IACS INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE AT HARVARD UNIVERSITY

HARVARD School of Engineering and Applied Sciences

Solutions HW: A. Parallel Processing Fundamentals

1. Introduction to Parallel Thinking

1.1. Revenge on the Campus Store

1.

256 seconds (or 255 if the initialization of bag 1 does not take time).

2.

They can divide the number of bags among them and count them, this leads to: (256/8)-1 = 31 seconds,

They then have to combine their numbers, which can be done in 3 steps (8 numbers to 4 numbers to 2 numbers to 1 number) so: (256/8)-1 +3 = 34s.

3.

The total time (T) it takes to count 256 bags with E employees can be expressed as:

T = ceiling(256/E)-1 +ceiling($\log_2$(E))

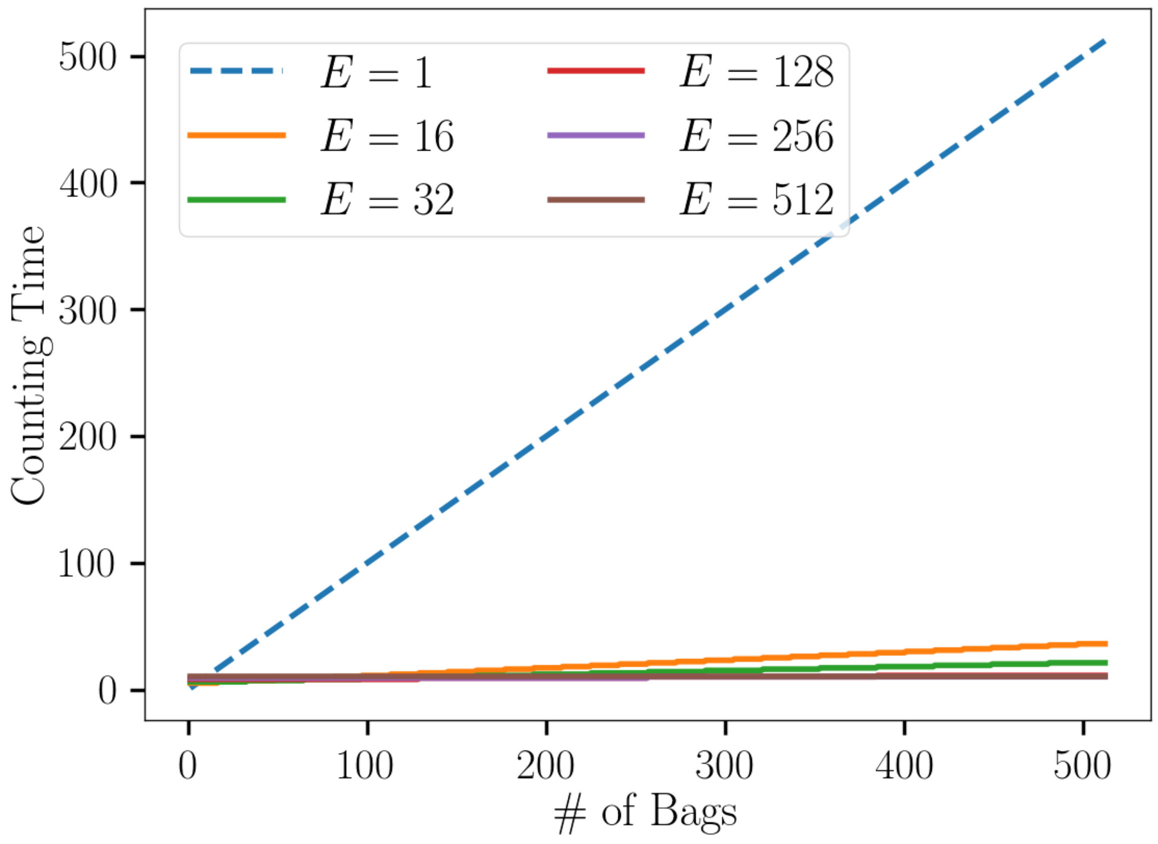
Because of the ceiling functions, this cannot be solved algebraically. Trying integers 1-256 for E in the equation for T gives a minimum time of T = 8. The lowest number of employees to realize this time with is 128.

4.

The time it takes for N bags is given by: T = ceiling(N/E)-1 +ceiling($\log_2$E)

Which means the optimal number of employees is given by: E = N/2 and the optimal time is given by T = 1+ceiling($\log_2$(E)). Meanwhile, for the lone employee, T = N-1.

