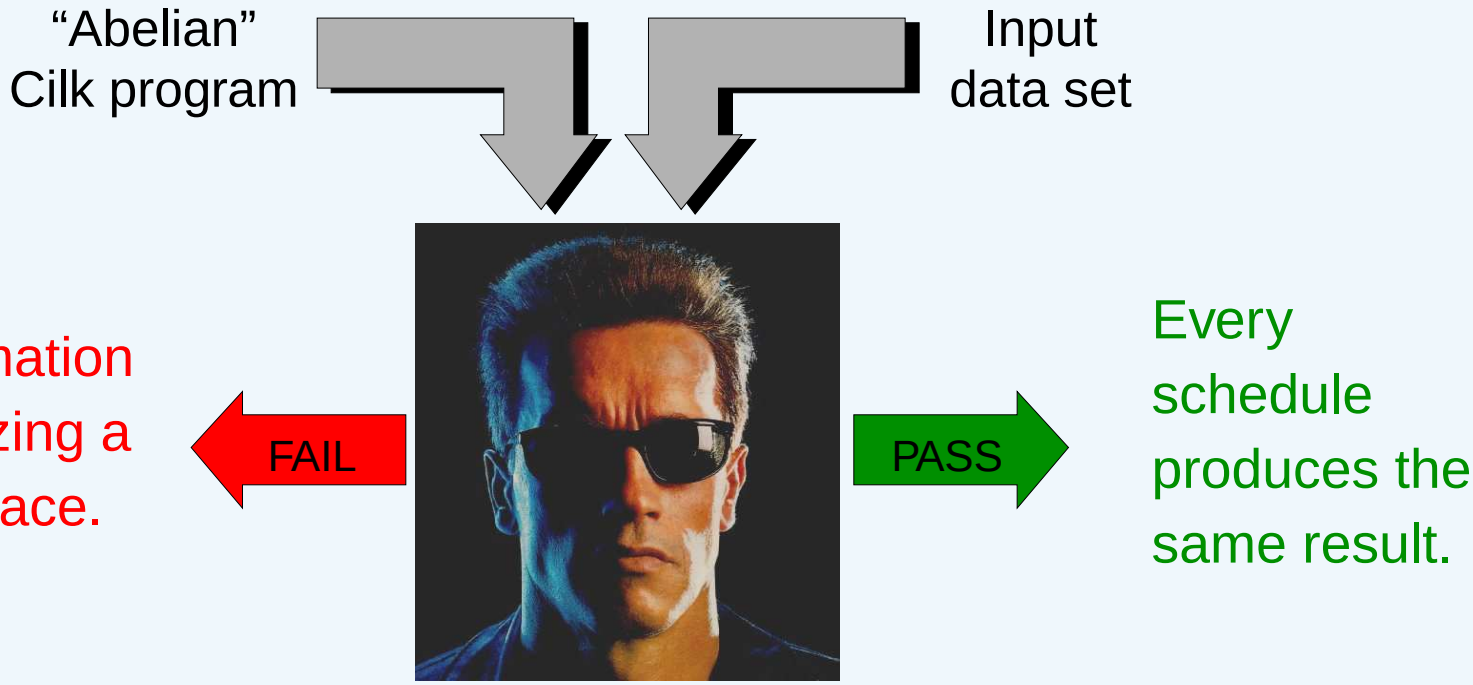
Spawn is cheap

A spawn/return in Cilk is over 450 times faster than a pthread create/exit and about 3 times slower than an ordinary C function call on contemporary x86 processors.

Debugging

The *Nondeterminator* debugging tool:

Provably guarantees to detect and localize data-race bugs.



Data race:

Occurs whenever a thread modifies a location and another thread, holding no locks in common, accesses the location simultaneously.

Advanced features

Support for nondeterministic programs:

- The **inlet** keyword specifies an internal function that can be called to incorporate a returned result into the parent frame in a nonstandard way when a spawned child returns.
- The **abort** keyword forces all spawned children to terminate abruptly.
- The **SYNCHED** pseudovariable tests whether a sync would succeed.
- A Cilk library provides **mutex locks** for atomicity.

HPC challenge benchmarks

2006 Class 2 Award for Best Overall Productivity

	HPL		DGEMM		STREAM		PTRANS		FFTE	
P	Gflop/s	η	Gflop/s	η	GB/s	η	GB/s	η	Gflop/s	η
1	5.2		5.1		0.8		0.7		0.7	
2	9.4	89	9.7	96	0.9	56	0.5	36	0.9	67
4	17.3	85	19.7	97	1.8	57	0.9	33	1.8	68
8	30.8	73	35.7	88	2.9	46	1.7	30	2.9	55
16	52.5	63	64.9	80	4.0	32	3.3	29	4.0	38
32	88.6	52	118.9	73	6.8	27	6.1	27	6.8	32
64	101.6	30	248.0	76	14.0	28	11.6	26	14.0	33
128			463.1	71	25.0	25	18.3	20	25.9	31
256			943.0	73	44.2	22	27.2	15	49.5	29

[Kuszmaul 2006], SGI Altix 3700.

Productivity

Only submission that implemented all six benchmarks

One programmer, about one week of work.

Cilk-ifying all six benchmarks required 137 keywords

Benchmark	MPI SLoC	Cilk SLoC	Dist. to Multicore
STREAM	658	58	11
PTRANS	2261	81	13
RandomAccess	1883	123	18
HPL	15608	348	41
DGEMM	184†	97	19
FFTE	1747	230	35

†MPI DGEMM uses the HPL parallel matrix multiplication.

Conclusion

Cilk offers:

- Compatibility with existing C/C++ serial code.
- Lightweight primitives for expressing parallelism.
- Robust load-balancing scheduler.
- Tools to ensure parallel correctness.