Introduction to Scheduling

Arnaud Legrand, CNRS, University of Grenoble

LIG laboratory, arnaud.legrand@imag.fr

November 16, 2015

Outline

Outline

Outline

1. Modeling Applications, General Notions
   - Introducing Fundamental Notions Through the Matrix Product Example
   - Adaptive Parallel Programs
   - Task Graphs and Parallel Tasks From Outer Space
2. Defining a Scheduling Problem
   - Rules of the Game
   - Criteria: How Do You Win the Game?
   - Analysis Method
   - Graham Notation
3. Batch Scheduling
   - Principles
   - Theoretical results
   - Basic idea: FCFS + Backfilling
   - EASY
   - How Good is the Schedule?
4. Gang Scheduling as an Alternative
   - Principles

{ *To compute* $C \leftarrow C + A \times B$ };
for $i = 1$ **to** n **do**
  for $j = 1$ **to** n **do**
    for $k = 1$ **to** n **do**
      $C_{i,j} \leftarrow C_{i,j} + A_{i,k} \times B_{k,j}$

$B_{1,1}$	$B_{1,2}$
$B_{2,1}$	$B_{2,2}$

$A_{1,1}$	$A_{1,2}$
$A_{2,1}$	$A_{2,2}$

$C_{1,1}$	$C_{1,2}$
$C_{2,1}$	$C_{2,2}$

Matrix Product: Sequential Version

2 $C_{1,1} \leftarrow C_{1,1} + A_{1,1} \times B_{1,1}$;

4 $C_{1,1} \leftarrow C_{1,1} + A_{1,2} \times B_{2,1}$;

6 $C_{1,2} \leftarrow C_{1,2} + A_{1,1} \times B_{1,2}$;

8 $C_{1,2} \leftarrow C_{1,2} + A_{1,2} \times B_{2,2}$;

9 ...

CPU	2	4	6	8

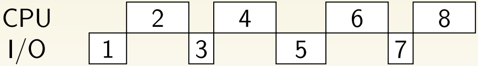
$B_{1,1}$	$B_{1,2}$
$B_{2,1}$	$B_{2,2}$

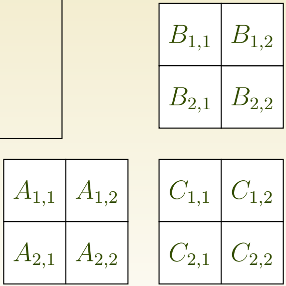
$A_{1,1}$	$A_{1,2}$
$A_{2,1}$	$A_{2,2}$

$C_{1,1}$	$C_{1,2}$
$C_{2,1}$	$C_{2,2}$

Matrix Product: Sequential Version

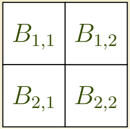
1 Load $C_{1,1}$, $A_{1,1}$, $B_{1,1}$;
2 $C_{1,1} \leftarrow C_{1,1} + A_{1,1} \times B_{1,1}$;
3 Unload $A_{1,1}$, $B_{1,1}$. Load $A_{1,2}$, $B_{2,1}$;
4 $C_{1,1} \leftarrow C_{1,1} + A_{1,2} \times B_{2,1}$;
5 Unload $C_{1,1}$, $A_{1,2}$, $B_{2,1}$. Load $C_{1,2}$, $A_{1,1}$, $B_{1,2}$;
6 $C_{1,2} \leftarrow C_{1,2} + A_{1,1} \times B_{1,2}$;
7 Unload $A_{1,1}$, $B_{1,2}$;
8 $C_{1,2} \leftarrow C_{1,2} + A_{1,2} \times B_{2,2}$;
9 ...





Matrix Product: Sequential Version

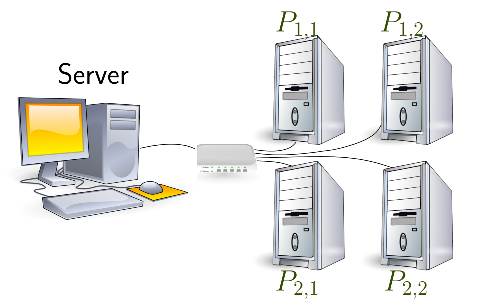
```
Load C_{1,1}, A_{1,1}, B_{1,1};
C_{1,1} ← C_{1,1} + A_{1,1} × B_{1,1};
Unload A_{1,1}, B_{1,1}. Load A_{1,2}, B_{2,1};
C_{1,1} ← C_{1,1} + A_{1,2} × B_{2,1};
Unload C_{1,1}, A_{1,2}, B_{2,1}. Load C_{1,2}, A_{1,1}, B_{1,2};
C_{1,2} ← C_{1,2} + A_{1,1} × B_{1,2};
Unload A_{1,1}, B_{1,2};
C_{1,2} ← C_{1,2} + A_{1,2} × B_{2,2};
...
```



Sequential Programs

Sequential programs are generally a succession of CPU burst and I/O burst.

Matrix Product: Parallel Version (1/2)
Setting



- A, B, and C are initially located on the server.
- We will distribute A, B, and C on $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$.
- We will make use of all four processors to compute $C \leftarrow C + A \times B$.
- Such a parallel program could be written using for example MPI. We want a SPMD algorithm.

Matrix Product: Parallel Version (2/2)

Algorithm

{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };

Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;

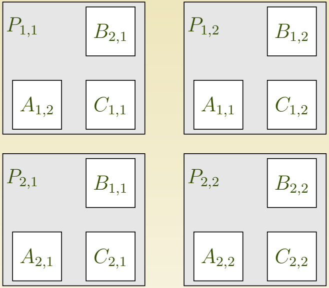
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;

Exchange A_{local} with horizontal neighbor;

Exchange B_{local} with vertical neighbor;

$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;

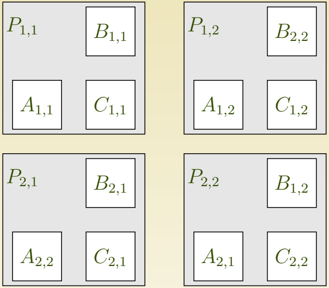
Unload $C_{i,j}$ to the server;



Matrix Product: Parallel Version (2/2)

Algorithm

{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };

Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;

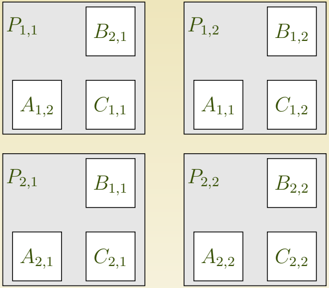
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;

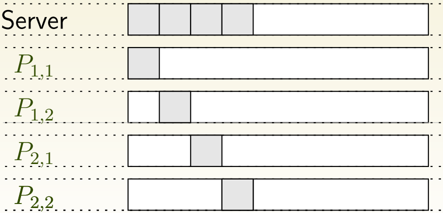
Exchange A_{local} with horizontal neighbor;

Exchange B_{local} with vertical neighbor;

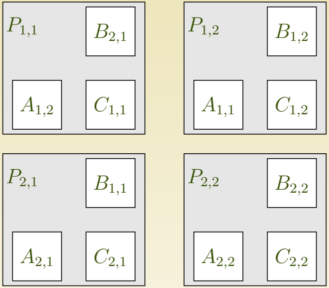
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;

Unload $C_{i,j}$ to the server;





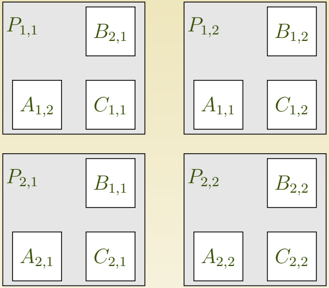
{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



Matrix Product: Parallel Version (2/2)

Algorithm

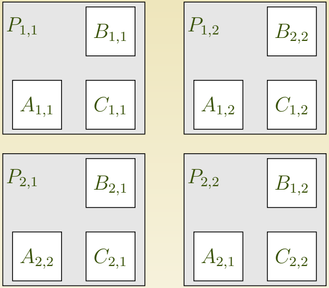
{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



Matrix Product: Parallel Version (2/2)

Algorithm

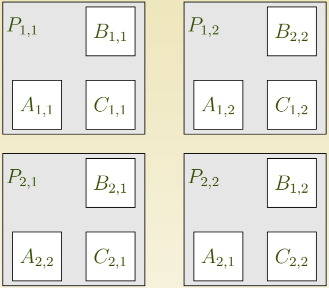
{ *$P_{i,j}$ is responsible for computing $C_{i,j}$.* };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



Matrix Product: Parallel Version (2/2)

Algorithm

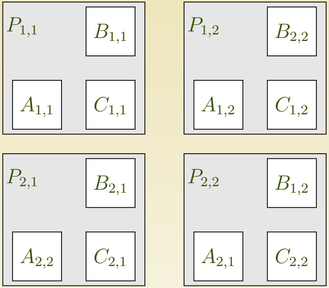
{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



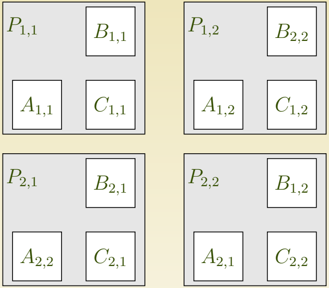
Matrix Product: Parallel Version (2/2)

Algorithm

{ $P_{i,j}$ *is responsible for computing* $C_{i,j}$. };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;



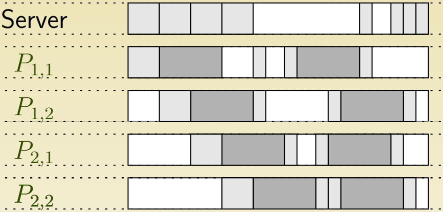
{ *$P_{i,j}$ is responsible for computing $C_{i,j}$.* };
Load $C_{i,j}$, $A_{i,(i+j)\%2}$, $B_{(i+j)\%2,j}$ from the server;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Exchange A_{local} with horizontal neighbor;
Exchange B_{local} with vertical neighbor;
$C_{local} \leftarrow C_{local} + A_{local} \times B_{local}$;
Unload $C_{i,j}$ to the server;

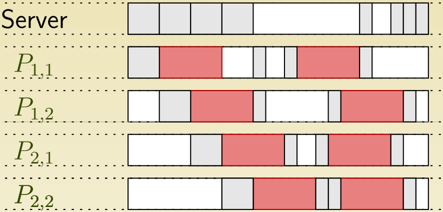


Parallel Programs

Parallel programs are generally a succession of CPU burst and communication burst. The synchronization pattern generally incurs idle time. This is the parallelization overhead.

Work, Cost, Speed-up and Efficiency

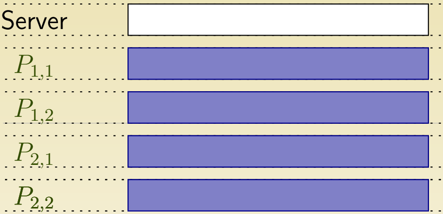




Definition: Work.

The work is the amount of computation performed (the surface of the pink area).
In the previous parallel Matrix Multiplication example, the work is the same as in the sequential Matrix Multiplication example.
However, parallel algorithms generally do not do the same operations as the sequential ones. They often have to do more. Therefore, the work $W(p)$ generally depends on the number of processors that are alloted!

Work, Cost, Speed-up and Efficiency

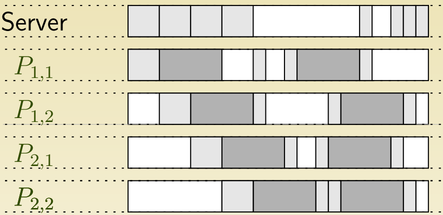


Definition: Cost.

$$C(p) = p \times \mathsf{TotalTime}(p).$$

It is the total surface.
The cost accounts for the idle time of the processing units.

Work, Cost, Speed-up and Efficiency



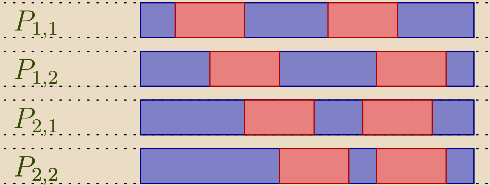
Definition: Speed-up and Efficiency.

- Speed-up: $s(p) = \dfrac{\text{SequentialTime}}{\text{TotalTime}(p)}$.
- Efficiency: $e(p) = \dfrac{s(p)}{p} = \dfrac{\text{SequentialTime}}{p \times \text{TotalTime}(p)}$.

Side Note on Speed-up and Efficiency

Speed-up

We have SequentialTime $\leqslant C(p) \leqslant p \times$ TotalTime(p).



Side Note on Speed-up and Efficiency

Speed-up

We have $\text{SequentialTime} \leqslant C(p) \leqslant p \times \text{TotalTime}(p)$.

Hence, $s(p) = \dfrac{\text{SequentialTime}}{\text{TotalTime}(p)} \leqslant p$ and $e(p) = \frac{\text{SequentialTime}}{p\text{TotalTime}(p)} \leqslant 1$.
The speed-up is bounded by the number of processors and the efficiency is thus in $[0, 1]$.

Side Note on Speed-up and Efficiency

Speed-up

We have SequentialTime $\leqslant C(p) \leqslant p \times$ TotalTime(p).

Hence, $s(p) = \dfrac{\text{SequentialTime}}{\text{TotalTime}(p)} \leqslant p$ and $e(p) = \frac{\text{SequentialTime}}{p\text{TotalTime}(p)} \leqslant 1$.

The speed-up is bounded by the number of processors and the efficiency is thus in $[0, 1]$.

Still, Supra-linear speed-up may happen!.

Side Note on Speed-up and Efficiency

Speed-up

We have SequentialTime $\leqslant C(p) \leqslant p \times$ TotalTime(p).

Hence, $s(p) = \dfrac{\text{SequentialTime}}{\text{TotalTime}(p)} \leqslant p$ and $e(p) = \frac{\text{SequentialTime}}{p\text{TotalTime}(p)} \leqslant 1$.

The speed-up is bounded by the number of processors and the efficiency is thus in $[0, 1]$.

Still, Supra-linear speed-up may happen!.

We did not take I/O into account. With p processor, we have p times more available memory. Swapping sometimes kills the sequential algorithm.

Side Note on Speed-up and Efficiency

Speed-up

We have SequentialTime $\leqslant C(p) \leqslant p \times$ TotalTime(p).

Hence, $s(p) = \dfrac{\text{SequentialTime}}{\text{TotalTime}(p)} \leqslant p$ and $e(p) = \frac{\text{SequentialTime}}{p\text{TotalTime}(p)} \leqslant 1$.
The speed-up is bounded by the number of processors and the efficiency is thus in $[0, 1]$.

Still, Supra-linear speed-up may happen!.

We did not take I/O into account. With p processor, we have p times more available memory. Swapping sometimes kills the sequential algorithm.