PLOT



5.

If communication takes 1s then the time equation changes to: T = ceiling(256/E)-1 +2 ceiling($\log_2$E)

Numerically solving this problem shows that the minimum time spent is 15s, which is reached with 128 employees.

6.

Handing over the bags at the start of the process would add E interactions. What makes this problematic is that part of the counting could already start while the handing over of bags is still taking place. Here, it is assumed this is not the case.

If one bag is transferred from the student to the cashiers at one time, it would simply add 256s, giving a total of 8+256 = 264s. If two bags were taken at once, it would be 8+128 = 136s. The number of bags given to an employee can be set equal to the number of bags they have to count, and as such be included in the optimization, giving the time equation: T = E + ceiling(256/E) -1+2*ceiling($\log_2$E)

1.2. Graph Parallelization

1. Maximum degree of concurrency => the maximum number of tasks that can be executed in parallel

a. 8

b. 8

c. 8

d. 8

2. Critical path length => The longest directed path between any pair of start and finish nodes

a. 4

b. 4

c. 7

d. 8

3. The minimum number of processors needed to obtain the maximum speed-up

a. Work: 15 => Max Speed-up: 15/4 => Min. Num. Proc.: 8

b. Work: 15 => Max Speed-up: 15/4 => Min. Num. Proc.: 8

c. Work: 14 => Max Speed-up: 14/7 => Min. Num. Proc.: 3

d. Work: 15 => Max Speed-up: 15/8 => Min. Num. Proc.: 2

4. The maximum achievable speed-up if the number of processors is limited to (i) 2 and (ii) 5

a. I: 15/8 ii: 15/5

b. i: 15/8 ii: 15/5

c. i: 14/8 ii: 14/7

d. i: 15/8 ii: 15/8

2. Parallel Processing Architectures

2.1. Basic Concepts

1.

Incorrect: The number of transistors continued doubling approximately every two years (slowing of Moore's Law started later, in 2012 approx.). However because single processor speed is limited by memory latency, instruction level parallelism and an exponential growth of power consumption, the extra transistors started to be used to offer more cores.

2.

Correct: Basically it means that cooling is becoming prohibitively expensive

3.

Incorrect. Having new programming models for parallel computing does not mean that any code can be parallelized. Many algorithms are sequential and we should use alternative algorithms with more parallelism level that solve the same problem

4.

Incorrect. Multi-core CPU is something relatively new, but there have been shared-memory parallel systems since the 70s.

2.2. Superscalar Processors

Modern CPUs contain multiple cores and each core has multiple copies of the same physical resources, such as execution units, issue control or data paths. This allows the execution of more than 1 instruction per core and CPU cycle. For example, the new Intel(R) Core(TM) i9-7900X CPU contains 10 cores, where each core allows the execution of 2 simultaneous threads. Its CPU Core Pipeline functionality is based on the Skylake Microarchitecture with a 6-wide superscalar design per thread.

1.

In each cycle: 6 operations/thread * 2 threads/core * 10 cores/cpu => 120 operations/cpu; * 3.3 GHz => 396 Gflops

2.

Matrix multiplication flops is 2 n^3 => 2 10^{12} flops => on 396 10^9 flops/second => 5 seconds

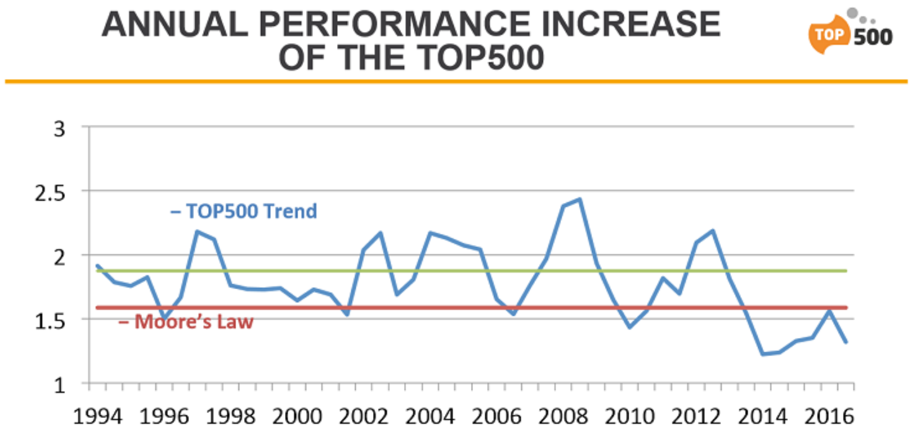
It could be more accurate (n^2)(2n - 1), or n^2.807 as given by wikipedia

3.

