

# **CS528**

## **Amdhal's Law, and Cilk**

A Sahu

Dept of CSE, IIT Guwahati

# Outline

- Amdhal's Law
- Cilk

# Speedup and Efficiency in Multiprocessor System

# Speed up and efficiency

- $T_1$  = Time on Uni-processor
- $T_p$  = Time on  $p$  Processors
- Speed up

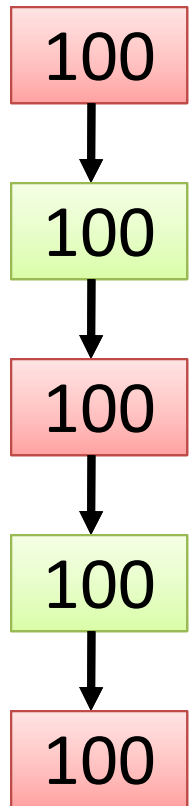
$$S_p = T_1 / T_p \leq p$$

- Efficiency

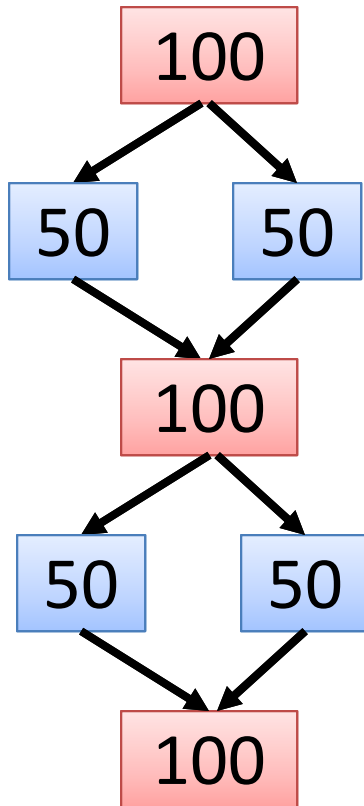
$$E_p = T_1 / (p \cdot T_p)$$

- Usually  $S_p < p$  or  $E_p < 1$  due to overhead
- Some time superliner speed up reported ( $S_p > p$  or  $E_p > 1$ )
  - Failure to use the best sequential algorithm
  - Advantage due to larger memory

