Datenstrukturen und Algorithmen

Exercise 12

FS 2020

Program of today

1. Feedback of last exercise
2. Parallel Programming
3. C++ Threads
4. In-Class Exercise: Image Segmentation

1. Feedback of last exercise

2. Parallel Programming

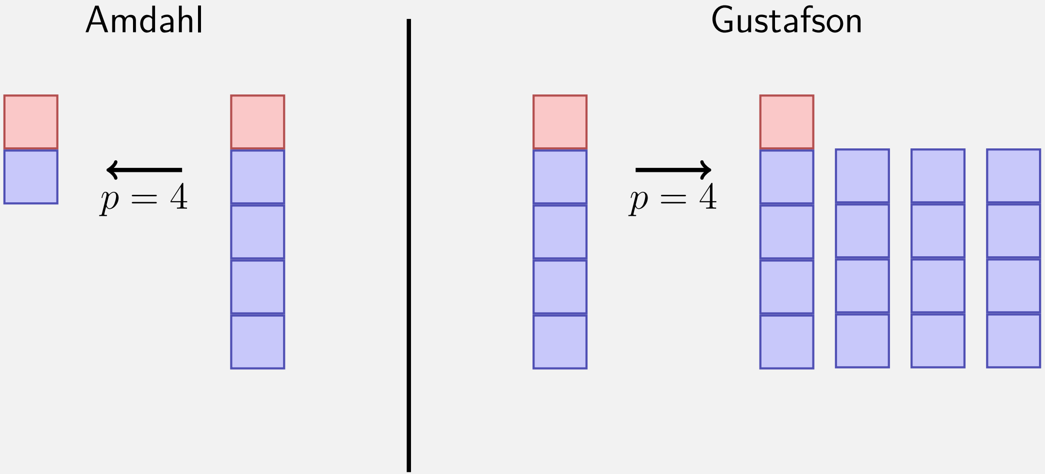
Parallel Performance

Given

- fixed amount of computing work W (number computing steps)
- Sequential execution time T_1
- Parallel execution time on p CPUs

	runtime	speedup	efficiency
perfection (linear)	$T_p = T_1/p$	$S_p = p$	$E_p = 1$
loss (sublinear)	$T_p > T_1/p$	$S_p < p$	$E_p < 1$
sorcery (superlinear)	$T_p < T_1/p$	$S_p > p$	$E_p > 1$

Amdahl vs. Gustafson



Amdahl vs. Gustafson, or why do we care?

Amdahl	**Gustafson**
pessimist	optimist
strong scaling	weak scaling

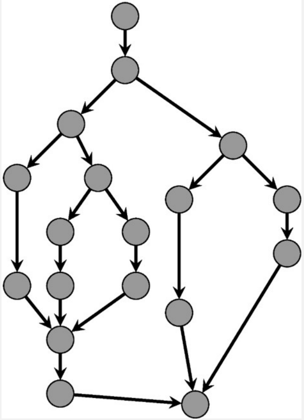
Amdahl vs. Gustafson, or why do we care?

Amdahl	**Gustafson**
pessimist	optimist
strong scaling	weak scaling

$\Rightarrow$ need to develop methods with small sequential protion as possible.

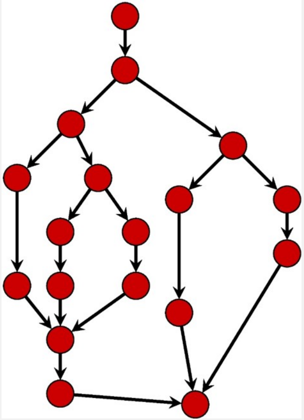
Task Parallelism: Performance Model

- p processors
- Dynamic scheduling
- T_p: Execution time on p processors



Performance Model

- T_p: Execution time on p processors
- T_1: *work*: time for executing total work on one processor
- T_1/T_p: Speedup

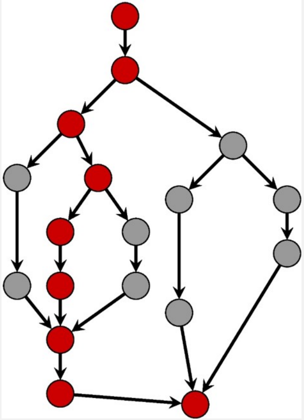


Performance Model

- T_∞: *span*: critical path, execution time on ∞ processors. Longest path from root to sink.
- T_1/T_∞: *Parallelism:* wider is better
- Lower bounds:

$T_p \geq T_1/p$ Work law

$T_p \geq T_\infty$ Span law



Greedy Scheduler

Greedy scheduler: at each time it schedules as many as availbale tasks.

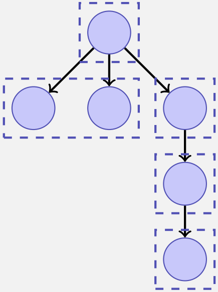
Theorem

On an ideal parallel computer with p processors, a greedy scheduler executes a multi-threaded computation with work T_1 and span T_∞ in time

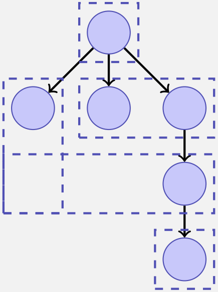
$$T_p \leq T_1/p + T_\infty$$

Beispiel

Assume $p = 2$.