Bandwidth demand: In each cycle 120 instructions/cpu => 12 store instructions/cpu => 96 bytes /cpu => 316.8 GB/sec => 2 CPUs!

2.3. Top500 Showing Signs of Stagnation

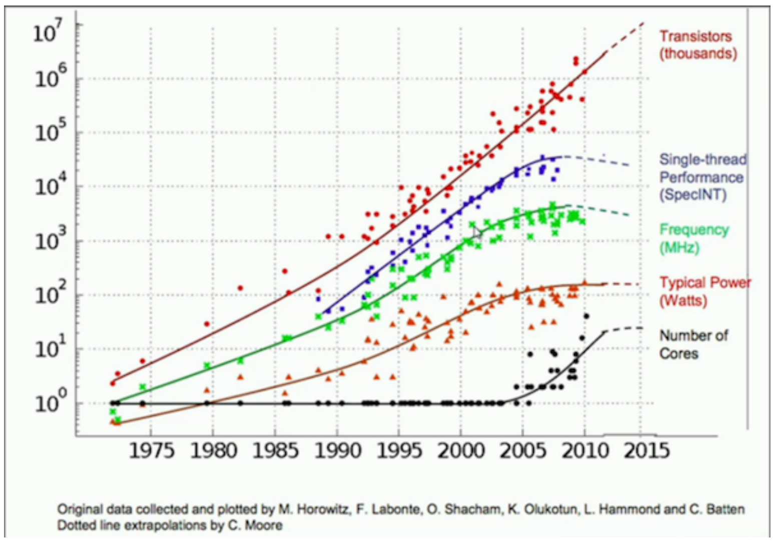
1. Students should provide a graph built by themselves



2.

Slowing of Moore's Law and reduction of investment (financial crisis).

3.



3. Parallel Processing on the Cloud

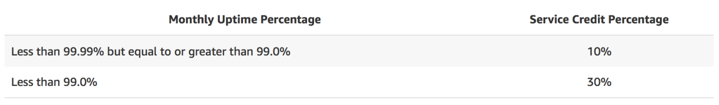
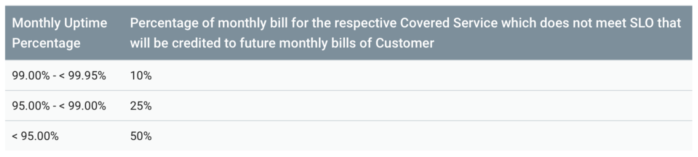
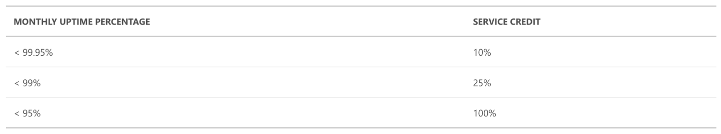
3.1. Public Cloud Pricing and SLAs

You can go to the next article for a complete **pricing comparison** (some data are outdated)

https://www.rightscale.com/blog/cloud-cost-analysis/comparing-cloud-instance-pricing-aws-vs-azure-vs-google-vs-ibm