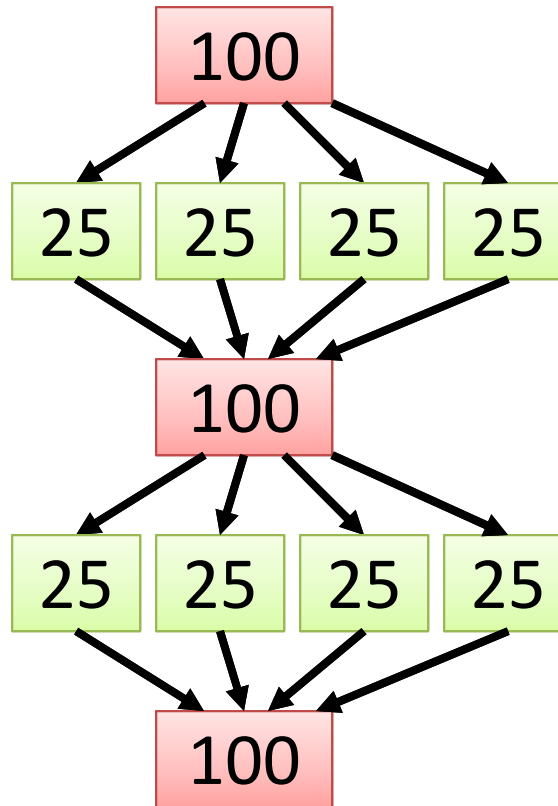
# Amdahl's law



Work 500,  
Time 500  
Sp=1X

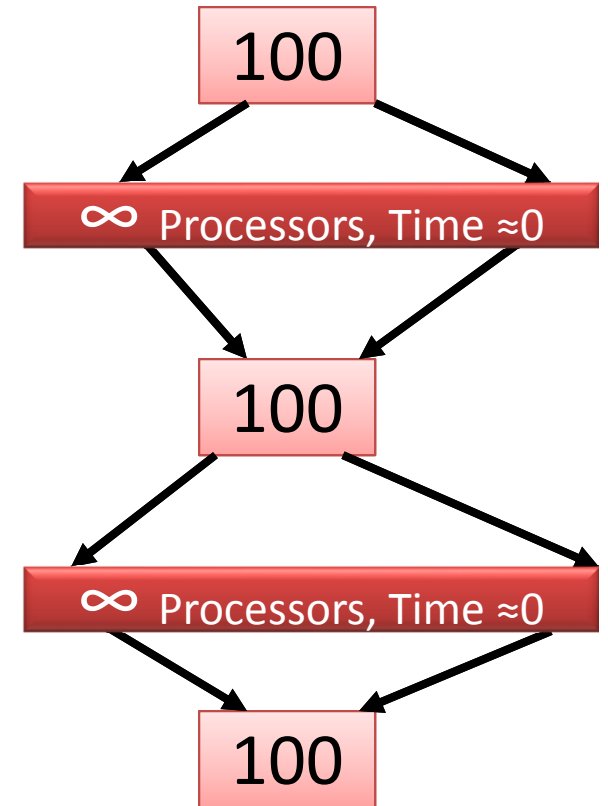


Work 500,  
Time 400  
Sp=1.25X



Work 500,  
Time 350  
Sp=1.4X

A Sahu



Work 500,  
Time 300  
Sp=1.7X

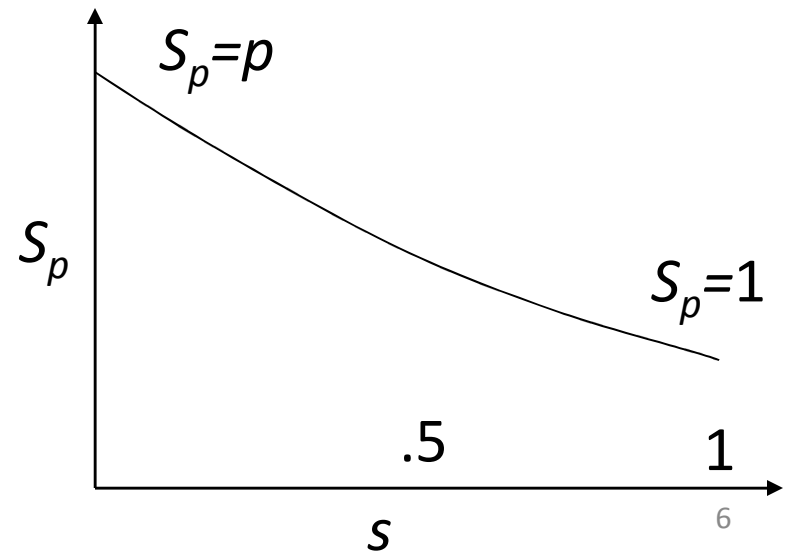
# Amdahl's Law

$$\text{Serial fraction} = s = \frac{T_s}{T_1}, T_p = T_s + \frac{T_1 - T_s}{p}$$