Efficiency

TotalTime(p) does not necessarily decrease with p due to the parallelization overhead.
Using more processors may hurt and may be particularly inefficient!

```
var A, B, C:  array[0..m−1,0..m−1] of real;
var bufferA, bufferB:  array[0..m−1,0..m−1] of real;
q ←SRQT(Num_Procs());
myrow ←My_Proc_Row();
mycol ←My_Proc_Col();
for k = 0 to q − 1 do
    for i = 0 to m − 1 do                          { Broadcast A along rows }
        BroadcastRow(i, k, A, bufferA, m × m)
    for j = 0 to m − 1 do                          { Broadcast B along columns }
        BroadcastCol(k, j, B, bufferB, m × m)
    { Multiply matrix blocks } if (myrow = k) And (mycol = k) then
        MatrixMultiplyAdd(C, A, B, m)
    else if (myrow = k) then  MatrixMultiplyAdd(C, bufferA, B, m) ;
    else if (mycol = k) then  MatrixMultiplyAdd(C, A, bufferB, m) ;
    else  MatrixMultiplyAdd(C, bufferA, bufferB, m) ;
```

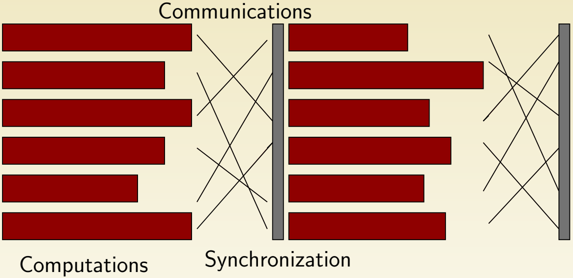
```
var A, B, C: array[0..m−1,0..m−1] of real;
var bufferA, bufferB: array[0..m−1,0..m−1] of real;
q ←SRQT(NUM_PROCS());
```

Two Comments

- Many parallel programs take the number of processors as an input and adapt to it.
- Many parallel programs use collective communication operations and synchronization.

```
        BROADCASTCOL(k, j, B, bufferB, m × m)
    { Multiply matrix blocks } if (myrow = k) And (mycol = k) then
        MATRIXMULTIPLYADD(C, A, B, m)
    else if (myrow = k) then MATRIXMULTIPLYADD(C, bufferA, B, m) ;
    else if (mycol = k) then MATRIXMULTIPLYADD(C, A, bufferB, m) ;
    else MATRIXMULTIPLYADD(C, bufferA, bufferB, m) ;
```

Bulk Synchronous Parallel is a programming paradigm whose principle is a series of independent steps of computations and communication/synchronization.



The cost of a superstep is determined as the sum of three terms:

$$T = \max_i w(i) + \max h(i) g + l$$

Scheduling under BSP is about finding a tradeoff between load-balancing and number of communication/synchronizations.

Outline

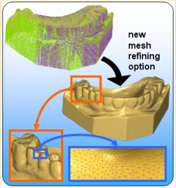
Moldable Parallel Programs

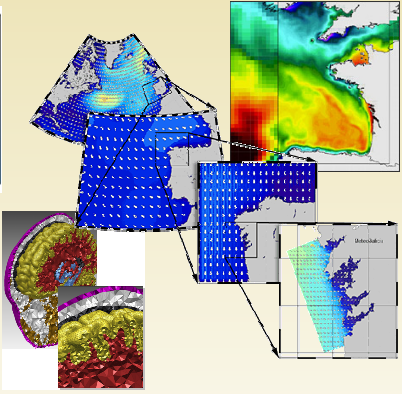
Remember the previous “Block Version of the Outer-Product Algorithm”.

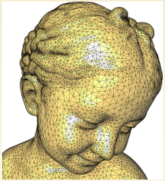
```
q ←SRQT(NUM_PROCS());
myrow ←MY_PROC_ROW();
mycol ←MY_PROC_COL();
for k = 0 to q − 1 do
    ...
```

This q is not hard-coded. The algorithm adapts to the number of available processors at the beginning of the execution.
It uses this number to distribute the data and organize the communications.
Such programs are called moldable.

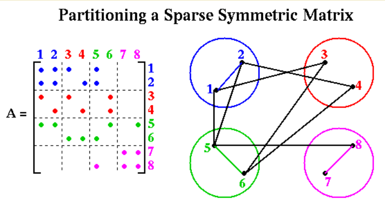
Code Coupling and Adaptive Mesh-Refining

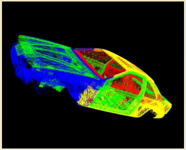


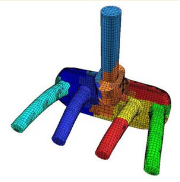


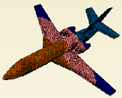


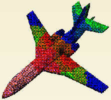
Mesh Partitioning

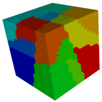




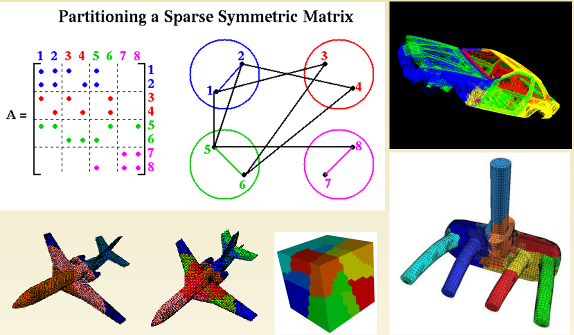






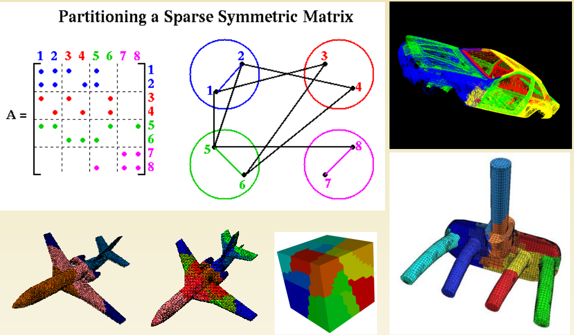


Code Coupling and Adaptive Mesh-Refining



When using adaptive mesh-refining, load imbalance occurs. Coupling code makes it worse.

Code Coupling and Adaptive Mesh-Refining



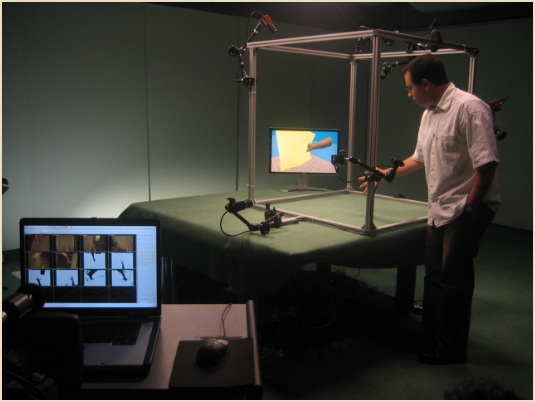
When using adaptive mesh-refining, load imbalance occurs. Coupling code makes it worse. Recomputing a good partition and redistributing the data is not necessarily a good option. However adding computing resources on the fly is often very efficient.

$\leadsto$ The resource requirements vary over the time.

This kind of program is called dynamic.

Adaptive Programs

FlowVR: Adaptive Interactive Rendering



These programs can adapt to the resource they are alloted over the time.

This kind of program is called malleable.

Adaptive Programs

KAAPI: Adaptive, Asynchronous Parallel and Interactive Computing

KAAPI is based on work-stealing algorithms and contains non-blocking and scalable algorithms.
KAAPI/Taktuk won the 4th and 5th International Challenge GRIDS@WORK (2007, 2008).

2007 N-queens

2008 Super Quant Monte-Carlo, pricing application.

- 3609 cores used between France and Japan during one hour.
- The KAAPI/Taktuk team was able to price 988 actions on the 1000 of the challenge and was scored 8760/18000.
- The second team was able to price 177 actions using 4329 and was scored 1459/18000.

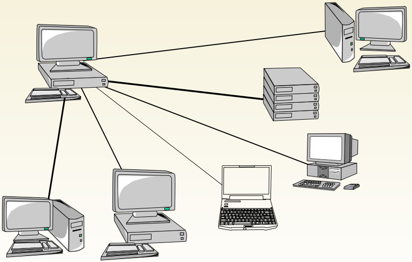
These programs can adapt to the resource they are alloted over the time.
This kind of program is called malleable.

Divisible Load Scheduling

Parallelizing generally has a price. There is a computation overhead and a communication/synchronization overhead.
Some applications however have a very low computation overhead and can be very easily divided.

- Pattern Searching
- Database Computation
- Video encoding
- Image processing

Such applications are very well suited to master-slave computing.



Computation times and communication times are linear with the fraction of alloted load.

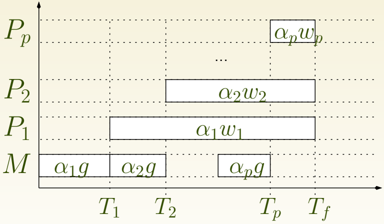
This kind of program is called divisible.

Divisible Load Scheduling

Parallelizing generally has a price. There is a computation overhead and a communication/synchronization overhead.
Some applications however have a very low computation overhead and can be very easily divided.

- Pattern Searching
- Database Computation
- Video encoding
- Image processing

Such applications are very well suited to master-slave computing.



Computation times and communication times are linear with the fraction of alloted load.

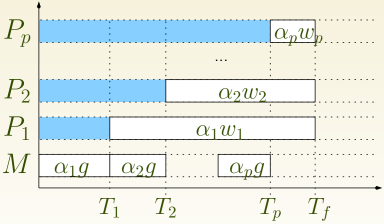
This kind of program is called divisible.

Parallelizing generally has a price. There is a computation overhead and a communication/synchronization overhead.

Some applications however have a very low computation overhead and can be very easily divided.

- Pattern Searching
- Database Computation
- Video encoding
- Image processing

Such applications are very well suited to master-slave computing.



Computation times and communication times are linear with the fraction of alloted load.

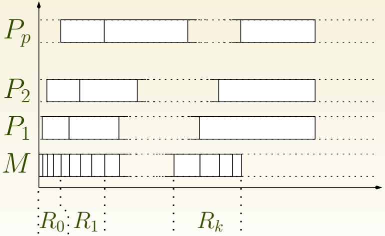
This kind of program is called divisible.

Divisible Load Scheduling

Parallelizing generally has a price. There is a computation overhead and a communication/synchronization overhead.
Some applications however have a very low computation overhead and can be very easily divided.

- Pattern Searching
- Database Computation
- Video encoding
- Image processing

Such applications are very well suited to master-slave computing.



Computation times and communication times are linear with the fraction of alloted load.

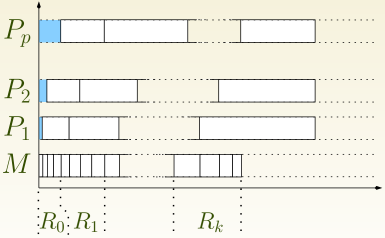
This kind of program is called divisible.

Divisible Load Scheduling

Parallelizing generally has a price. There is a computation overhead and a communication/synchronization overhead.
Some applications however have a very low computation overhead and can be very easily divided.

- Pattern Searching
- Database Computation
- Video encoding
- Image processing

Such applications are very well suited to master-slave computing.



Computation times and communication times are linear with the fraction of alloted load.

This kind of program is called divisible.

Outline

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} \\ \\ \\ t = 8/4 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} \\ \\ \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

$$\begin{cases} \\ \\ 6z + t = -4 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ \quad\;\; y - 3z - t = 5 \\ \qquad\quad 6z + t = -4 \\ \qquad\qquad\; 4t = 8 \end{cases}$$

$$\begin{cases} \\ \\ 6z + 2 = -4 \\ \quad\;\; t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} \\ \\ 6z = -4 - 2 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x+ \; y+ \; z+ \; t = 6 \\ \quad\; y- 3z- \; t = 5 \\ \qquad\quad 6z+ \; t = -4 \\ \qquad\qquad\quad 4t = 8 \end{cases}$$

$$\begin{cases} \\ \\ z = -6/6 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} \\ \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

$$\begin{cases} y - 3z - t = 5 \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x+ \; y+ \; z+ \; t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

$$\begin{cases} \\ y + (3-2) = 5 \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ \quad y - 3z - t = 5 \\ \qquad 6z + t = -4 \\ \qquad\quad 4t = 8 \end{cases}$$

$$\begin{cases} \\ y + 1 \quad = 5 \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\left\{\begin{array}{rrrrr} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{array}\right.$$

$$\left\{\begin{array}{l} \\ y = (5-1)/1 \\ z = -1 \\ t = \ 2 \end{array}\right.$$

Solving a triangular system (step by step)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

$$\begin{cases} \\ y = & 4 \\ z = & -1 \\ t = & 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & y- & 3z- & t = & 5 \\ & & 6z+ & t = & -4 \\ & & & 4t = & 8 \end{cases}$$

$$\begin{cases} x+ & y+ & z+ & t = & 6 \\ & & & y = & 4 \\ & & & z = & -1 \\ & & & t = & 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} x + (4 - 1 + 2) = 6 \\ y = 4 \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

$$\begin{cases} x + 5 \qquad = 6 \\ y = 4 \\ z = -1 \\ t = 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x+\ y+\ \ z+\ \ t = \ \ 6 \\ \quad\ \ y-3z-\ \ t = \ \ 5 \\ \qquad\quad 6z+\ \ t = -4 \\ \qquad\qquad\quad 4t = \ \ 8 \end{cases}$$

$$\begin{cases} x = (6-5)/1 \\ y = \ \ 4 \\ z = -1 \\ t = \ \ 2 \end{cases}$$

Solving a triangular system (step by step)

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases} \qquad \begin{cases} x = 1 \\ y = 4 \\ z = -1 \\ t = 2 \end{cases}$$

The main steps are:

- we start from the bottom and proceed to the top
- we make horizontal sums of products
- we divide the results by a coefficient

(without using np.linalg.solve(A,b), of course! ☺)

$$\begin{aligned} x+\ y+\ z+\ t &= 6\\ y-3z-\ t &= 5\\ 6z+\ t &= -4\\ 4t &= 8 \end{aligned}$$

```
import numpy as np
A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
      [0, 0, 6, 1] , [0, 0, 0, 4]], float)
b = np.array([6, 5, -4, 8], float)

n = len(b)
x = np.zeros(n, float)
for i in reversed(range(0,n)): # from the bottom to the top
  S = 0
  for j in range(i+1,n):
     S = S + A[i][j] * x[j]     # the sum of products
  x[i] = (b[i] - S) / A[i][i]   # the division
print(x)
```

```
[ 1.  4. -1.  2.]
```

As such, this code is intrinsically sequential (because of S!)

Solving a triangular system (minor rewriting)

```
for i in reversed(range(0,n)):          # from the bottom to the top
  S = np.dot(A[i][i+1:n],x[i+1:n])    # the sum of products
  x[i] = (b[i] - S) / A[i][i]          # the division
```

This version is much faster than the previous one because:

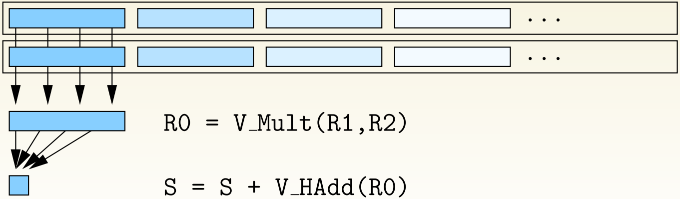
- no interpretation of the inner loop, no need to check bounds, ...