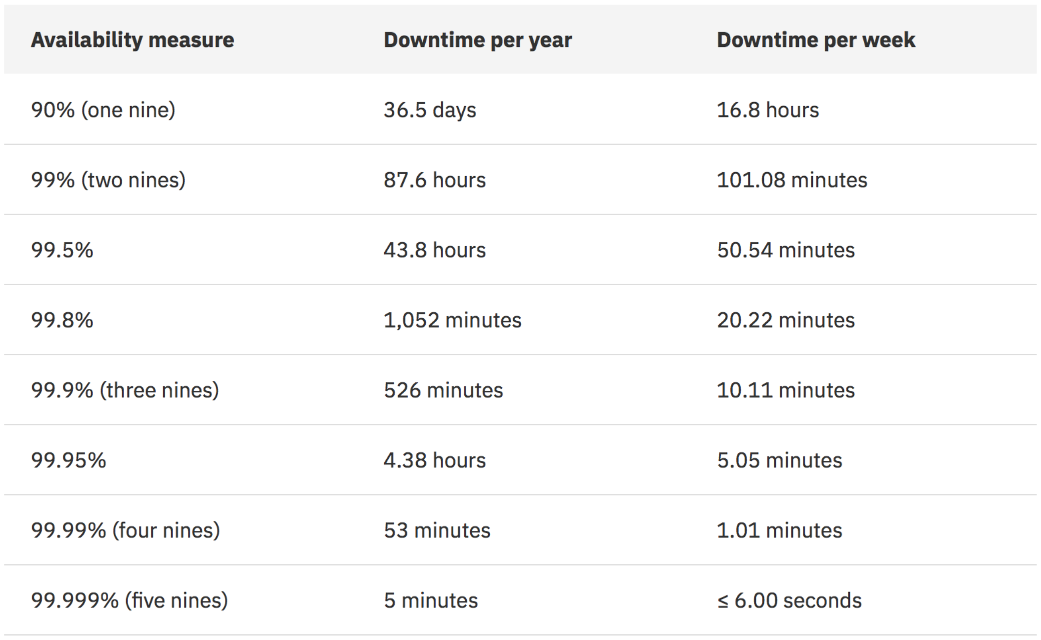
About **SLAs**

https://aws.amazon.com/ec2/sla/
https://cloud.google.com/compute/sla
https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_6/
http://www.softlayer.com/sites/default/files/softlayer_service_level_agreement_-_standard_services_final_may_2016.pdf

Provider	**Price ($/hour)**	**SLA**	**Penalty (Service Credit)**
Amazon AWS	m3.large 0.133	99.99%	 **Monthly Uptime Percentage** / **Service Credit Percentage** Less than 99.99% but equal to or greater than 99.0% — 10% Less than 99.0% — 30%
Google Compute	n1.standard-2 0.095	99.95%	 Monthly Uptime Percentage / Percentage of monthly bill for the respective Covered Service which does not meet SLO that will be credited to future monthly bills of Customer 99.00% - < 99.95% — 10% 95.00% - < 99.00% — 25% < 95.00% — 50%
Microsoft Azure	D2 v3 0.096	99.99%[1]	 MONTHLY UPTIME PERCENTAGE / SERVICE CREDIT < 99.95% — 10% < 99% — 25% < 95% — 100%
IBM Cloud	BL1.2x8.200 0.137	99.95%[2]	For each 30 continuous minute period of Qualifying Outage Minutes in a Measurement Period, SoftLayer will provide an SLA Credit in the amount of 5% of the charges for the Cloud Service which was subject to the Loss of Service. Any period of Qualifying Outage Minutes which is less than 30 continuous minutes will not be eligible for any SLA Credit. Claimed Outages for different Cloud Services may not be combined to meet this calculation. SLA Credits for failure to meet specified response time for hardware replacement or hardware upgrade are set forth in Section 7.

The nines are a tempting target for setting availability goals. As shown in below table, the nines consist of having all numeral nines in a percentage that represents the time the system is available.

[1] https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_8/
[2] https://www-03.ibm.com/software/sla/sladb.nsf/pdf/6605-15/$file/i126-6605-15_01-2019_en_US.pdf



Availability measure	Downtime per year	Downtime per week
90% (one nine)	36.5 days	16.8 hours
99% (two nines)	87.6 hours	101.08 minutes
99.5%	43.8 hours	50.54 minutes
99.8%	1,052 minutes	20.22 minutes
99.9% (three nines)	526 minutes	10.11 minutes
99.95%	4.38 hours	5.05 minutes
99.99% (four nines)	53 minutes	1.01 minutes
99.999% (five nines)	5 minutes	≤ 6.00 seconds