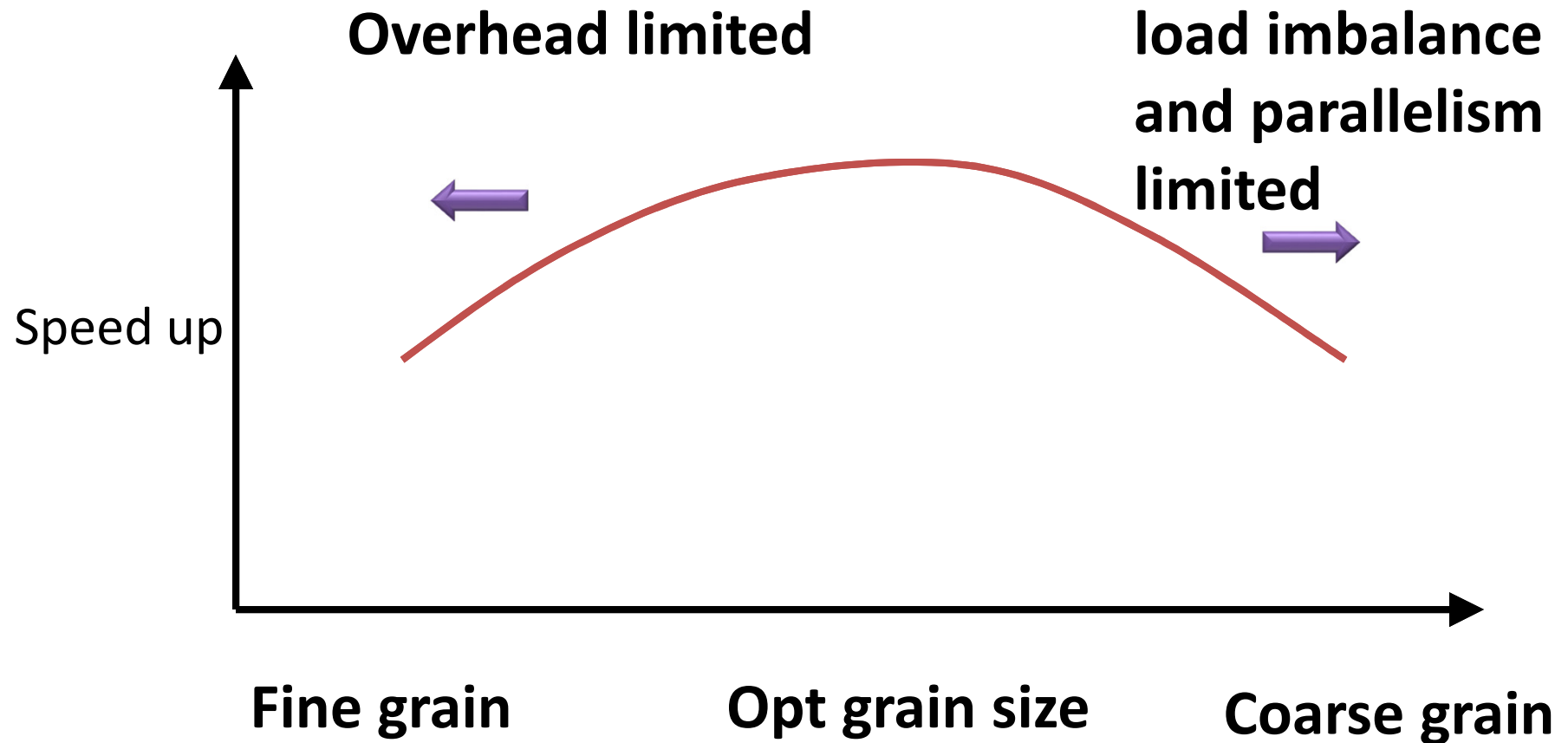
$$S_p = \frac{T_1}{T_p} = \frac{T_1}{T_s + \frac{T_1 - T_s}{p}} = \frac{T_1}{T_s(1 - \frac{1}{p}) + \frac{T_1}{p}}$$

$$Sp = \frac{1}{s(1 - \frac{1}{p}) + \frac{1}{p}} = \frac{p}{s(p-1) + 1}$$

$$\text{Lt } p \rightarrow \infty \quad S_p = \frac{1}{s}$$



# Grain size and performance



# Assumption behind Amdahl's Law

- All the processors are homogeneous
- All the communication costs are zero
- All the memory accesses takes unit time (PRAM)
- All the parallel section are purely parallel:  
Divisible load