Solving a triangular system (minor rewriting)

```
for i in reversed(range(0,n)):        # from the bottom to the top
  S = np.dot(A[i][i+1:n],x[i+1:n])   # the sum of products
  x[i] = (b[i] - S) / A[i][i]         # the division
```

This version is much faster than the previous one because:

- no interpretation of the inner loop, no need to check bounds, ...
- the sum can be *vectorized* (MMX, SSE, *GPU*,...)
- the sum can be *multithreaded* (threads, multicore)
- the sum can be *distributed* (MPI, cluster)

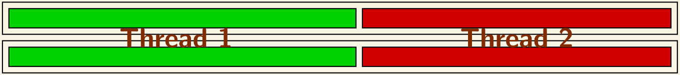


Solving a triangular system (minor rewriting)

```python
for i in reversed(range(0,n)):        # from the bottom to the top
  S = np.dot(A[i][i+1:n],x[i+1:n])   # the sum of products
  x[i] = (b[i] - S) / A[i][i]        # the division
```

This version is much faster than the previous one because:

- no interpretation of the inner loop, no need to check bounds, ...
- the sum can be *vectorized* (MMX, SSE, *GPU*,...)
- the sum can be *multithreaded* (threads, multicore)
- the sum can be *distributed* (MPI, cluster)




```
thread_create()
S1 = prod(A1,B1)        S2 = prod(A2,B2)
thread_join()
S = S1 + S2
```

Solving a triangular system (minor rewriting)

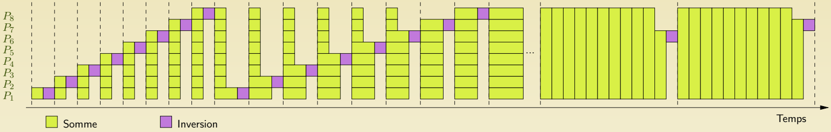
```
for i in reversed(range(0,n)):          # from the bottom to the top
  S = np.dot(A[i][i+1:n],x[i+1:n])      # the sum of products
  x[i] = (b[i] - S) / A[i][i]           # the division
```

This version is much faster than the previous one because:

- no interpretation of the inner loop, no need to check bounds, ...
- the sum can be *vectorized* (MMX, SSE, *GPU*,...)
- the sum can be *multithreaded* (threads, multicore)
- the sum can be *distributed* (MPI, cluster)

Node 1	Node 2	Node...
MPI_Scatter	MPI_Scatter	MPI_S...
S = prod(A,B)	S = prod(A,B)	S = ...
MPI_Gather(Sp)	MPI_Gather(Sp)	MPI_G...
S = sum(Sp)		

Is it satisfying ?



No!

- CPUs are mostly inactive (except in the end)
- Communications = very small and very frequent
- If one process is ever late, it delays all the others

To obtain performance, we should completely reorganize this code:

- change the granularity
- get rid of synchronizations

Some non trivial reorganization

$$\begin{cases} x + y + z + t = 6 \\ y - 3z - t = 5 \\ 6z + t = -4 \\ 4t = 8 \end{cases}$$

Let's propagate new values in all lines (sums) as soon as possible

$$\begin{cases} x + y + z + \mathbf{2} = 6 \\ y - 3z - \mathbf{2} = 5 \\ 6z + \mathbf{2} = -4 \\ 4t = 8 \end{cases}$$

Let's propagate new values in all lines (sums) as soon as possible (we now proceed "vertically")

```python
import numpy as np
A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
     [0, 0, 6, 1] , [0, 0, 0, 4]], float)
b = np.array([6, 5, -4, 8], float)

n = len(b)
x = np.zeros(n, float)
S = np.zeros(n, float)
for j in reversed(range(0,n)):      # from the bottom to the top
  x[j] = (b[j] - S[j]) / A[j][j]    # the division           (D_j)
  for i in range(0,j):              # A true parallel loop
     S[i] = S[i] + A[i][j] * x[j]   # Update                 (U_i,j
```

Analyzing a Simple Code

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
     S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

For a given value $1 \leqslant i \leqslant n$, all tasks $U_{i,*}$ are computations done during the i^{th} iteration of the outer loop.

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
     S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

For a given value $1 \leqslant i \leqslant n$, all tasks $U_{i,*}$ are computations done during the i^{th} iteration of the outer loop.

$<_{seq}$ is the sequential order :

$$D_n <_{seq} U_{1,n} <_{seq} U_{2,n} <_{seq} \dots <_{seq} U_{n,n} <_{seq} D_{n-1} <_{seq} U_{1,n-1} <_{seq} \dots <_{seq} D_1 \ .$$

Independence

However, some independent tasks could be executed in parallel.

- Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
- Two independent tasks may read the value but never write to the same memory location.

Independence

However, some independent tasks could be executed in parallel.

- Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
- Two independent tasks may read the value but never write to the same memory location.

For a given task T, $In(T)$ denotes the set of input variables and $Out(T)$ the set of output variables.

Independence

However, some independent tasks could be executed in parallel.

- Independent tasks are the ones whose execution order can be changed without modifying the result of the program.
- Two independent tasks may read the value but never write to the same memory location.

For a given task T, $In(T)$ denotes the set of input variables and $Out(T)$ the set of output variables.

In the previous example, we have:

$$\begin{cases} In(D_j) = \{b(j), S(j), a(j,j)\} \\ Out(D_j) = \{x(j)\} \text{ and} \end{cases}$$

$$\begin{cases} In(U_{i,j}) = \{a(i,j), x(i), S(i)\} \\ Out(U_{i,j}) = \{S(i)\} \text{ for } i < j. \end{cases}$$

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
     S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Bernstein66].

Bernstein Conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Bernstein66].

We can check that:

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
    S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Bernstein66].

We can check that:

- $Out(D_n) \cap In(U_{n,1}) = \{x(n)\}$ $\leadsto D_n \perp U_{n,1}$.

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
    S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

Bernstein Conditions

Definition.

Two tasks T and T' are not independent ($T \perp T'$) whenever they share a written variable:

$$T \perp T' \Leftrightarrow \begin{cases} & In(T) \cap Out(T') \neq \emptyset \\ \text{or} & Out(T) \cap In(T') \neq \emptyset \\ \text{or} & Out(T) \cap Out(T') \neq \emptyset \end{cases} .$$

Those conditions are known as Bernstein's conditions [Bernstein66].

We can check that:

- $Out(D_n) \cap In(U_{n,1}) = \{x(n)\}$ $\leadsto D_n \perp U_{n,1}$.
- $Out(U_{3,n}) \cap Out(U_{3,n-1}) = \{S(3)\}$ $\leadsto U_{3,n} \perp U_{3,n-1}$.

```
for j in reversed(range(0,n)):
  x[j] = (b[j] - S[j]) / A[j][j]  # (D_j)
  for i in range(0,j):
    S[i] = S[i] + A[i][j] * x[j] # (U_i,j)
```

Precedence

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.

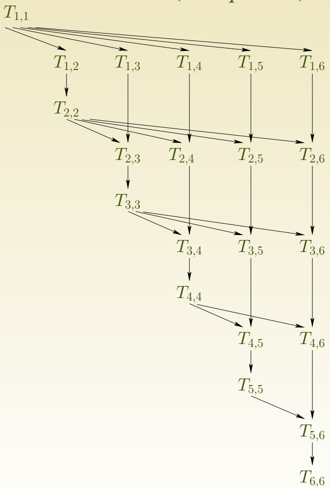
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.

More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ **to** n **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ **to** n **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

A dependency graph G is used.



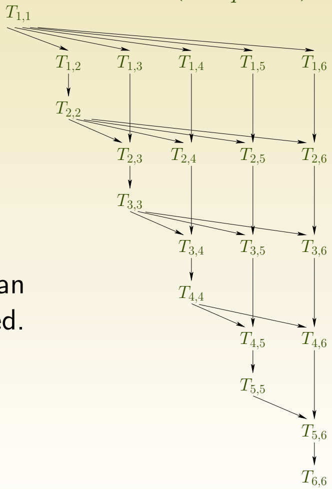
Precedence

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.
More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ **to** n **do**
    Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
    for $j = i+1$ **to** n **do**
        Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$

A dependency graph G is used.

$(e : T \to T') \in G$ means that T' can start only if T has already been finished.
T is a predecessor of T'.

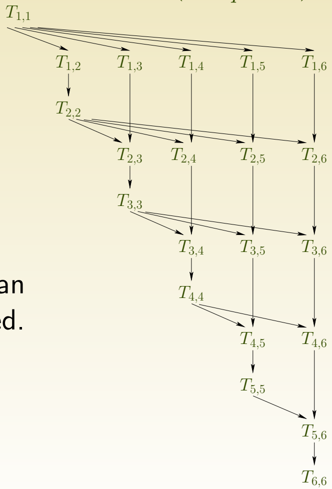


Precedence

If $T \perp T'$, then they should be ordered with the sequential execution order. $T \prec T'$ if $T \perp T'$ and $T <_{seq} T'$.

More precisely $\prec$ is defined as the transitive closure of $(<_{seq} \cap \perp)$.

for $i = 1$ **to** n **do**
  Task $T_{i,i}$: $x(i) \leftarrow b(i)/a(i,i)$
  for $j = i+1$ **to** n **do**
    Task $T_{i,j}$: $b(j) \leftarrow b(j) - a(j,i) \times x(i)$



A dependency graph G is used.

$(e : T \rightarrow T') \in G$ means that T' can start only if T has already been finished. T is a predecessor of T'.

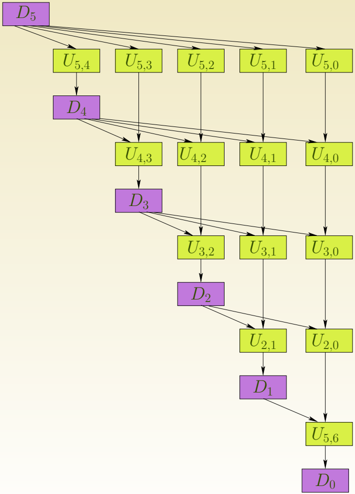
Transitivity arcs are generally omitted.

If I_1 writes in z and I_2 reads/writes z, then I_1 and I_2 should be done in the right (sequential) order [Bernstein66]

Data access define dependencies between instructions/tasks

```python
import numpy as np
A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
              [0, 0, 6, 1] , [0, 0, 0, 4]], float)
b = np.array([6, 5, -4, 8], float)

n = len(b)
x = np.zeros(n, float)
S = np.zeros(n, float)
for j in reversed(range(0,n)):      # from the bottom to the top
  x[j] = (b[j] - S[j]) / A[j][j]    # the division        (D_j)
  for i in range(0,j):              # A true parallel loop
     S[i] = S[i] + A[i][j] * x[j]   # Update              (U_i,j)
```



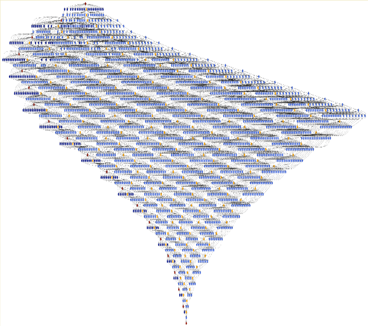
. . . pour obtenir des graphes de tâches

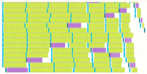
If I_1 writes in z and I_2 reads/writes z, then I_1 and I_2 should be done in the right (sequential) order [Bernstein66]

Data access define dependencies between instructions/tasks

```python
import numpy as np
A = np.array([[1, 1, 1, 1], [0, 1, -3, -1],
              [0, 0, 6, 1] , [0, 0, 0, 4]], float)
b = np.array([6, 5, -4, 8], float)

n = len(b)
x = np.zeros(n, float)
S = np.zeros(n, float)
for j in reversed(range(0,n)):     # from the bottom to the top
  x[j] = (b[j] - S[j]) / A[j][j]   # the division       (D_j)
  for i in range(0,j):             # A true parallel loop
     S[i] = S[i] + A[i][j] * x[j]  # Update             (U_i,j)
```



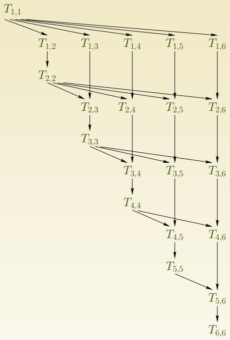


- allows to adapt granularity
- optimized versions depending on resources (CPU/GPU/auto-tuning)
- dynamic load-balancing
- more portables performances

From Coarse-grain Task Graphs. . .

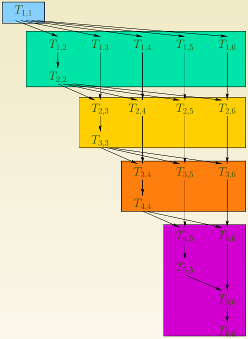
The previous task graph comes from a low-level analysis of the code.

- It probably makes little sense to do a parallel implementation with MPI with such a low task granularity.
- Can totally make sense with OpenMP.
- Such task graphs can also be used by compilers to do code optimization by exploiting multiple functional units, pipelines functional units, etc.
- With blocking these tasks could become MPI (parallel) tasks.

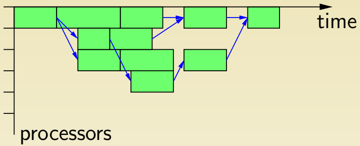


The previous task graph comes from a low-level analysis of the code.

- It probably makes little sense to do a parallel implementation with MPI with such a low task granularity.
- Can totally make sense with OpenMP.
- Such task graphs can also be used by compilers to do code optimization by exploiting multiple functional units, pipelines functional units, etc.
- With blocking these tasks could become MPI (parallel) tasks.

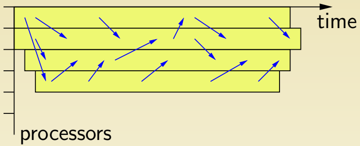


. . . to Parallel Tasks

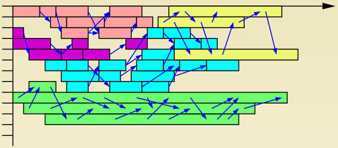


Hide applications' complexity

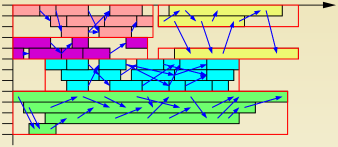
. . . to Parallel Tasks



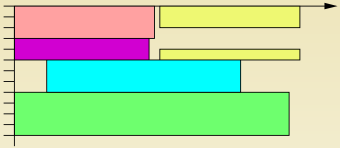
Hide applications' complexity

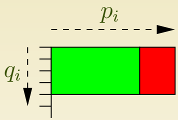


Hide applications' complexity



Hide applications' complexity





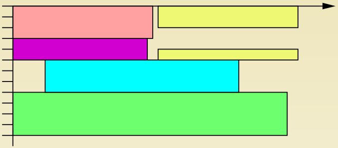
Hide applications' complexity

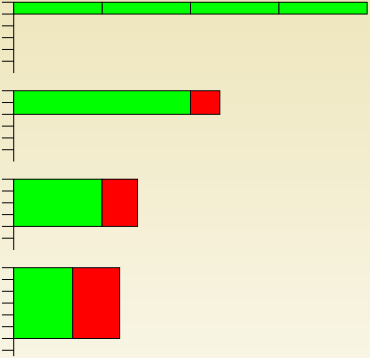
3 versions:

- Rigid Tasks

The execution time *generally* decreases with the number of processors but the penalty incurred by communications and synchronizations increases.

. . . to Parallel Tasks





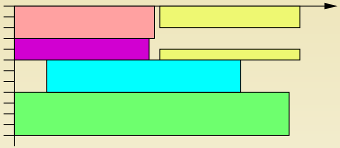
Hide applications' complexity

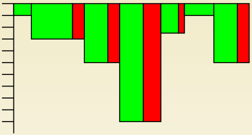
3 versions:

- Rigid Tasks
- Moldable Tasks

The execution time *generally* decreases with the number of processors but the penalty incurred by communications and synchronizations increases.

