Parallel Algorithms (vs. serial algorithms)

Random-access machine model used for serial algorithms
Many models for parallel algorithms, no general agreement.

Model used: Dynamic Multithreading

- appropriate for multicore machines, shared memory programming
- not appropriate for distributed memory programs

Ex:

```
    Fib (n)
A      if n < 2
          then return n
       x ← spawn Fib (n-1)
B      y ← spawn Fib (n-2)
empty  sync
C      return (x+y)
```

spawn, sync, or return terminate current thread threads

A: - includes n-1 computation
- upto spawn

B: - includes n-2 computation
- upto spawn

empty: ignored now

C: - includes x+y computation
- upto return, after sync

spawn: - subroutine can execute at same time as parent
sync: - wait until all children are done

Description of logical parallelism, not actual (does not describe # processors)

A scheduler determines how to map dynamically unfolding execution onto processors.

Serial instruction stream: when in a loop, chain of subsequent instructions.

Logical serial instruction stream is actually not executed sequentially by a processor → instruction-level parallelism not a focus here. The focus is on logical parallelism.

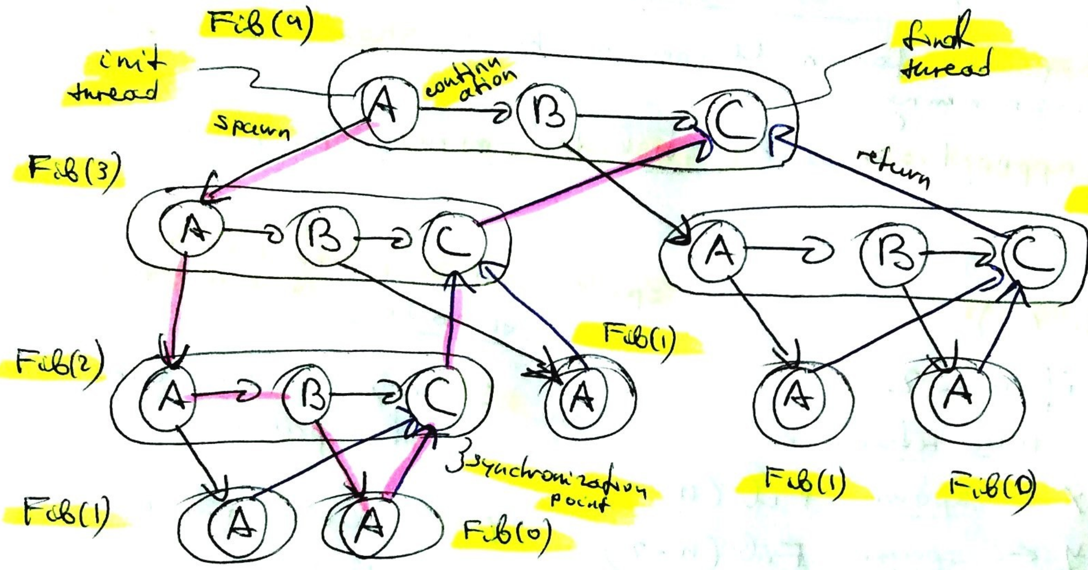
Parallel instruction stream: DAG

Multithreaded computation

Parallel instruction stream = DAG

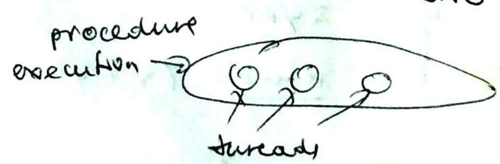
vertices are threads : maximal sequence of instructions not containing parallel control (spawn, sync, return)

edges : spawn, return, continuation



my comment
one root
=> connected DAG

Performance measures



T_P = running time on P processors

T_1 = **work** = serial time (just like getting rid of spawn, sync)

my comment
longest path across topologically sorted DAG
⇓
other threads can be parallel

T_∞ = critical path length = longest path in DAG

Ex Fib(4) [redacted] $T_1 = 17$ (assume each thread is unit time)
\# of threads

$T_\infty = 8$ (longest path in DAG threads that must be sequentially executed)

$\underbrace{\qquad\qquad}$
if unit-time threads

6.046

lecture 22

this model does not take communication into account

Lower bounds on T_p

$T_p \geq \frac{T_1}{p}$

- p proc can do $\leq p$ work in 1 step
- if [redacted],
  [redacted] $T_p < \frac{T_1}{p}$,
  Processor can do $> p$ work in 1 step.

my comment

[redacted]

Suppose $T_p < \frac{T_1}{p}$, then $\exists$ a thread that is not executed. contradiction to T_p def.