$$S_p = \frac{1}{s \left(1 - \frac{1}{p}\right) + \frac{1}{p}} =$$

$$p \rightarrow \infty \quad S_p = \frac{1}{s}$$

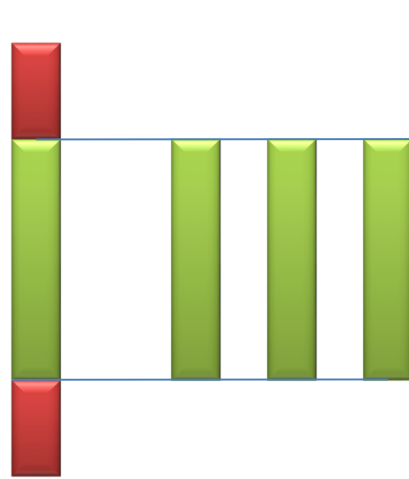


# ~~All the memory accesses takes unit time (PRAM)~~

- Memory Hierarchy: Cache Memory
- Suppose Application A run on 2Ghz Intel Pentium P4 uni-processor takes 10 minutes
- Same Application A run on 2Ghz Intel i5 Processor (Quad core) **may run much faster than 4X. Super linear Speedup**
  - Earlier cache size was 1MB in P4
  - Now cahce size is 4MB, the whole App A may be fit into Cache. No capacity misses...

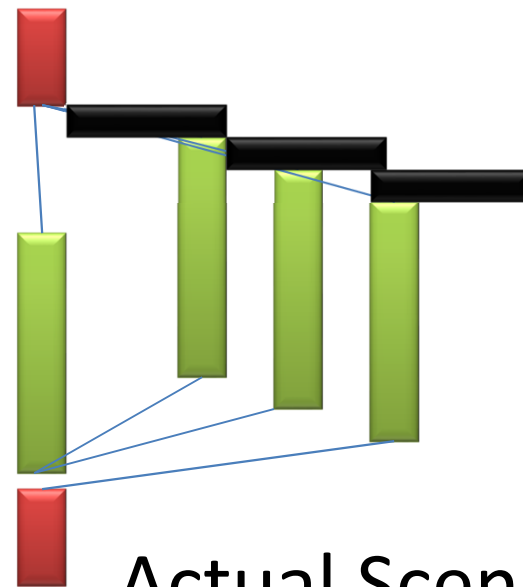
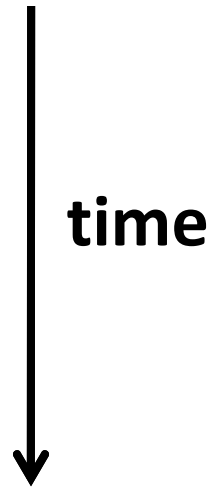
~~All the communication cost are zero~~

- Pthread Creation, Fork/Join takes significant amount of time



Ideal Scenario

$$T = T_{s1} + T_{s2} + T_p$$



Actual Scenario

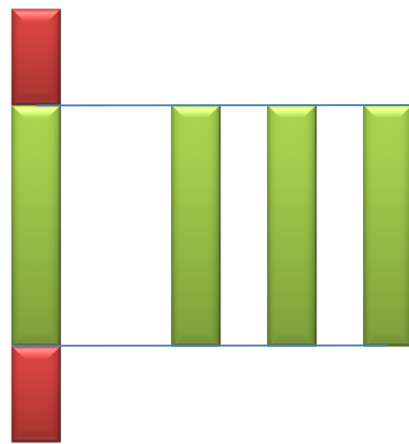
$$T = T_{s1} + T_{s2} + T_p + 3(T_f + T_j)$$

## ~~All the parallel section are purely parallel: Divisible load~~

- **Parallel threads** accessing to shared resources make it serial
- Using higher number of processor may need to collaborate and have more communication
- Application parallel section may not be scale up with processor : **Grain size**