. . . to Parallel Tasks





Hide applications' complexity

3 versions:

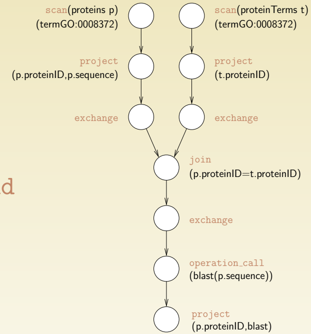
- Rigid Tasks
- Moldable Tasks
- Malleable Tasks

The execution time *generally* decreases with the number of processors but the penalty incurred by communications and synchronizations increases.

Workflow

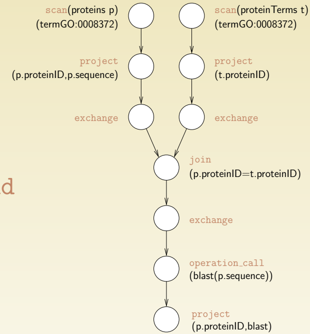
Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



Task-graph do not necessarily come from instruction-level analysis.

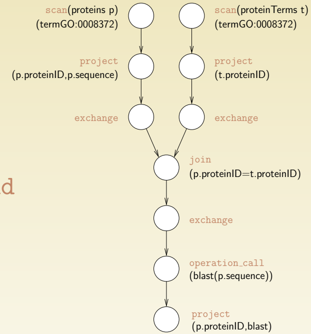
```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



- Each task may be a parallel job. . .

Task-graph do not necessarily come from instruction-level analysis.