$T_p = 5$



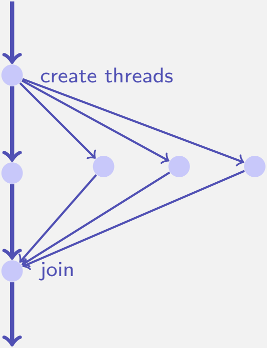
$T_p = 4$

3. C++ Threads

C++11 Threads

```
void hello(int id){
  std::cout << "hello from " << id << "\n";
}

int main(){
  std::vector<std::thread> tv(3);
  int id = 0;
  for (auto & t:tv)
    t = std::thread(hello, ++id);
  std::cout << "hello from main \n";
  for (auto & t:tv)
        t.join();
  return 0;
}
```



Nondeterministic Execution!

One execution:

```
hello from main
hello from 2
hello from 1
hello from 0
```

Other execution:

```
hello from 1
hello from main
hello from 0
hello from 2
```

Other execution:

```
hello from main
hello from 0
hello from hello from 1
2
```

Technical Details I

- With allocating a thread, reference parameters are copied, except explicitly std::ref is provided at the construction.

Technical Details I

- With allocating a thread, reference parameters are copied, except explicitly std::ref is provided at the construction.

```
void calc( std::vector<int>& very_long_vector ){
  // doing funky stuff with very_long_vector
}
int main(){
  std::vector<int> v( 1000000000 );
  std::thread t1( calc, v );         // bad idea, v is copied
  // here v is unchanged
  std::thread t2( calc, std::ref(v) ); // good idea, v is not copied
  // here v is modified
  std::thread t2( [&v]{calc(v)}; } ); // also good idea
  // here v is modified
  // ...
```

Technical Details II

- Threads cannot be copied.

Technical Details II

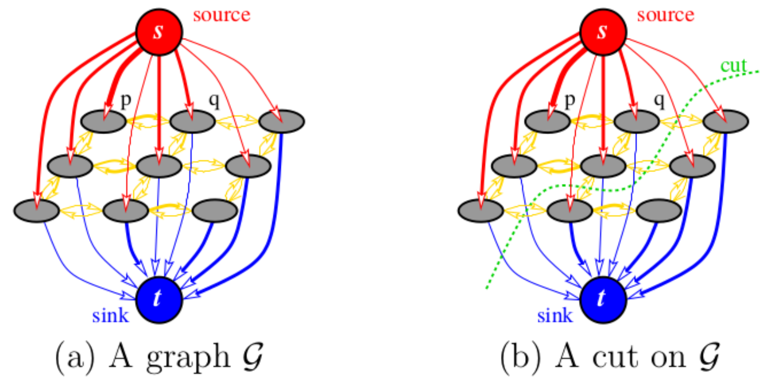
- Threads cannot be copied.

```
{
  std::thread t1(hello);
  std::thread t2;
  t2 = t1; // compiler error
  t1.join();
}
{
  std::thread t1(hello);
  std::thread t2;
  t2 = std::move(t1); // ok
  t2.join();
}
```

4. In-Class Exercise: Image Segmentation

Max Flow / Edmonds-Karp / Push-Relabel

Idea: Max-Flow/Min-Cut



(a) A graph $\mathcal{G}$ (b) A cut on $\mathcal{G}$

Source: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, Y.Boykov and V.Kolmogorov, IEEE Transactions on PAMI, Vol. 26, No. 9, pp. 1124-1137, Sept. 2004

Challenges

- Capacities source-pixel/pixel-sink between neighbouring pixels?
  - Similarity to foreground/background color
  - Similarity of color values

  ⇒ Heuristics / experience (=literature)
- Edmonds-Karp algorithm too slow
  ⇒ Push-Relabel Algorithm

Implementation Push-Relabel?

Input: Flow graph $G = (V, E, c)$, with source s and sink t $n := |v|$

```
$h(s) \leftarrow n$
foreach $v \neq s$ do  $h(v) \leftarrow 0$
foreach $(u, v) \in E$ do  $f(u, v) \leftarrow 0$
foreach $(s, v) \in E$ do  $f(s, v) \leftarrow c(s, v)$

while $\exists u \in V \setminus \{s, t\} : \alpha_f(u) > 0$ do
    choose $u$ with $\alpha_f(u) > 0$ and maximal $h(u)$ ← in $\mathcal{O}(1)$?
    if $\exists v \in V : c_f(u, v) > 0 \wedge h(v) = h(u) - 1$ then
        push$(u, v)$ ←Efficient way to find edges?                // push
    else
        $h(u) \leftarrow h(u) + 1$                                 // relabel
```

Possibilities

Management of the nodes:

- Maximal height $2n-1 \Rightarrow$ Node lists by height
  Algorithm Running Time $\mathcal{O}(n^2\sqrt{m})$
- Weaken the order: use FIFO list or relabel-to-front heuristics for nodes with excess.
  Algorithm Running Time $\mathcal{O}(n^3)$

Management of the edges:

- Memorize the most recently used edge (=iterator) per node.
- Unnecessary for image segmentation, because only few edges per node

Questions?