$T_p \geq T_\infty$ – p processors can't do more work than ∞ processors

Speedup

T_1/T_p = speedup of p processors

$T_1/T_p = \Theta(p) \Rightarrow$ linear speedup each processor contributes within a constant factor its measure of full support

$T_1/T_p = p \Rightarrow$ perfect linear speedup

$T_1/T_p > p \Rightarrow$ superlinear speedup
NOT possible in this model $\left(T_p < \frac{T_1}{p}\right.$ contr.$\left.\right)$
In other models possible (e.g. caching effects)

Max possible speedup, given T_1, T_∞, is T_1/T_∞ = parallelism
= average amount of work that can be done in parallel along each step of critical path.
= $\overline{P}$

adding more processors does not improve speedup

[redacted]

[redacted]

Scheduling

Map computation to P processors
Done by runtime system (scheduler algorithm)
typically language runtime system

On-line schedulers are complex (randomized schedulers with guarantees!!!)
Illustrate ideas using off-line scheduler.

Greedy scheduler (P processors)

in DAG: cannot execute a node until nodes preceding it are executed

- Do as much as possible on every step.
  - ⇒ do not guess if something is worth delaying
- **Complete step**: $\geq P$ threads ready to run.
  execute any P threads. May be not optimal. There maybe a particular thread, if executed now, enables more parallelism later.
- **Incomplete step**: $< P$ threads ready to run.
  Execute all of them.

!! Scheduling optimally a DAG on P processors is NP-complete.

Theorem (Graham, Brent):
A greedy scheduler executes any computation with work T_1 and critical path length T_∞ in time

$$\boxed{T_P \leq T_1/P + T_\infty} \leq 2\,OPT$$

on a computer with P processors.

2-competitive.

$T_P \geq \frac{T_1}{P}$, $T_P \geq T_\infty$ ⇒ $2\,OPT$

$OPT \geq \max\left(\frac{T_1}{P}, T_\infty\right)$

my comment

$\frac{T_1}{P} \geq T_\infty \Rightarrow \frac{T_1}{P} + T_\infty \leq 2\frac{T_1}{P} \leq 2\,OPT$

6.046
Lecture 22

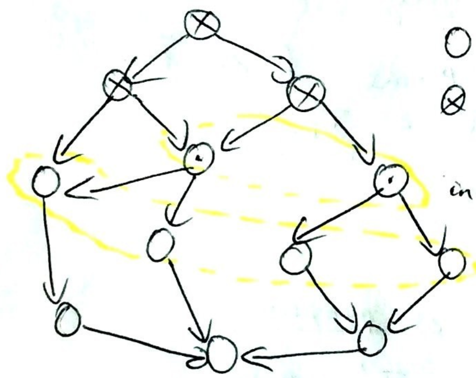
my comment
min # available

Proof: # complete steps $\le \frac{T_1}{P}$ ← assume P threads each step then at most $\frac{T_1}{P}$ steps
since otherwise more than T_1 work would be done.

Consider an incomplete step, and let G' be subgraph of G that remains to be executed.

G:

$P = 3$



O : G'
⊗ : already executed
WLOG each is a unit-time thread

incomplete step

complete step

Threads with in-degree 0 in G' are ready to be executed
The critical path length of G' is reduced by 1

$\Rightarrow$ # incomplete steps $\le T_\infty$

my comment
critical path must include one of O nodes and each of them cannot be reached if not included as start node in the critical path of G' (a DAG) may not be connected but each "root" is included because the step is incomplete

$$\Rightarrow \underbrace{T_P \le \frac{T_1}{P} + T_\infty}$$

foundational theorem of Scheduling!

Corollary: $T_1/T_P = \Theta(P)$ ← linear speedup when $P = O(\bar{P})$ ← $\bar{P} = \frac{T_1}{T_\infty}$
with greedy scheduler

bound of P growth as T_1 and T_∞ grow

$$\bar{P} = T_1/T_\infty \Rightarrow P = O(T_1/T_\infty) \Rightarrow T_\infty = O(T_1/P)$$

Thus $T_P \le \frac{T_1}{P} + O\left(\frac{T_1}{P}\right) = O\left(\frac{T_1}{P}\right)$ ✓

when running on fewer processor than $\bar{P}$, can get speedup.

Cilk

Randomized online scheduler

$E[T_P] = T_1/P + O(T_\infty)$ provably

$T_P \approx T_1/P + T_\infty$ empirically

Near-perfect linear speedup if $P \ll \bar{P}$

i.e. $T_\infty \ll T_1/P$

Chessprograms vs. DeepBlue

Orig. program	Opt. program
$T_{32} = 65$ sec	$T'_{32} = 40$ sec
	Reject.
$T_1 = 2048$	$T_1' = 1024$
$T_\infty = 1$	$T_1^\infty = 8$

$T_{32} = T_1/32 + T_\infty = 65$ $\quad$ $T'_{32} = T'_1/32 + T'_\infty = 40$

Extrapolate on a larger machine

$T_{512} = T_1/512 + T_\infty = 5$ $\quad$ $T'_{512} = T'_1/512 + T'_\infty = 10$