```
select p.proteinID,
       blast(p.sequence)
from proteins p, proteinTerms t
where p.proteinID = t.proteinID and
t.term = GO:0008372
```



- Each task may be a parallel job...
- Each edge depicts a dependency i.e. most of the times some data to transfer.

Conclusion

I have presented you a few different parallel program models:

- rigid jobs
- moldable jobs
- dynamic jobs
- malleable jobs
- divisible jobs
- BSP jobs
- DAGs of the previous jobs

The rationale behind all these models is:

- the diversity and the complexity of parallel programs;
- the level of details we need/wish to expose to the one in charge of the execution.

Modeling is an art.

You have to know your application to know what is negligible and what is important. Even if your model is imperfect, you may still derive interesting results.

Outline

Outline

Some questions you need to answer

Preemption Are we allowed to suspend a program and resume it later ?

- Resumed from the beginning or from where it was stopped?
- May be resumed on another machine or not (migration)?
- Does preemption/migration has a cost or not?

Release dates Are all tasks available at the very beginning or not?

Deadlines Are the tasks associated to a deadline before which they should complete? What happens when the deadline is missed?

Dependencies Are there dependencies between tasks (DAGs)?

Users Are there many users and should this be taken into account?

Long-term vs. short-term What kind of constraints do you have on the time needed to take your scheduling decisions?

Online vs. Off-line, clairvoyance

What kind of of information do you have to make your scheduling choices?

Off-line You know everything (release dates and processing time of each task) at the very beginning.
It is the "simplest" setting and will give you insights on your scheduling problem even though these hypothesis do not really hold in practice.
This kind of problem should thus be studied *before everything else*.

On-line/clairvoyant You do not know in advance when tasks arrive. However, once a new task are available, you know its computation time.

On-line/non clairvoyant You know nothing!

Sometimes (often?), reality is in between:

- We could have "informations" about the task arrival (e.g., periodic creation, random process, use the past to predict the future).
- We could have "informations" about the task computation requirement (e.g., mix of short tasks and long tasks).

Outline

Criteria: Intuitive Notion

CPU utilization (max) percent usage of CPU. Only useful computations (mix CPU, I/O; preemption overhead).

Throughput (max) *average* number of tasks that complete their execution per time-unit.

Makespan (min) Completion time of the last finishing task.

Load (min) Completion time of the last finishing task for a given processor.

Turnaround Time/Response Time/Flow (min) amount of time it takes between the task arrival and its completion.

Waiting Time (min) amount of time spent waiting for being executed.

Slowdown/Stretch (min) slowdown factor encountered by a task relative to the time it would take on an unloaded system.

The previous quantities are task- or CPU-centric and need to be aggregated into a single objective function.

- max (the worst case)
- average: arithmetic (i.e. sum) or something else...
- variance (to be "fair" between the tasks).

Criteria: Classical Definitions

A given task T_i is defined by:

- processing time p_i
- release date r_i
- completion time C_i
- (number of required processors q_i)
- (deadline d_i)

Completion Time

- Makespan: $C_{\max} = \max_i C_i$
  This metric is the most classical and is relevant when scheduling a *single* application.
- Total Completion Time: $SC = \sum_i C_i$

Criteria: Classical Definitions

A given task T_i is defined by:

- processing time p_i
- release date r_i
- completion time C_i
- (number of required processors q_i)
- (deadline d_i)

Response Time

$$F_i = C_i - r_i$$

- Maximum Flow Time: $F_{\max} = \max_i F_i$
- Total Completion Time: $SF = \sum_i F_i = SC - \sum_i r_i$

Criteria: Classical Definitions

A given task T_i is defined by:

- processing time p_i
- release date r_i
- completion time C_i
- (number of required processors q_i)
- (deadline d_i)

Waiting Time

$$W_i = C_i - r_i - p_i$$

- Maximum Waiting time: $W_{\max} = \max_i W_i$
- Total Waiting Time: $SW = \sum_i W_i = SF - \sum_i p_i$

Criteria: Classical Definitions

A given task T_i is defined by:

- processing time p_i
- release date r_i
- completion time C_i
- (number of required processors q_i)
- (deadline d_i)

Slowdown

$$S_i = \frac{C_i - r_i}{p_i}$$

- Maximum Stretch: $S_{\max} = \max_i S_i$
- Total Stretch: $SS = \sum_i S_i$

Outline

NP-completeness

Most scheduling problem are NP-complete but you may be lucky. . . So the first question to answer is: P or NP-hard ?

For a given objective function Obj:

Definition: Decision and Optimization.

$Dec(M)$: Is there a schedule σ such that $Obj(\sigma) \leqslant M$?

Opt: Find M^* such that $M^* = \min_\sigma Obj(\sigma)$.

If Dec can be solved in polynomial time, then so can Opt (using a dichotomy). And conversely. . .

Note that since $SW(\sigma) = SF(\sigma) - \sum_i p_i = SC(\sigma) - \sum_i r_i - \sum_i p_i$, all these problem are equivalent on a complexity point of view.

Worst-Case Analysis: ϱ-approximation

Your scheduling problem is NP-hard so you need to propose a heuristic and compare it to the best possible solution.
Consider a given objective function Obj.

Definition: ϱ-approximation.

An algorithm $\mathcal{A}$ is a ϱ-approximation
iff
for any instance I, $Obj(\mathcal{A}(I)) \leqslant \varrho.Obj^*(I)$.

The approximation ratio of $\mathcal{A}$ is:

$$\varrho(\mathcal{A}) = \max_I \frac{Obj(\mathcal{A}(I))}{Obj^*(I)}$$

Note that even though $SW(\sigma) = SF(\sigma) - \sum_i p_i = SC(\sigma) - \sum_i r_i - \sum_i p_i$, these problem are **not** equivalent on an approximation point of view.

Worst-Case Analysis: ϱ-competitive

What is the best solution to an online problem (where the heuristic doesn't know in advance the jobs arrival) ?
We keep comparing to the best possible solution, i.e. the one that knows everything.

Definition: ϱ-competitive.

An algorithm $\mathcal{A}$ is a ϱ-approximation
iff
for any instance I, $Obj(\mathcal{A}(I)) \leqslant \varrho.Obj^*(I)$.

The approximation ratio of $\mathcal{A}$ is:

$$\varrho(\mathcal{A}) = \max_I \frac{Obj(\mathcal{A}(I))}{Obj^*(I)}$$

It is the same definition except that it applies to online algorithms.
For such a pessimistic evaluation, one commonly uses an adversary.

If we have a probability distribution over the set of instances, Obj can thus be seen as a random variable.
We can define the expectation of Obj.

$$\mathbb{E}[Obj(\mathcal{A})] = \int_I Obj(\mathcal{A}(I))p(I).dI = \sum_I Obj(\mathcal{A}(I))p(I)$$

People often try to evaluate at (at least through experiments)

$$\varrho(\mathcal{A}) = \int_I \frac{Obj(\mathcal{A}(I))}{Obj^*(I)} p(I).dI \text{ or}$$

$$\varrho(\mathcal{A}) = \frac{\mathbb{E}[Obj(\mathcal{A})]}{\mathbb{E}[Obj^*]} = \frac{\int_I Obj(\mathcal{A}(I))p(I).dI}{\int_I Obj^*(I)p(I).dI}$$

However, in the literature, there are many different ways of comparing random variables (and thus to compare and evaluate algorithms). These techniques will be presented in much more details later.

Outline

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\langle\alpha|\beta|\gamma\rangle$ [Brucker-Book]

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\langle\alpha|\beta|\gamma\rangle$ [Brucker-Book]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\langle\alpha|\beta|\gamma\rangle$ [Brucker-Book]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;

Graham Notation

Many parameter can change in a scheduling problem. Graham has then proposed the following classification : $\langle\alpha|\beta|\gamma\rangle$ [Brucker-Book]

- α is the processor environment (a few examples):
  - $\emptyset$: single processor;
  - P: identical processors;
  - Q: uniform processors;
  - R: unrelated processors;
- β describe task and resource characteristics (a few examples):
  - $pmtn$: preemption;
  - $prec$, $tree$ or $chains$: general precedence constraints, tree constraints, set of chain constraints and independent tasks otherwise;
  - r_j: tasks have release dates;
  - $p_j = p$ or $\underline{p} \leqslant p_j \leqslant \overline{p}$: all task have processing time equal to p, or comprised between $\underline{p}$ and $\overline{p}$, or have arbitrary processing times otherwise;
  - $\tilde{d}$: deadlines;
- γ denotes the optimization criterion (a few examples):
  - $C_{\max}$: makespan;
  - $\sum C_i$: average completion time;
  - $\sum w_i C_i$: weighted A.C.T;
  - $L_{\max}$: maximum lateness $(\max C_i - d_i)$;
  - ...

Graham Notation: Exercise

Understand the following problems and propose a practical situation to illustrate them:

- $\langle P|prec|C_{\max}\rangle$
- $\langle P|q_j, prec|C_{\max}\rangle$
- $\langle P|q_j|F_{\max}\rangle$
- $\langle 1|r_j; pmtn|S_{\max}\rangle$
- $\langle 1|r_j; pmtn, d_i|L_{\max}\rangle$

Scheduling. . .

Scheduling is a very generic word that encompass a very wide range of situations, problems and analysis techniques.

Scheduling is generally about deciding *who*, *where* and *when*.

It is thus almost everywhere and when you start looking at a given scheduling problem, with very high probability, many people already worked on it.

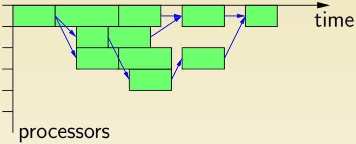
Doing a serious and **thorough bibliographical study** is thus of uttermost importance!

Outline

Outline

Need for Batch Scheduling

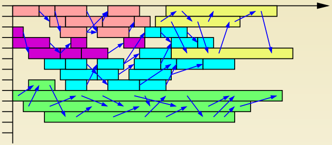
- Parallel Tasks from Scientific Computations (simulation, medical)



Need for Batch Scheduling

- Parallel Tasks from Scientific Computations (simulation, medical)

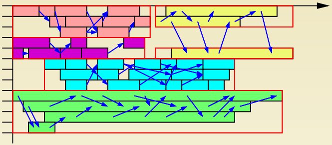
- Parallel Tasks from Scientific Computations (simulation, medical)



- When one purchases a cluster, typically many users want to use it.
  - One cannot let them step on each other's toes
  - Every user wants to be on a dedicated machine
  - Applications are written assuming some amount of RAM, some notion that all processors go at the same speed, etc.

Need for Batch Scheduling

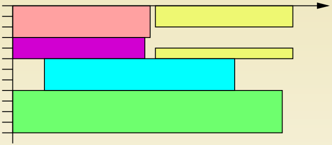
- Parallel Tasks from Scientific Computations (simulation, medical)



- When one purchases a cluster, typically many users want to use it.
  - One cannot let them step on each other's toes
  - Every user wants to be on a dedicated machine
  - Applications are written assuming some amount of RAM, some notion that all processors go at the same speed, etc.

Need for Batch Scheduling

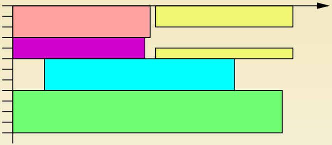
- Parallel Tasks from Scientific Computations (simulation, medical)



- When one purchases a cluster, typically many users want to use it.
  - One cannot let them step on each other's toes
  - Every user wants to be on a dedicated machine
  - Applications are written assuming some amount of RAM, some notion that all processors go at the same speed, etc.

Need for Batch Scheduling

- Parallel Tasks from Scientific Computations (simulation, medical)



- When one purchases a cluster, typically many users want to use it.
  - One cannot let them step on each other's toes
  - Every user wants to be on a dedicated machine