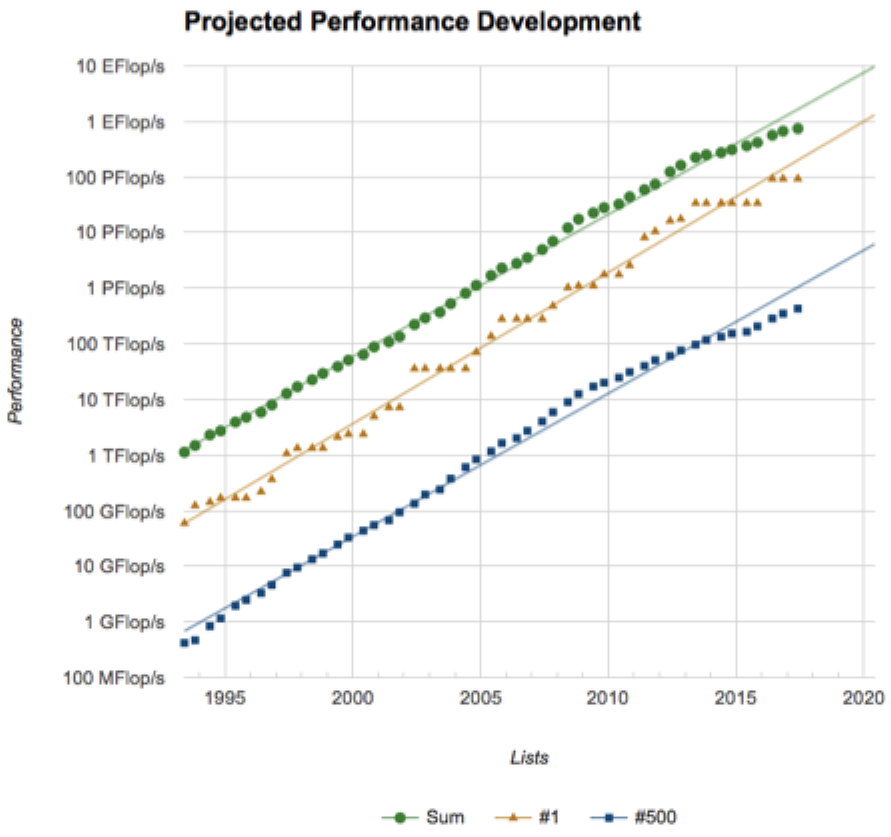
3.2. Linpack on a Single AWS Node

For installation see document (**this should be explained in the second lab session**)

https://docs.google.com/document/d/1txrg_1Gfl8_IwEqVKE0R2MuiQcdkYcjjNLGHURNGtqk/edit

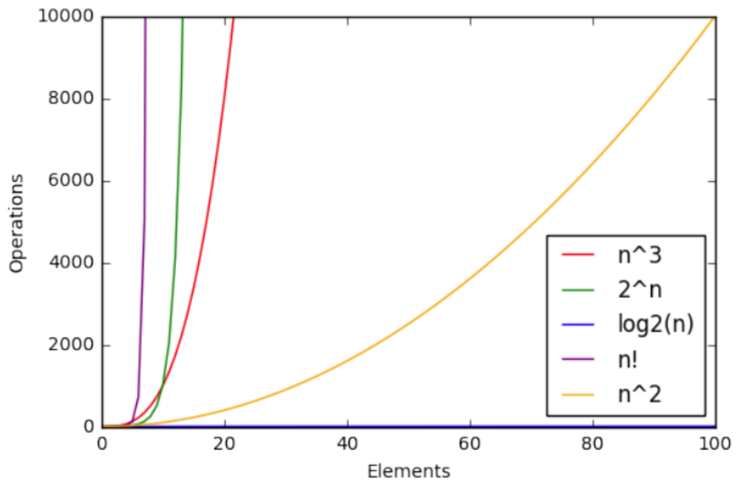
My results

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2C2       30000   256     2     2             459.46              3.918e+01
HPL_pdgesv() start time Wed Jan 16 04:38:13 2019

HPL_pdgesv() end time   Wed Jan 16 04:45:52 2019

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0012625 ...... PASSED
```



4. Designing Parallel Programs

4.1 Time Complexity Plots



4.2. Amdahl's Law

1.

0.6 = 1 / (0.08p+0.92) => 9 processing elements

2.

$S = (n^2\ t_{smooth})/((n^2/p)\ t_{smooth} + n\ t_{comm}) \Rightarrow E = (n\ t_{smooth})/(n\ t_{smooth} + p\ t_{comm}) \Rightarrow p < n\ (t_{smooth}/t_{comm})$

3.

Gustafson's law addresses the shortcomings of Amdahl's law, which is based on the assumption of a fixed problem size, that is of an execution workload that does not change with respect to the improvement of the resources. Gustafson's law instead proposes that programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve. Therefore, if faster equipment is available, larger problems can be solved within the same time.

Both can be used to measure performance and scalability. The one to use depends on the type of scaling we want to measure, fixed problem size and reduce execution time, or fixed size per processor and keep execution time.