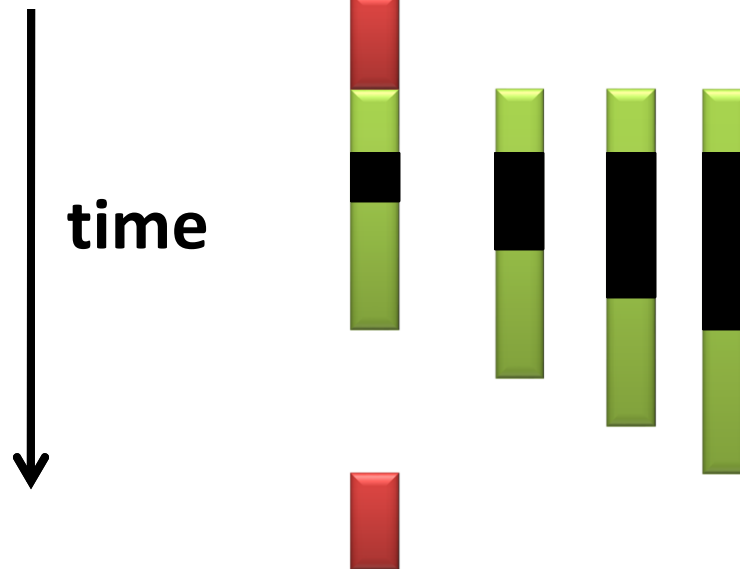
~~All the parallel section are purely  
parallel: Divisible load~~

- **Parallel threads** accessing to shared resources make it serial



Ideal Scenario

$$T = T_{s1} + T_{s2} + T_p$$



Actual Scenario with  
share Resource

$$T = T_{s1} + T_{s2} + T_p + 3(T_{cs})$$

~~All the processors are homogeneous~~

- Asymmetric Processing Environment
- One **big** core and many **small or tiny** cores
- Intel Xeon : 8 big cores
- GPU : 4/8 **big** cores+ 2000 **tiny** cores
- Intel Phi : 4/8 **big** cores(host )+ 250 **small** cores



Grain Size

# CS528

## Cilk

**Slides are adopted from**

<http://supertech.csail.mit.edu/cilk/>

Charles E. Leiserson

A Sahu

Dept of CSE, IIT Guwahati

# Cilk

- Developed by **Leiserson at CSAIL, MIT**
  - **Chapter 27, Multithreaded Algorithm, Introduction to Algorithm, Coreman, Leiserson and Rivest**
- Initiated a startup: Cilk Plus
  - Added Cilk\_for Keyword, Cilk Reduction features
  - Acquired by Intel, Intel uses Cilk Scheduler
- Addition of 6 keywords to standard C
  - Easy to install in linux system
  - With gcc and pthread

# Cilk

- In 2008, ACM SIGPLAN awarded **Best influential paper of Decade**
  - **The Implementation of the Cilk-5 Multithreaded Language**, PLDI 1998
- PLDI 2008 Best paper Award
  - Reducers and Other Cilk++ Hyperobjects , PLDI 2008



# Cilk : Biggest principle

- Programmer should be responsible for
  - Exposing the parallelism,
  - Identifying elements that can safely be executed in parallel
- Work of run-time environment (scheduler) to
  - Decide during execution how to actually divide the work between processors
- Work Stealing Scheduler
  - Proved to be good scheduler
  - Now also in GCC, Intel CC, **Intel acquire Cilk++**