  - Applications are written assuming some amount of RAM, some notion that all processors go at the same speed, etc.

The Job Scheduler is the entity that prevents them from stepping on each other's toes

The Job Scheduler gives out nodes to applications

Batch Scheduling

Each job is defined as a Number of nodes (q_i) and a Time (p_i):

I want 6 nodes for 1h

Typically users are "charged" against an "allocation": e.g. "*You only get 100 CPU hours per week*".

A batch scheduler is a central middleware to manage resources (e.g. processors) of parallel machines:

- accept jobs (computing tasks) submitted by users
- decide when and where jobs are executed
- start jobs execution

They take into account:

- unavailability of some nodes
- users jobs mutual exclusion
- specific needs for jobs (memory, network, ...)

While trying to :

- maximize resources usage
- be fair among users

Batch Scheduling

Typical wanted features:

- Interactive mode
- Batch mode
- Parallel jobs support
- Multi-queues with priorities
- Admission policies (limit on usage, notions of user groups, power users)
- Resources matching
- File staging
- Jobs dependences
- Backfilling
- Reservations
- Best effort jobs
- Environment reconfiguration

There are many existing batch schedulers : LSF, PBS/Torque, Maui scheduler, Sun Grid Engine, EASY, OAR, . . .

These are complex systems with many config options !

Main Batch Schedulers Features

	OpenPBS	SGE	Maui Scheduler (+ OpenPBS)	OAR
Interactive mode	×	×	×	×
Batch mode	×	×	×	×
Parallel jobs support	×	×	×	×
Multi-queues with priorities	×	×	×	×
Resources matching	×	×	×	×
Admission policies	×	×	×	×
File staging	×	×	×	
Jobs dependences	×	×	×	
Backfilling			×	×
Reservations			×	×
Best effort jobs				×
Environment reconfiguration				×
Fair sharing			×	×

Outline

A batch scheduler may have to solve something like

$$\langle P|size_j, prec, r_j|F_{\max}\rangle$$

But this is quite a complicated problem.

- In particular $\langle P2||C_{\max}\rangle$ is already (weakly) NP-hard:

  Given n numbers $a_1, \dots, a_n$ whose sum is even, find a subset of indices I such that $\sum_{i\in I} a_i = \sum_{i\notin I} a_i$

A batch scheduler may have to solve something like

$$\langle P | size_j, prec, r_j | F_{\max} \rangle$$

But this is quite a complicated problem.

- In particular $\langle P2 || C_{\max} \rangle$ is already (weakly) NP-hard:

  Given n numbers $a_1, \dots, a_n$ whose sum is even, find a subset of indices I such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$

- So let's look at $\langle P | p_j = 1 | C_{\max} \rangle$.

Base results

A batch scheduler may have to solve something like

$$\langle P|size_j, prec, r_j|F_{\max}\rangle$$

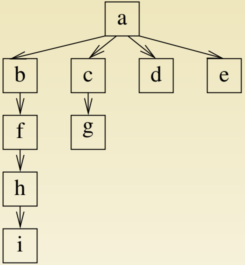
But this is quite a complicated problem.

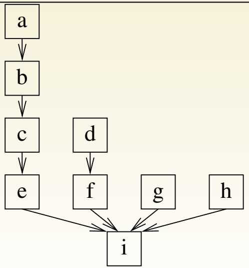
- In particular $\langle P2||C_{\max}\rangle$ is already (weakly) NP-hard:

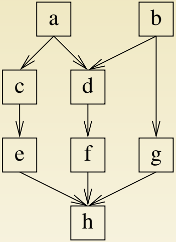
  Given n numbers $a_1, \dots, a_n$ whose sum is even, find a subset of indices I such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$

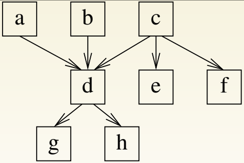
- So let's look at $\langle P|p_j = 1|C_{\max}\rangle$. Alright, this one is really trivial. ☺
- So maybe we should look at $\langle P|p_j = 1, prec|C_{\max}\rangle$

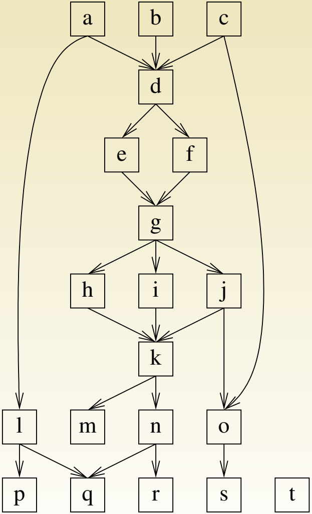
Assume we have 2 machines:











Prioritize according to the "critical path"

We can define a notion of "depth" and schedule ready tasks accordingly ("highest" tasks go first).

This is known as Hu's algoritm and works great for $intrees$ and $outtrees$ but not the general case:

- $\langle P|p_j = 1, intree|C_{\max}\rangle$ is polynomial (Hu, 1961). Note that although the result seems trivial, the original proof was 8 pages long! We had to wait a bit to get a 2 page long proof (James A. M. McHugh, 1984)
- $\langle P|p_j = 1, prec|C_{\max}\rangle$ is NP-hard
- $\langle P2|p_j = 1, prec|C_{\max}\rangle$ is polynomial (Coffman, 1972) but the algorithm is very specific to the 2 machine case and does not provide much intuition.
- One of the difficulty is to decide between scheduling critical tasks or tasks that will release a lot of work

List Scheduling

When simple problems are hard, we should try to find good approximation heuristics. A ϱ-approximation is an algorithm whose output is never more than a factor ϱ times the optimum solution.

Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).

When simple problems are hard, we should try to find good approximation heuristics. A ϱ-approximation is an algorithm whose output is never more than a factor ϱ times the optimum solution.

Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).

Any strategy that does not let on purpose a processor idle is efficient [Coffman76]. Such a schedule is called list-schedule.

Theorem 1: Coffman.

Let $G = (V, E, w)$ be a DAG of sequential tasks, p the number of processors, and σ_p a list-schedule of G on p processors.

$$C_{\max}(\sigma_p) \leqslant \left(2 - \frac{1}{p}\right) C^*_{\max}(p) .$$

When simple problems are hard, we should try to find good approximation heuristics. A ϱ-approximation is an algorithm whose output is never more than a factor ϱ times the optimum solution.

Natural idea: using greedy strategy like trying to allocate the most possible task at a given time-step. However at some point we may face a choice (when there is more ready tasks than available processors).

Any strategy that does not let on purpose a processor idle is efficient [Coffman76]. Such a schedule is called list-schedule.

Theorem 1: Coffman.

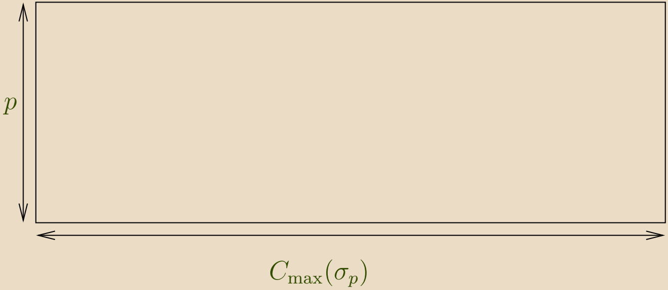
Let $G = (V, E, w)$ be a DAG of sequential tasks, p the number of processors, and σ_p a list-schedule of G on p processors.

$$C_{\max}(\sigma_p) \leqslant \left(2 - \frac{1}{p}\right) C^*_{\max}(p) \ .$$

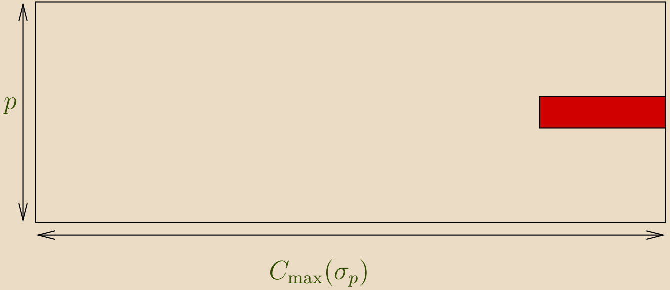
Most of the time, list-heuristics are based on the critical path.

List Scheduling: proving the Coffman result

Proof.

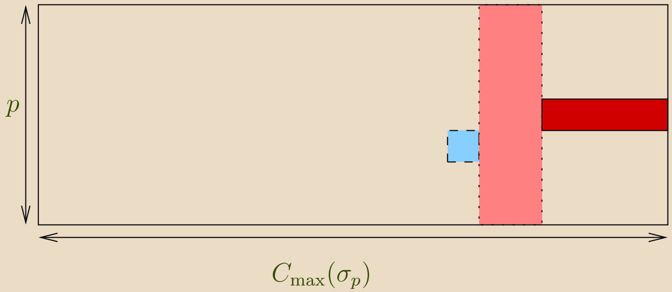


Proof.



List Scheduling: proving the Coffman result

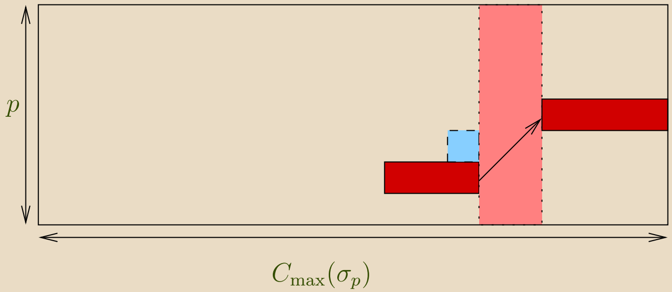
Proof.



□

List Scheduling: proving the Coffman result

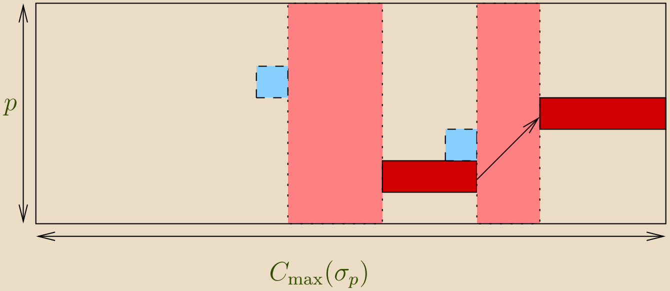
Proof.



□

List Scheduling: proving the Coffman result

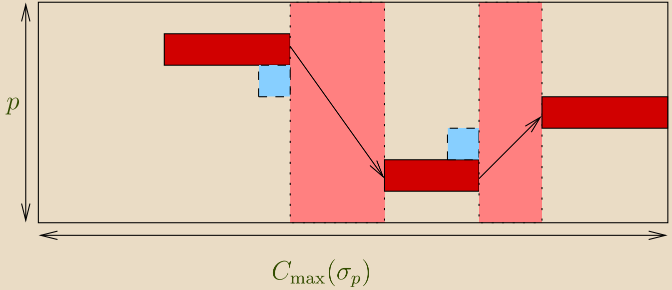
Proof.



□

List Scheduling: proving the Coffman result

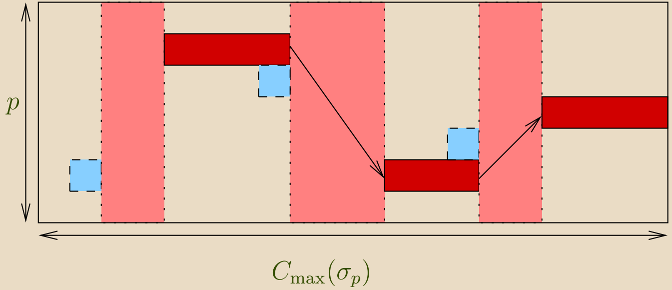
Proof.



□

List Scheduling: proving the Coffman result

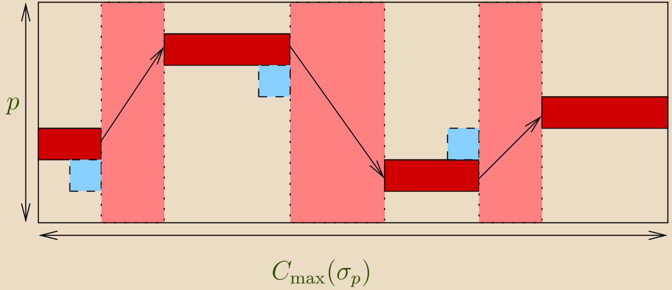
Proof.



□

List Scheduling: proving the Coffman result

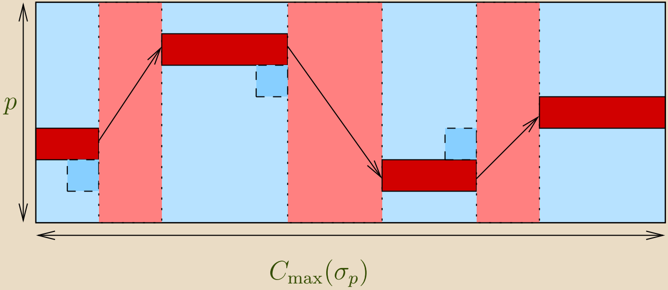
Proof.



□

List Scheduling: proving the Coffman result

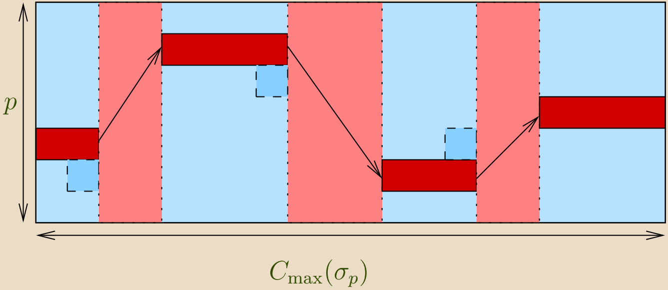
Proof.



□

List Scheduling: proving the Coffman result

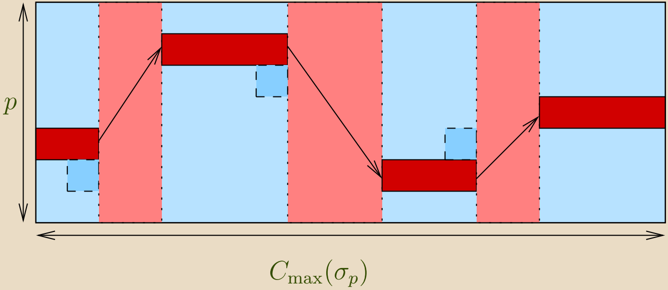
Proof.



Therefore, $Idle \leqslant (p-1).w(\Phi)$ for some Φ

□

Proof.



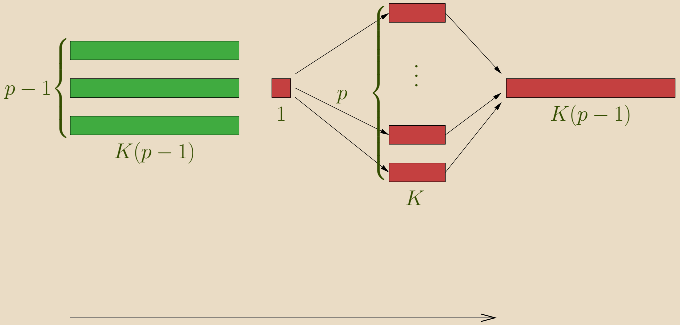
Therefore, $Idle \leqslant (p-1).w(\Phi)$ for some Φ

Hence,

$$p.C_{\max}(\sigma_p) = Idle + Seq \leqslant (p-1)w(\Phi) + Seq$$
$$\leqslant (p-1)C^*_{\max}(p) + p.C^*_{\max}(p) = (2p-1)C^*_{\max}(p)$$

One can actually prove that this bound cannot be improved.

Proof.

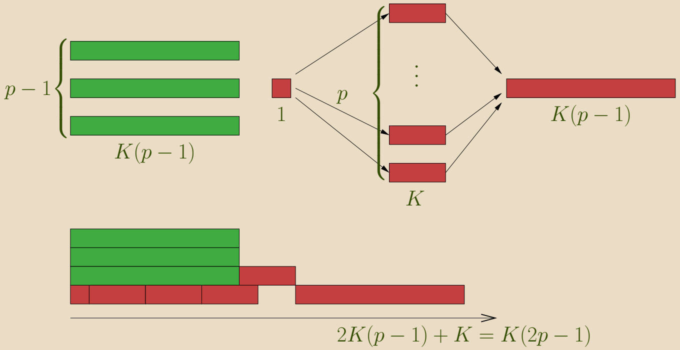


□

List Scheduling: proving the Coffman result

One can actually prove that this bound cannot be improved.

Proof.

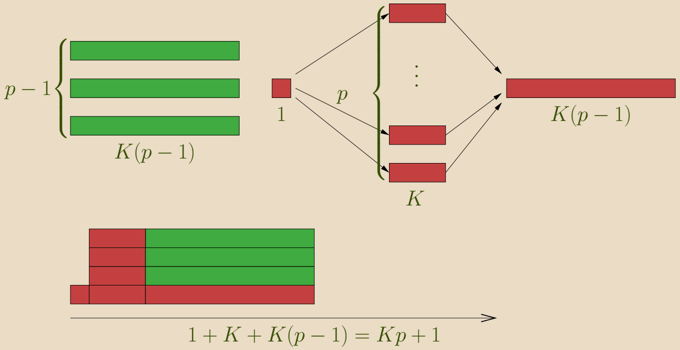


□

List Scheduling: proving the Coffman result

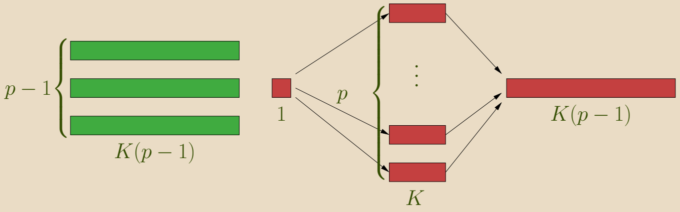
One can actually prove that this bound cannot be improved.

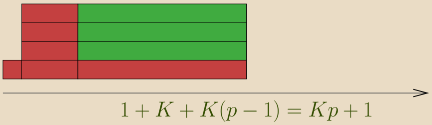
Proof.



One can actually prove that this bound cannot be improved.

Proof.

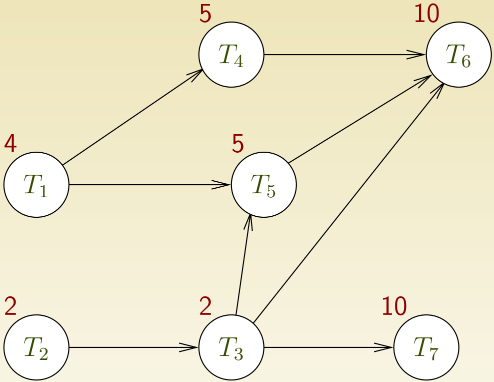


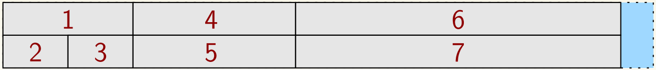


$$\varrho \geqslant \frac{K(2p-1)}{Kp+1} \xrightarrow[K\to\infty]{} \frac{2p-1}{p}$$

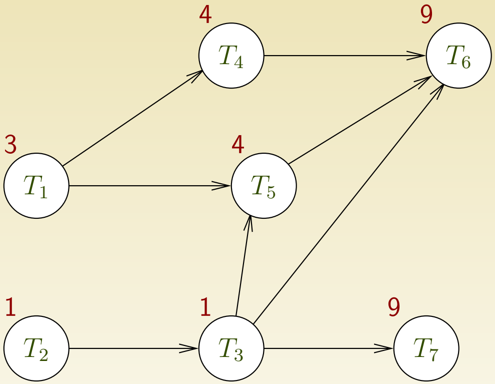
□

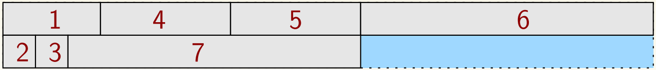
List scheduling Anomalies





$$MS = 19$$





$$MS = 20$$

List Scheduling for Parallel Rigid Tasks

Let us assume we have n independent rigid jobs $J_1 = (p_1, q_1), \ldots, J_n = (p_n, q_n)$ and m machines.
Let us denote by T^* the optimal makespan for this instance.

List Scheduling for Parallel Rigid Tasks

Let us assume we have n independent rigid jobs $J_1 = (p_1, q_1), \ldots, J_n = (p_n, q_n)$ and m machines.
Let us denote by T^* the optimal makespan for this instance.
Let us consider a list schedule of makespan T. Let us denote by $q(t)$ the number of active processors at time t.
We have $\forall t_1, t_2 \in [0, T] : t_1 \leqslant t_2 - T^* \Rightarrow q(t_1) + q(t_2) > m$ (otherwise, the tasks running at time t_2 could have been run at time t_1).

List Scheduling for Parallel Rigid Tasks

Let us assume we have n independent rigid jobs $J_1 = (p_1, q_1), \dots, J_n = (p_n, q_n)$ and m machines.

Let us denote by T^* the optimal makespan for this instance.

Let us consider a list schedule of makespan T. Let us denote by $q(t)$ the number of active processors at time t.

We have $\forall t_1, t_2 \in [0, T] : t_1 \leqslant t_2 - T^* \Rightarrow q(t_1) + q(t_2) > m$ (otherwise, the tasks running at time t_2 could have been run at time t_1).

Let us assume that $T > 2T^*$. Then we have:

$$mT^* \geqslant \sum_i q_i p_i = \int_0^T q(t) = \int_0^{2T^*} q(t) + \int_{2T^*}^T q(t)$$

$$\geqslant \underbrace{\int_0^{T^*} q(t) + q(t + T^*)}_{>mT^*} + \underbrace{\int_{2T^*}^T q(t)}_{\geqslant 0}, \text{which is absurd.}$$

Let us assume we have n independent rigid jobs $J_1 = (p_1, q_1), \dots, J_n = (p_n, q_n)$ and m machines.
Let us denote by T^* the optimal makespan for this instance.
Let us consider a list schedule of makespan T. Let us denote by $q(t)$ the number of active processors at time t.
We have $\forall t_1, t_2 \in [0, T] : t_1 \leqslant t_2 - T^* \Rightarrow q(t_1) + q(t_2) > m$ (otherwise, the tasks running at time t_2 could have been run at time t_1).
Let us assume that $T > 2T^*$. Then we have:

$$mT^* \geqslant \sum_i q_i p_i = \int_0^T q(t) = \int_0^{2T^*} q(t) + \int_{2T^*}^T q(t)$$

Theorem 2.

List-scheduling has an approximation factor of 2 for minimizing the Cmax of Parallel Rigid Tasks.

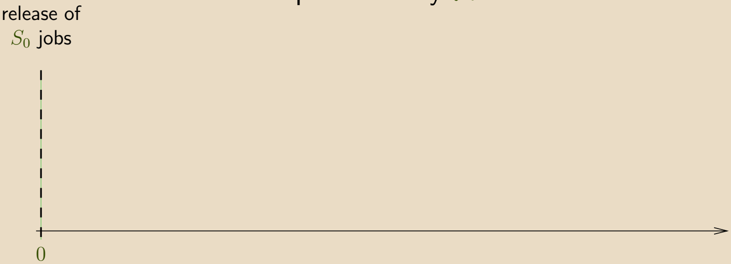
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

Going Online

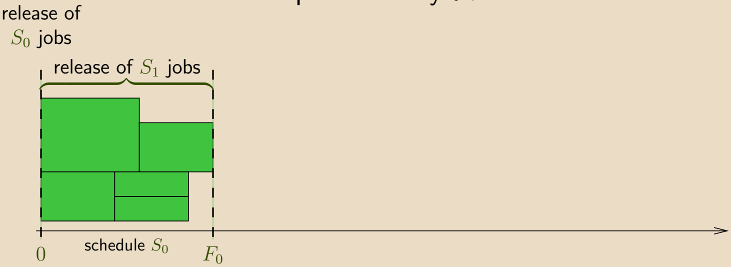
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

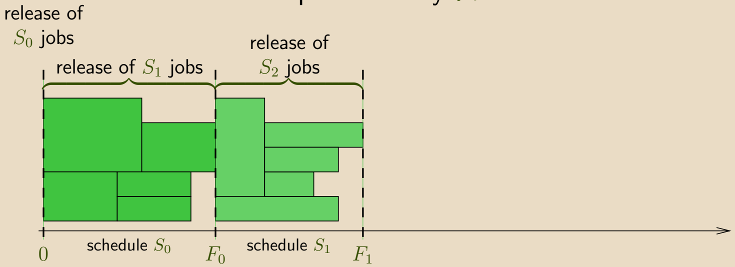
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

Going Online

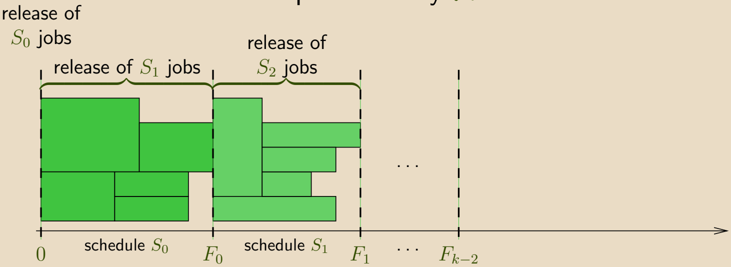
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

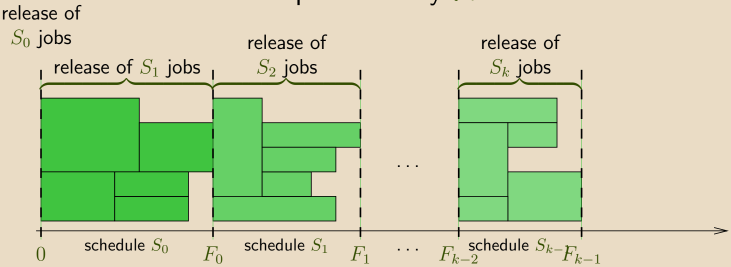
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

Going Online

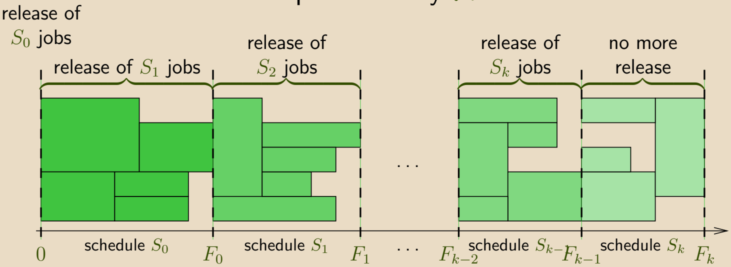
How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Let us look at the schedule produced by $\mathcal{A}$ on an instance $\mathcal{I}$.



□

Going Online

How can we use the previous result when going online?

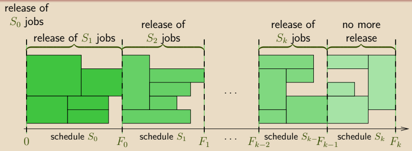
Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Consider $\mathcal{I}'$ where S_k jobs are released at time F_{k-2}. We have:

$$C^*_{\max}(\mathcal{I}') \leqslant C^*_{\max}(\mathcal{I}).$$



□

Going Online

How can we use the previous result when going online?

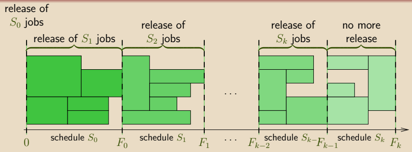
Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Consider $\mathcal{I}'$ where S_k jobs are released at time F_{k-2}. We have:

$$C^*_{\max}(\mathcal{I}') \leqslant C^*_{\max}(\mathcal{I}).$$



- $F_{k-2} + F_k - F_{k-1} \leqslant \varrho C^*_{\max}(\mathcal{I}')$

□

How can we use the previous result when going online?

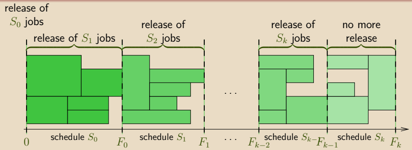
Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Consider $\mathcal{I}'$ where S_k jobs are released at time F_{k-2}. We have:

$$C^*_{\max}(\mathcal{I}') \leqslant C^*_{\max}(\mathcal{I}).$$



- $F_{k-2} + F_k - F_{k-1} \leqslant \varrho C^*_{\max}(\mathcal{I}')$
- $F_{k-1} - F_{k-2} \leqslant \varrho C^*_{\max}(\mathcal{I}')$

□

Going Online

How can we use the previous result when going online?

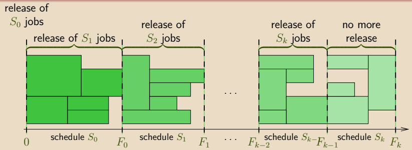
Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

Proof.

Consider $\mathcal{I}'$ where S_k jobs are released at time F_{k-2}. We have:

$$C^*_{\max}(\mathcal{I}') \leqslant C^*_{\max}(\mathcal{I}).$$



- $F_{k-2} + F_k - F_{k-1} \leqslant \varrho C^*_{\max}(\mathcal{I}')$
- $F_{k-1} - F_{k-2} \leqslant \varrho C^*_{\max}(\mathcal{I}')$

Hence $F_k \leqslant 2\varrho C^*_{\max}(\mathcal{I}') \leqslant 2\varrho C^*_{\max}(\mathcal{I})$ □

Going Online

How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

- There is a PTAS for $\langle Q||C_{\max}\rangle$. Hence, there is an $(2+\varepsilon)$-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.

Going Online

How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

- There is a PTAS for $\langle Q||C_{\max}\rangle$. Hence, there is an $(2+\varepsilon)$-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.
- There is a 2 approximation $\langle Q||C_{\max}\rangle$. Hence, there is an 4-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.

Going Online

How can we use the previous result when going online?

Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

- There is a PTAS for $\langle Q||C_{\max}\rangle$. Hence, there is an $(2+\varepsilon)$-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.
- There is a 2 approximation $\langle Q||C_{\max}\rangle$. Hence, there is an 4-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.
- There is a 2 approximation $\langle P|size_j|C_{\max}\rangle$. Hence, there is an 4-competitive online clairvoyant algorithm for $\langle Q|size_j|C_{\max}\rangle$.

Going Online

How can we use the previous result when going online?

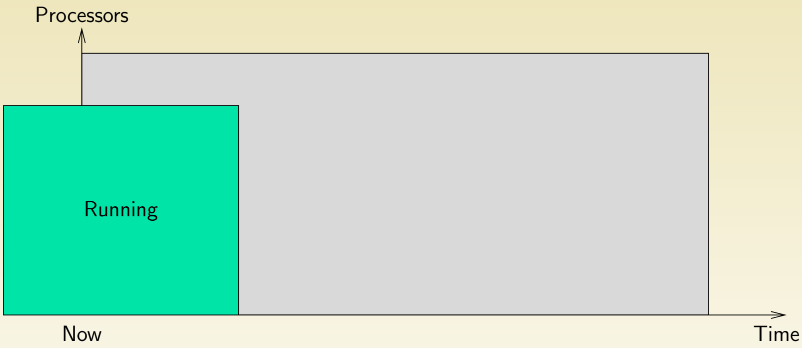
Theorem 3: [Shmoys91].

Let $\mathcal{A}$ be a polynomial-time ϱ-approximation for $\langle P|size_j|C_{\max}\rangle$. Based on $\mathcal{A}$, we can build a 2ϱ-competitive polynomial-time online clairvoyant algorithm for $\langle P|size_j, r_j|C_{\max}\rangle$.

- There is a PTAS for $\langle Q||C_{\max}\rangle$. Hence, there is an $(2+\varepsilon)$-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.
- There is a 2 approximation $\langle Q||C_{\max}\rangle$. Hence, there is an 4-competitive online clairvoyant algorithm for $\langle Q|r_j|C_{\max}\rangle$.
- There is a 2 approximation $\langle P|size_j|C_{\max}\rangle$. Hence, there is an 4-competitive online clairvoyant algorithm for $\langle Q|size_j|C_{\max}\rangle$.
- Actually, by doing a slightly finer analysis, on can show that the list-scheduling algorithm is a $(2-1/m)$-competitive non-clairvoyant algorithm for $\langle P|r_j|C_{\max}\rangle$.

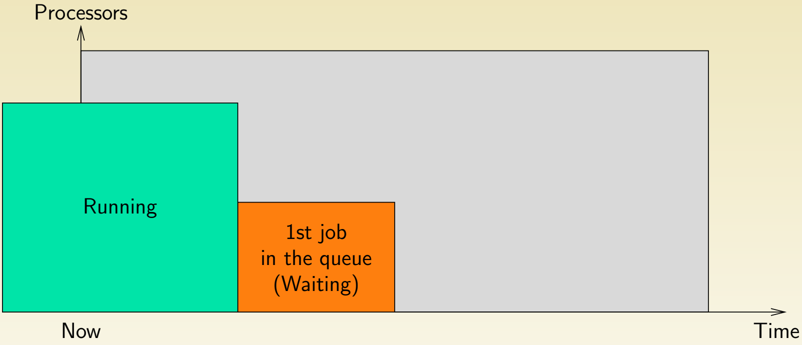
Outline

General Principle



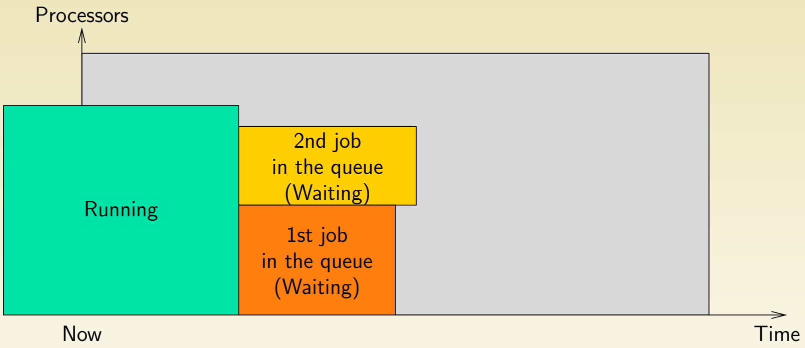
- Jobs arrive one after the other and are scheduled at arrival.

General Principle



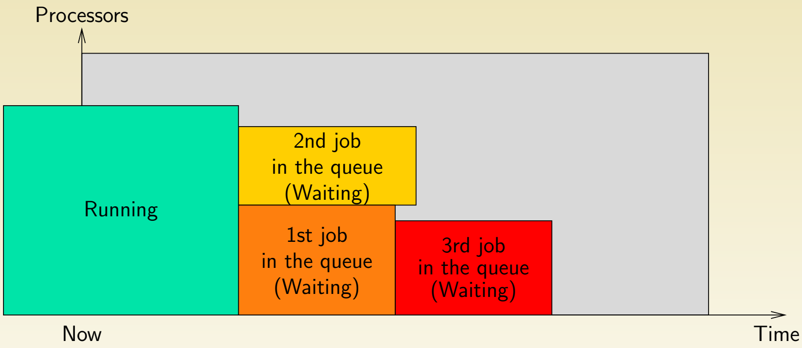
- Jobs arrive one after the other and are scheduled at arrival.

General Principle



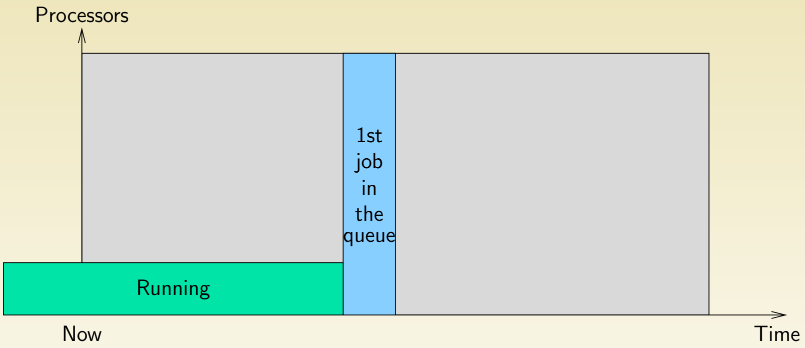
- Jobs arrive one after the other and are scheduled at arrival.

General Principle



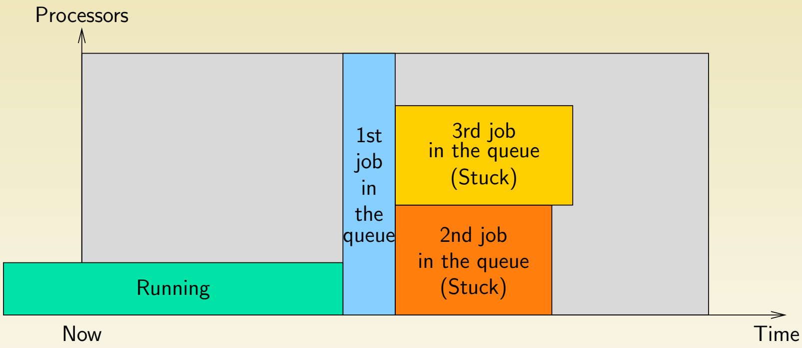
- Jobs arrive one after the other and are scheduled at arrival.

First Come First Served



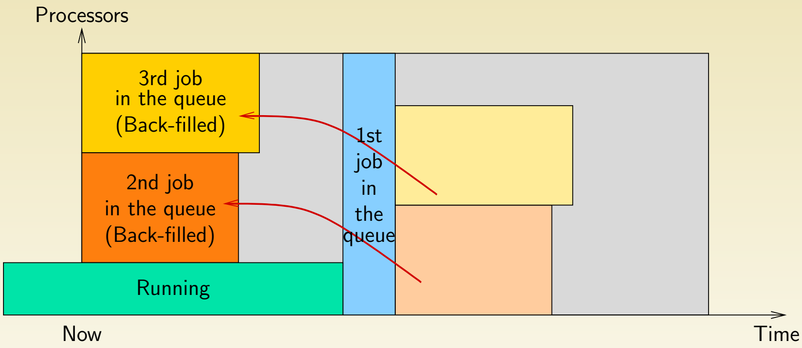
- FCFS = simplest scheduling option
- Fragmentation $\rightsquigarrow$ need for backfilling

First Come First Served



- FCFS = simplest scheduling option
- Fragmentation $\rightsquigarrow$ need for backfilling

First Come First Served



- FCFS = simplest scheduling option
- Fragmentation ⇝ need for backfilling

Backfilling: Question

- Which job(s) should be picked for promotion through the queue?
- Many heuristics are possible
- Two have been studied in detail
  - EASY
  - Conservative Back Filling (CBF)
- In practice EASY (or variants of it) is used, while CBF is not.
- Although, OAR, a recently proposed batch scheduler implements CBF.

Outline

EASY Backfilling

Extensible Argonne Scheduling System

Maintain only one *reservation*, for the first job in the queue.

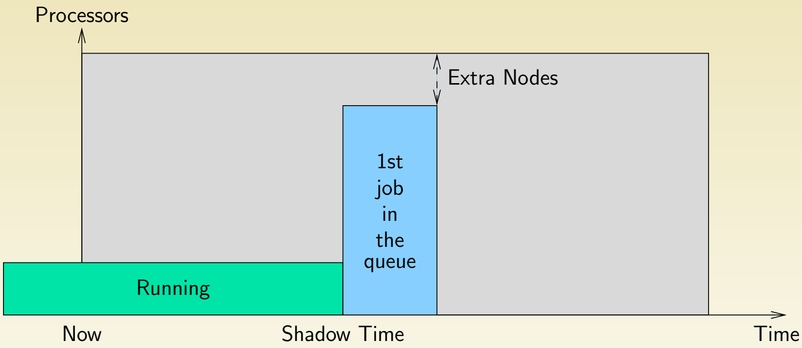
Definitions:

Shadow time
: time at which the first job in the queue starts execution

Extra nodes
: number of nodes idle when the first job in the queue starts execution

1. Go through the queue in order starting with the 2nd job.
2. Backfill a job if it will terminate by the shadow time, or it needs less than the extra nodes.

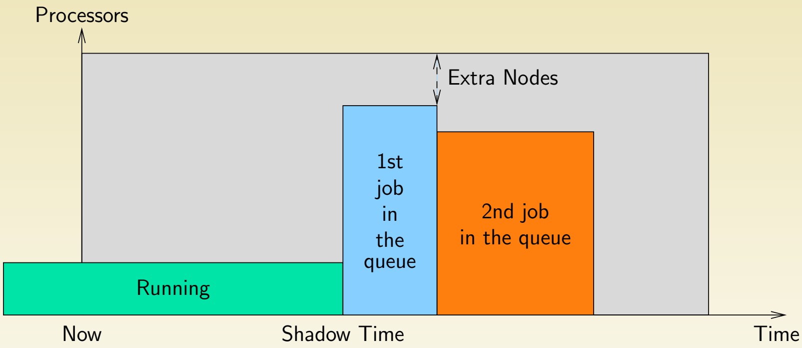
EASY



Property:

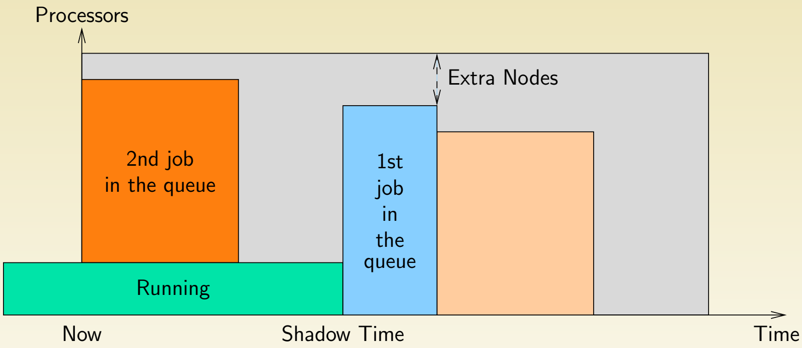
- The first job in the queue will never be delayed by backfilled jobs

EASY



Property:

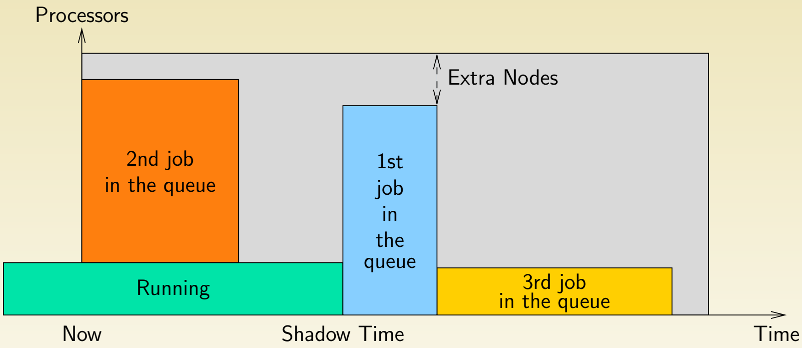
- The first job in the queue will never be delayed by backfilled jobs



Property:

- The first job in the queue will never be delayed by backfilled jobs

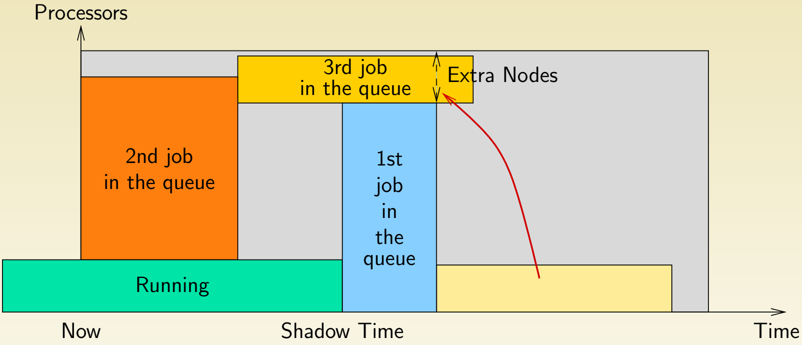
EASY



Property:

- The first job in the queue will never be delayed by backfilled jobs

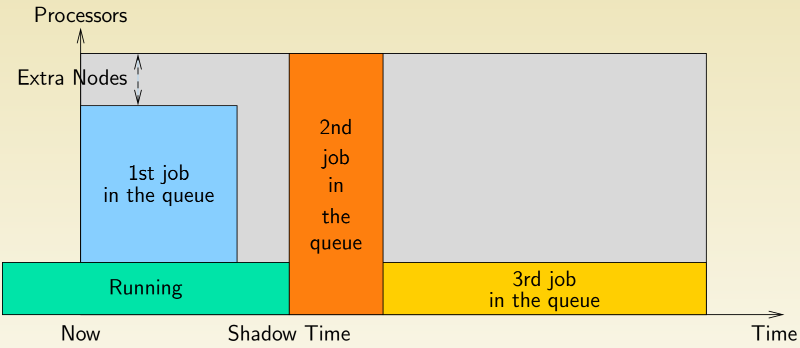
EASY



Property:

- The first job in the queue will never be delayed by backfilled jobs

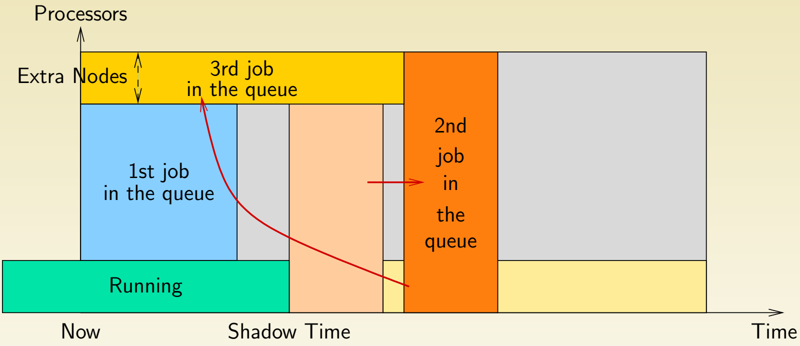
EASY



Property:

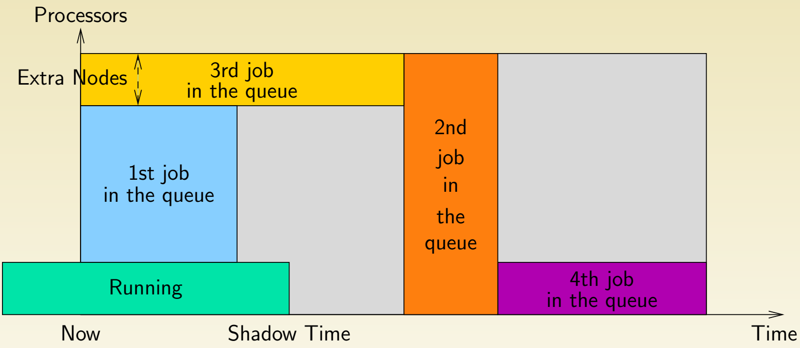
- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

EASY



Property:

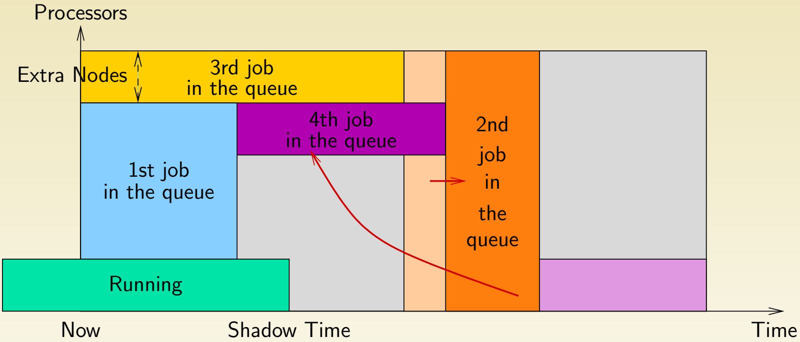
- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!



Property:

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

EASY



Property:

- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

EASY Properties

Unbounded Delay.
- The first job in the queue will never be delayed by backfilled jobs
- BUT, other jobs may be delayed infinitely!

No Starvation.
- Delay of first job is bounded by runtime of current jobs
- When the first job finishes, the second job becomes the first job in the queue
- Once it is the first job, it cannot be delayed further

Other approach.
- Conservative Backfilling. *EVERY* job has a *reservation*. A job may be backfilled only if it does not delay any other job ahead of it in the queue.
- Fixes the unbounded delay problem that EASY has. More complicated to implement (The algorithm must find holes in the schedule) though.
- EASY favors small long jobs and harms large short jobs.

Outline

When Does Backfilling Happen?

Possibly when

- A new job arrives

When Does Backfilling Happen?

Possibly when

- A new job arrives
- The first job in the queue starts

When Does Backfilling Happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

When Does Backfilling Happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

Users provide job runtime estimates (Jobs are killed if they go over).

When Does Backfilling Happen?

Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

Users provide job runtime estimates (Jobs are killed if they go over).

Trade-off:

- provide a conservative estimate: you goes through the queue faster (may be backfilled)
- provide a loose estimate: your job will not be killed

When Does Backfilling Happen?

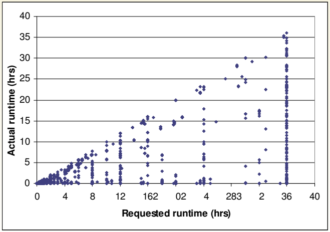
Possibly when

- A new job arrives
- The first job in the queue starts
- When a job finishes early

Users provide job runtime estimates (Jobs are killed if they go over). Trade-off:

- provide a conservative estimate: you goes through the queue faster (may be backfilled)
- provide a loose estimate: your job will not be killed

Are estimates accurate?



How Good is the Schedule ?

All of this is great, but how do we know what a “good” schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

How Good is the Schedule ?

All of this is great, but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

1. Turn-around time or flow (Wait time + Run time).
   Job 1 needs 1h of compute time and waits 1s
   Job 2 needs 1s of compute time and waits 1h
   Clearly Job 1 is really happy, and Job 2 is not happy at all

How Good is the Schedule ?

All of this is great, but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

1. Turn-around time or flow (Wait time + Run time).
   Job 1 needs 1h of compute time and waits 1s
   Job 2 needs 1s of compute time and waits 1h
   Clearly Job 1 is really happy, and Job 2 is not happy at all
2. Wait time (equivalent to "user happiness")
   Job 1 asks for 1 nodes and waits 1 h
   Job 2 asks for 512 nodes and waits 1h
   Again, Job 1 is unhappy while Job 2 is probably sort of happy.
   We need a metric that represents happiness for small, large, short, long jobs.

How Good is the Schedule ?

All of this is great, but how do we know what a "good" schedule is? FCFS, EASY, CFB, Random?

What we need are metrics to quantify how good a schedule is. It has to be an aggregate metric over all jobs

1. Turn-around time or flow (Wait time + Run time).
   Job 1 needs 1h of compute time and waits 1s
   Job 2 needs 1s of compute time and waits 1h
   Clearly Job 1 is really happy, and Job 2 is not happy at all
2. Wait time (equivalent to "user happiness")
   Job 1 asks for 1 nodes and waits 1 h
   Job 2 asks for 512 nodes and waits 1h
   Again, Job 1 is unhappy while Job 2 is probably sort of happy.
   We need a metric that represents happiness for small, large, short, long jobs.
3. Slowdown or Stretch (turn-around time divided by turn- around time if alone in the system)
   Doesn't really take care of the small/large problem. Could think of some scaling, but unclear !

Now What ?

Now we have a few metrics we can consider

We can run simulations of the scheduling algorithms, and see how they fare.

We need to test these algorithms in representative scenarios

Supercomputer/cluster traces. Collect the following for long periods of time:

- Time of submission
- How many nodes asked
- How much time asked
- How much time was actually used
- How much time spent in the queue

Uses of the traces:

1. Drive simulations
2. Come up with models of user behaviors

Sample Results

A type of experiments that people have done: replace user estimate by f times the actual run time

Possible to improve performance by multiplying user estimates by 2!

	EASY	CBF
Mean Slowdown		
KTH	-4.8%	-23.0%
CTC	-7.9%	-18.0%
SDSC	+4.6%	-14.2%
Mean Response time		
KTH	-3.3%	-7.0%
CTC	-0.9%	-1.6%
SDSC	-1.6%	-10.9%

Message

- These are all heuristics.
- They are not specifically designed to optimize the metrics we have designed.
- It is difficult to truly understand the reasons for the results.
- But one can derive some empirical wisdom.
- One of the reasons why one is stuck with possibly obscure heuristics is that we're dealing with an *on-line* problem: We don't know what happens next.