# Fibonacci

```
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*

*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

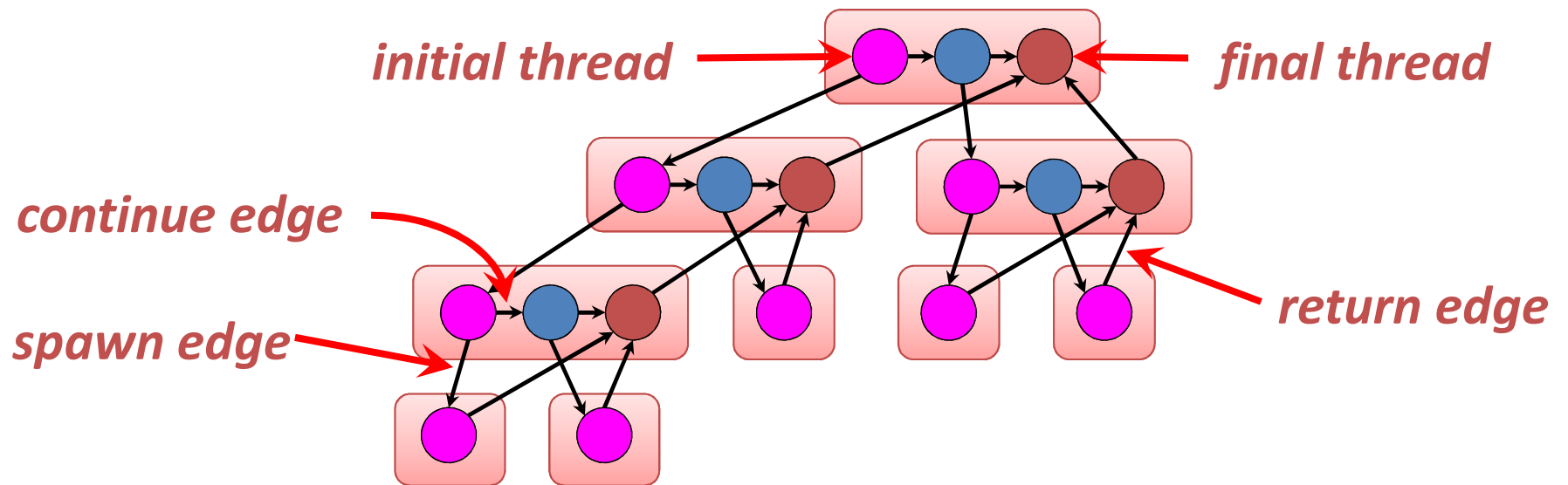
```
cilk int fib (int n) {  
    if (n<2) return (n);  
    else {  
        int x,y;  
        x = spawn fib(n-1);  
        y = spawn fib(n-2);  
        sync;  
        return (x+y);  
    }  
}
```

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.

# Multithreaded Computation

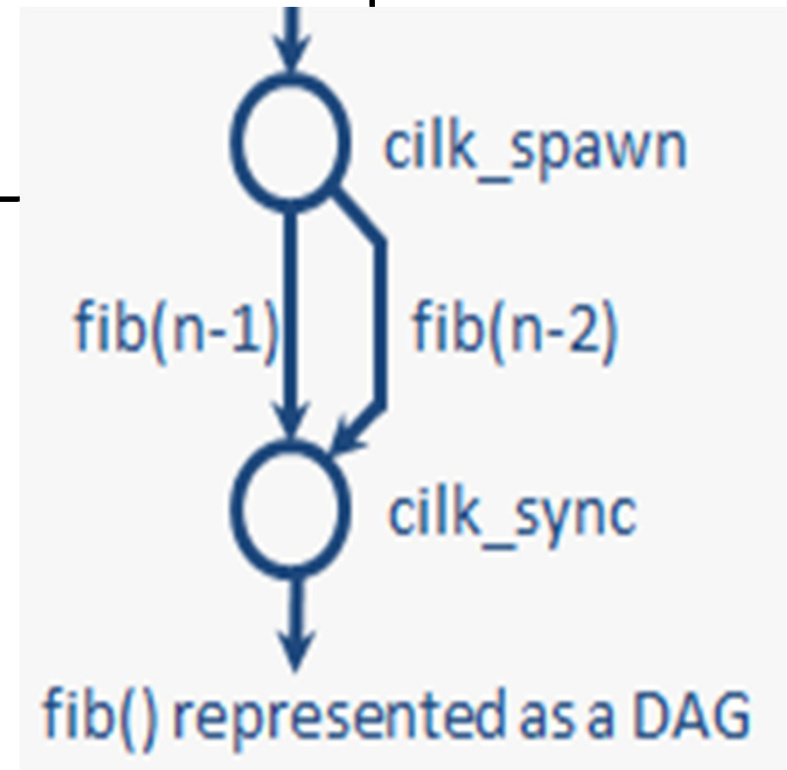