- We cannot wait for all jobs to be submitted to make a decision. But we can wait for a while, accumulate jobs, and schedule them together.

Summary

Batch Schedulers are what we're stuck with at the moment.
They are often hated by users.

- I submit to the queue asking for 10 nodes for 1 hour.
- I wait for two days.
- My code finally starts, but doesn't finish within 1 hour and gets killed!!

Summary

Batch Schedulers are what we're stuck with at the moment.
They are often hated by users.

- I submit to the queue asking for 10 nodes for 1 hour.
- I wait for two days.
- My code finally starts, but doesn't finish within 1 hour and gets killed!!

A lot of research, a few things happening "in the field".
When you go to a company that has clusters (like most of them), they typically have a job scheduler, so it's good to have some idea of what it is.

Summary

Batch Schedulers are what we're stuck with at the moment.
They are often hated by users.

- I submit to the queue asking for 10 nodes for 1 hour.
- I wait for two days.
- My code finally starts, but doesn't finish within 1 hour and gets killed!!

A lot of research, a few things happening "in the field".
When you go to a company that has clusters (like most of them), they typically have a job scheduler, so it's good to have some idea of what it is.
A completely different approach is gang scheduling, which we discuss next.

Outline

Outline

Gang Scheduling: Basis

- All processes belonging to a job run at the same time (the term gang denotes all processors within a job).
- Each process runs alone on each processor.
- BUT: there is rapid coordinated context switching.
- It is possible to suspend/preempt jobs arbitrarily

Gang Scheduling: Basis

- All processes belonging to a job run at the same time (the term gang denotes all processors within a job).
- Each process runs alone on each processor.
- BUT: there is rapid coordinated context switching.
- It is possible to suspend/preempt jobs arbitrarily $\rightsquigarrow$ May allow more flexibility to optimize some metrics.

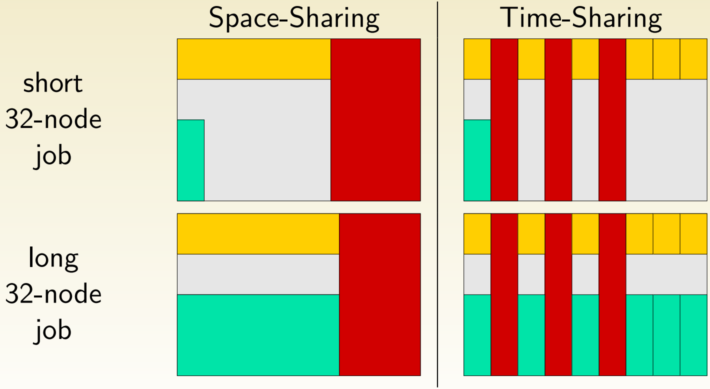
Gang Scheduling: Basis

- All processes belonging to a job run at the same time (the term gang denotes all processors within a job).
- Each process runs alone on each processor.
- BUT: there is rapid coordinated context switching.
- It is possible to suspend/preempt jobs arbitrarily $\rightsquigarrow$ May allow more flexibility to optimize some metrics.
- If processing times are not known in advance (or grossly erroneous), preemption can help short jobs that would be "stuck" behind a long job.
- Should improve machine utilization.

Gang Scheduling: an Example

- A 128 node cluster.
- A running 64-node job.
- A 32-node job and a 128-node job are queued.

Should the 32-node job be started ?



More uniform slowdown, better resource usage.

Outline

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).
- Overhead for coordinating context switching across multiple processors.

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).
- Overhead for coordinating context switching across multiple processors.
- Reduced cache efficiency(Frequent cache flushing).

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).
- Overhead for coordinating context switching across multiple processors.
- Reduced cache efficiency(Frequent cache flushing).
- RAM Pressure (more jobs must fit in memory, swapping to disk causes unacceptable overhead).

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).
- Overhead for coordinating context switching across multiple processors.
- Reduced cache efficiency(Frequent cache flushing).
- RAM Pressure (more jobs must fit in memory, swapping to disk causes unacceptable overhead).
- Typically not used in production HPC systems (batch scheduling is preferred).

Gang Scheduling: Drawbacks

- Overhead for context switching (trade-off between overhead and fine grain).
- Overhead for coordinating context switching across multiple processors.
- Reduced cache efficiency(Frequent cache flushing).
- RAM Pressure (more jobs must fit in memory, swapping to disk causes unacceptable overhead).
- Typically not used in production HPC systems (batch scheduling is preferred).
- Some implementations (MOSIX, Kerighed).

Outline

Batch Scheduling it is then

So it seems we're stuck with batch scheduling.
Why don't we like Batch Scheduling?

Batch Scheduling it is then

So it seems we're stuck with batch scheduling.
Why don't we like Batch Scheduling? Because queue waiting times are difficult to predict.

- depends on the status of the queue
- depends on the scheduling algorithm used
- depends on all sorts of configuration parameters set by system administrator
- depends on future job completions!
- etc.

So I submit my job and then it's in limbo somewhere, which is eminently annoying to most users.

Batch Scheduling it is then

So it seems we're stuck with batch scheduling.
Why don't we like Batch Scheduling? Because queue waiting times are difficult to predict.

- depends on the status of the queue
- depends on the scheduling algorithm used
- depends on all sorts of configuration parameters set by system administrator
- depends on future job completions!
- etc.

So I submit my job and then it's in limbo somewhere, which is eminently annoying to most users.
That is why there is more and more demand for reservation support.
Users build (badly?) the schedule by themselves.

Outline

Batch Scheduling and Grids

Grids result from the collaboration of many Universities/Computing Centers.

Everyone runs its own Batch Scheduler that cannot be bypassed. How to decide where we should submit our jobs?

Batch Scheduling and Grids

Grids result from the collaboration of many Universities/Computing Centers.

Everyone runs its own Batch Scheduler that cannot be bypassed.

How to decide where we should submit our jobs?

When in doubt, a brute-force approach is to:

- Do multiple submissions for different numbers of nodes
- Cancel all submissions but the first one that comes back
- Or possibly make some ad-hoc call regarding whether to keep a potentially poor request in the hope of getting a better one through shortly after.

Batch Scheduling and Grids

Grids result from the collaboration of many Universities/Computing Centers.

Everyone runs its own Batch Scheduler that cannot be bypassed. How to decide where we should submit our jobs?

When in doubt, a brute-force approach is to:

- Do multiple submissions for different numbers of nodes
- Cancel all submissions but the first one that comes back
- Or possibly make some ad-hoc call regarding whether to keep a potentially poor request in the hope of getting a better one through shortly after.

What happens if everybody does this?

Batch Scheduling and Grids

Grids result from the collaboration of many Universities/Computing Centers.

Everyone runs its own Batch Scheduler that cannot be bypassed. How to decide where we should submit our jobs?

When in doubt, a brute-force approach is to:

- Do multiple submissions for different numbers of nodes
- Cancel all submissions but the first one that comes back
- Or possibly make some ad-hoc call regarding whether to keep a potentially poor request in the hope of getting a better one through shortly after.

What happens if everybody does this?

Other issues:

- File Staging ?
- Load Balancing between sites ?

A set unrelated processors $P_1, \ldots, P_n$ and a set of sequential jobs $J_1, \ldots, J_n$ (processing time $p_{i,j}$).

Let's try a few natural scheduling strategies. We denote by a_i the time at which P_i is available (at the beginning $a_i = 0$ for all P_i):

Sequential Job Scheduling for Grids

A set unrelated processors $P_1, \dots, P_n$ and a set of sequential jobs $J_1, \dots, J_n$ (processing time $p_{i,j}$).
Let's try a few natural scheduling strategies. We denote by a_i the time at which P_i is available (at the beginning $a_i = 0$ for all P_i):

Min-Min Compute the minimum completion time $C_j = a_i + p_{i,j}$ of each J_j and choose the one with the smallest C_j. Update the corresponding a_i (its best host) accordingly ($a_i \leftarrow a_i + p_{i,j}$).

A set unrelated processors $P_1, \dots, P_n$ and a set of sequential jobs $J_1, \dots, J_n$ (processing time $p_{i,j}$).
Let's try a few natural scheduling strategies. We denote by a_i the time at which P_i is available (at the beginning $a_i = 0$ for all P_i):

Min-Min
: Compute the minimum completion time $C_j = a_i + p_{i,j}$ of each J_j and choose the one with the smallest C_j. Update the corresponding a_i (its best host) accordingly ($a_i \leftarrow a_i + p_{i,j}$).

Max-Min
: Choose J_j with the largest C_j and update the corresponding a_i (its best host) accordingly.

Sequential Job Scheduling for Grids

A set unrelated processors $P_1, \ldots, P_n$ and a set of sequential jobs $J_1, \ldots, J_n$ (processing time $p_{i,j}$).

Let's try a few natural scheduling strategies. We denote by a_i the time at which P_i is available (at the beginning $a_i = 0$ for all P_i):

Min-Min
: Compute the minimum completion time $C_j = a_i + p_{i,j}$ of each J_j and choose the one with the smallest C_j. Update the corresponding a_i (its best host) accordingly ($a_i \leftarrow a_i + p_{i,j}$).

Max-Min
: Choose J_j with the largest C_j and update the corresponding a_i (its best host) accordingly.

Sufferage
: S_j is the difference between the best completion time of J_j and its second best completion time. Choose the job with the largest sufferage and schedule it on its best processor.

Sequential Job Scheduling for Grids

A set unrelated processors $P_1, \dots, P_n$ and a set of sequential jobs $J_1, \dots, J_n$ (processing time $p_{i,j}$).

Let's try a few natural scheduling strategies. We denote by a_i the time at which P_i is available (at the beginning $a_i = 0$ for all P_i):

Min-Min Compute the minimum completion time $C_j = a_i + p_{i,j}$ of each J_j and choose the one with the smallest C_j. Update the corresponding a_i (its best host) accordingly ($a_i \leftarrow a_i + p_{i,j}$).

Max-Min Choose J_j with the largest C_j and update the corresponding a_i (its best host) accordingly.

Sufferage S_j is the difference between the best completion time of J_j and its second best completion time. Choose the job with the largest sufferage and schedule it on its best processor.

Problem: How do you get an estimate of $p_{i,j}$?

So Where are we ?

- Batch schedulers are complex pieces of software that are used in practice.
- A lot of experience on how they work and how to use them.
- But ultimately everybody knows they are an imperfect solution.
- Many view the lack of theoretical foundations as a big problem.
- Some just don't care. . .

> *Fools ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.*
>
> – *"Epigrams in Programming", by Alan J. Perlis of Yale University.*

Bibliography

A.J. Bernstein.
Analysis of programs for parallel processing.
IEEE Transactions on Electronic Computers, 15:757–762, October 1966.

Peter Brucker.
Scheduling Algorithms.
Springer, Heidelberg, 2 edition, 1998.

E. G. Coffman.
Computer and job-shop scheduling theory.
John Wiley & Sons, 1976.

D.B. Shmoys, J. Wein, and D.P. Williamson.
Scheduling parallel machines on-line.
Symposium on Foundations of Computer Science, 0:131–140, 1991.