- The dag  $G = (V, E)$  represents a parallel instruction stream.
- Each vertex  $v \in V$  represents a *(Cilk) thread*: a maximal sequence of instructions not containing parallel control (**spawn**, **sync**, **return**).
- Every edge  $e \in E$  is either a *spawn* edge, a *return* edge, or a *continue* edge.

## Fib: Cilk++ Version

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

Not available in  
Cilk



# For loop in Cilk

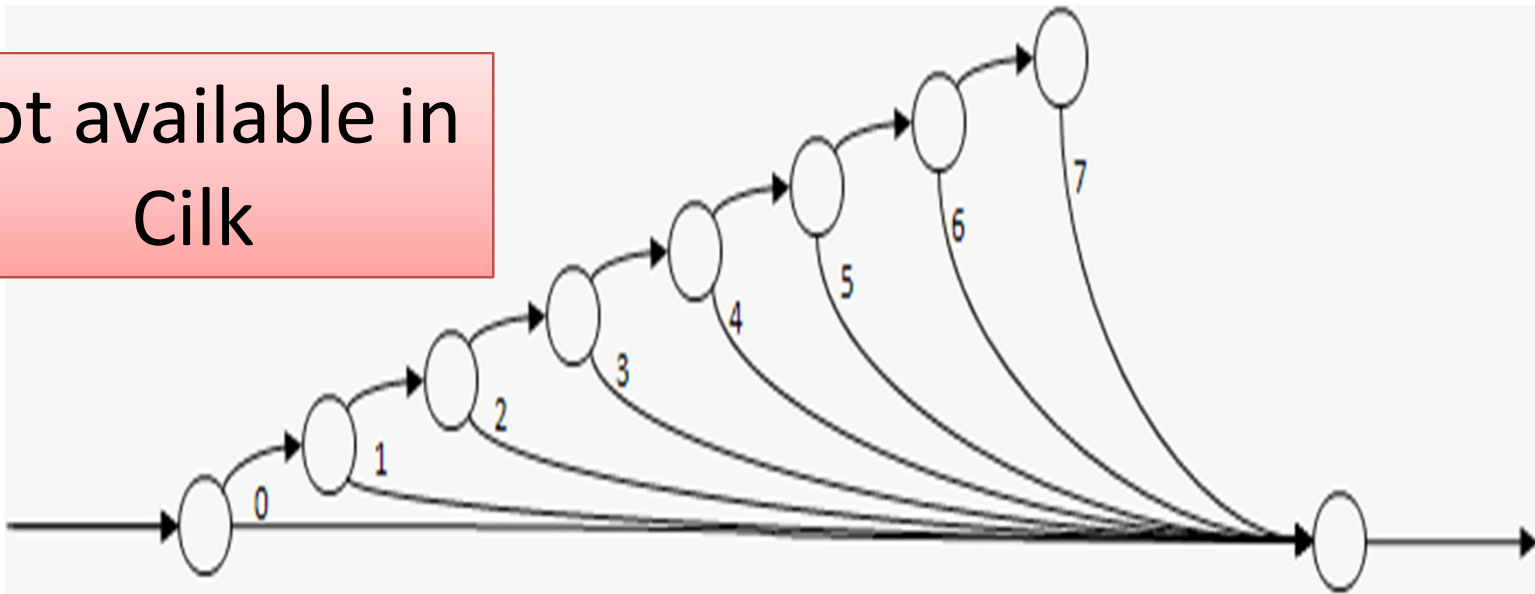
```
for (int i = 0; i < 8; ++i)  
    do_work(i);
```

Serial

```
for (int i = 0; i < 8; ++i)  
    cilk_spawn do_work(i);  
cilk_sync;
```

Parallel

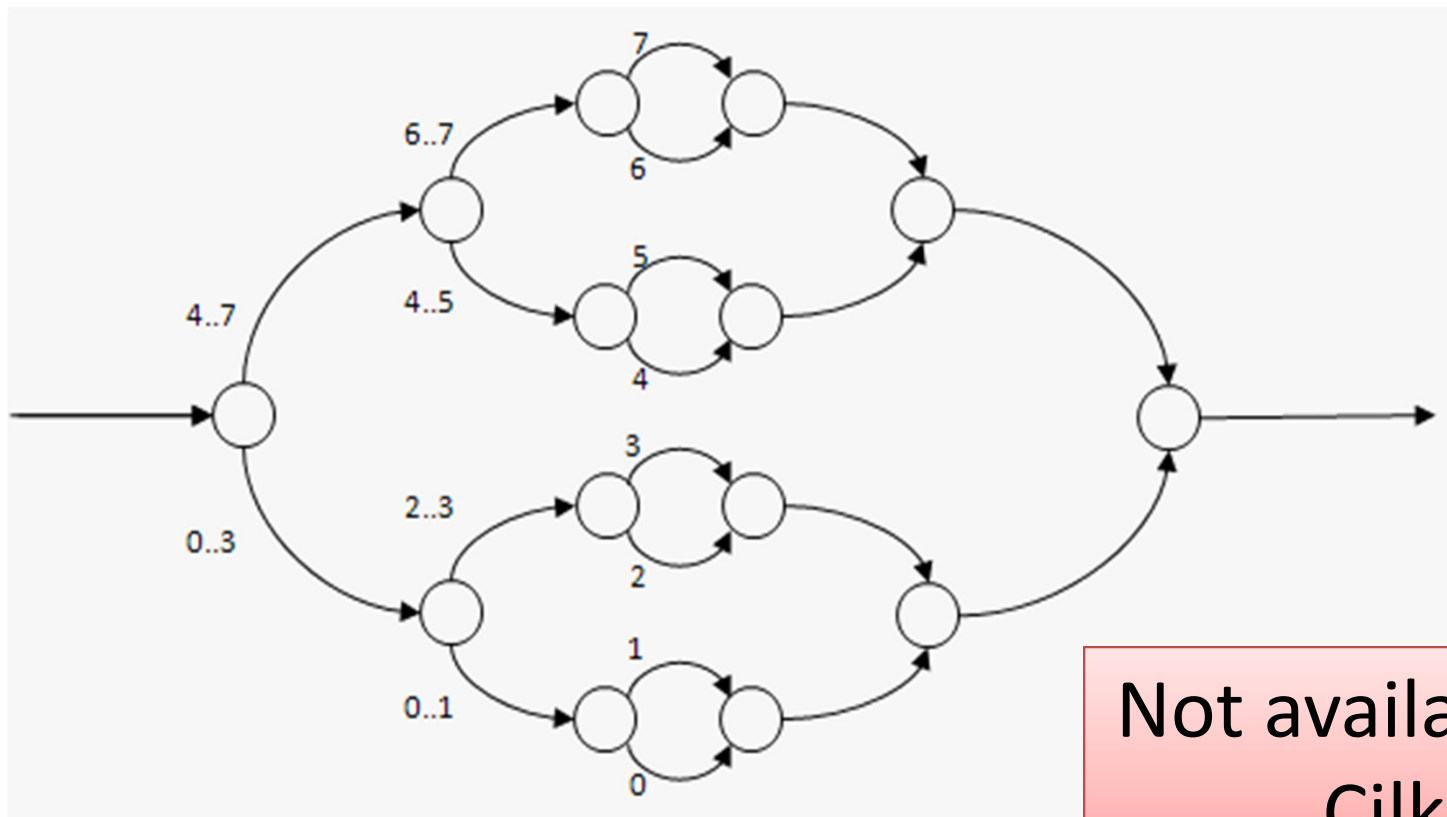
Not available in  
Cilk



# Loop\_for in Cilk++

```
cilk_for (int i=0; i<8; ++i) {  
    do_work(i);  
} // No sync required; auto sync
```

Parallel



Not available in  
Cilk

# Cilk Run Time Scheduler

- Distributed load balancing
  - Receiver initiated
- Work stealing : Free processor steal a task of busy processor
- When ever a process spawns a new process,
  - This processor starts executing the spawned one
  - Parent goes to waiting/suspend mode
  - Parent can be transferred to other processor