UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES

Computational Performance
Heterogeneous Computing

Professor: Dr. Joel Fuentes - jfuentes@ubiobio.cl

Teaching Assistants:
- Daniel López - daniel.lopez1701@alumnos.ubiobio.cl
- Sebastián González - sebastian.gonzalez1801@alumnos.ubiobio.cl

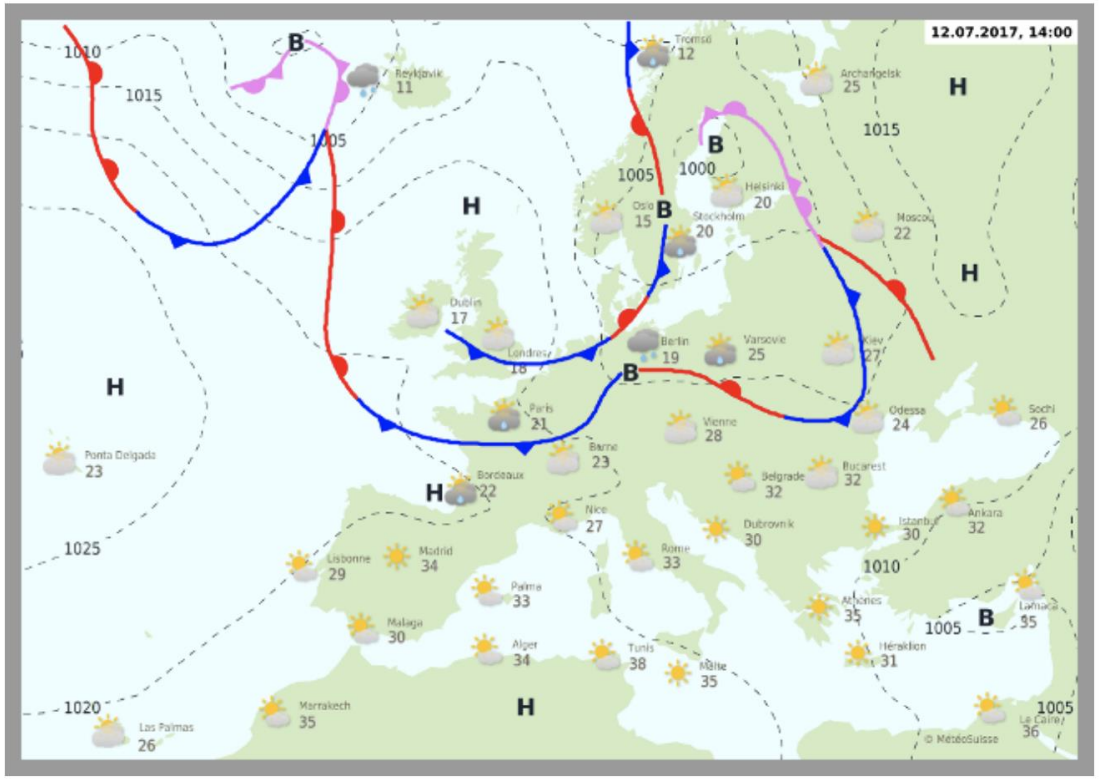
Course website: http://www.face.ubiobio.cl/~jfuentes/classes/ch

Content

- General concepts
- Performance and scaling
- Performance metrics
  - Amdahl's Law.
  - Gustafson-Barsis Law.
- Execution models
- DAG.

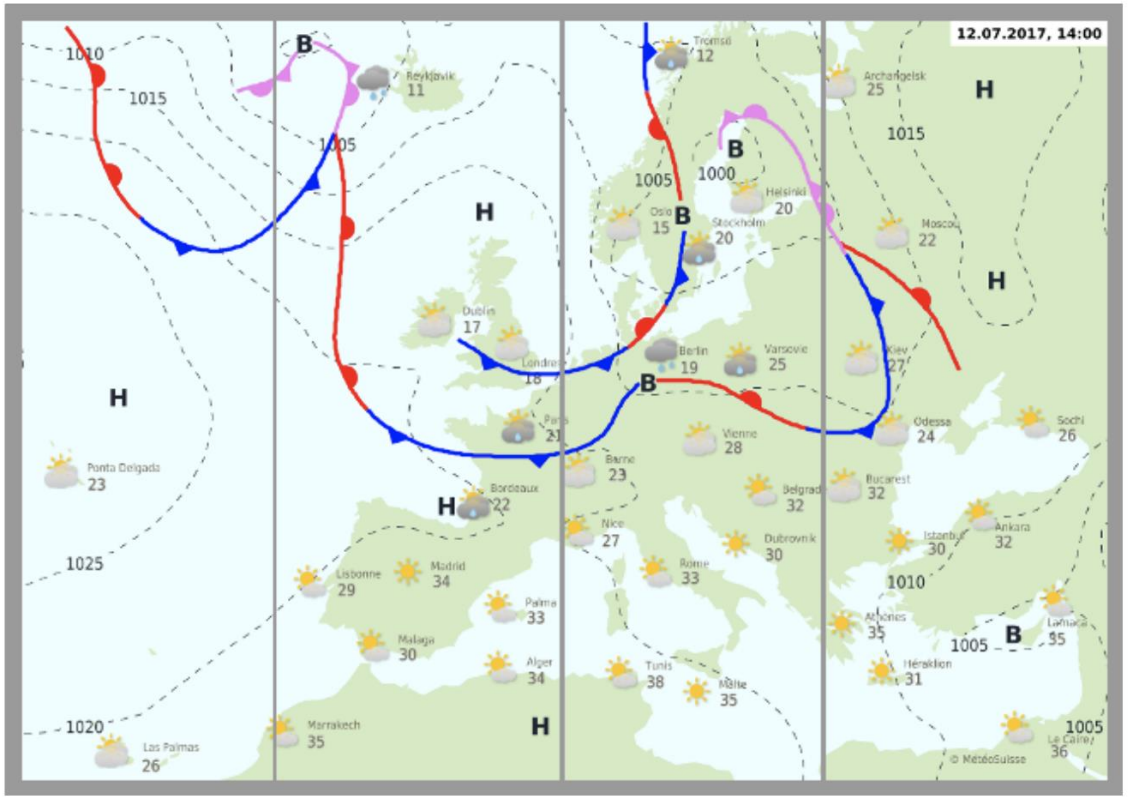
General concepts

- Sequential processing



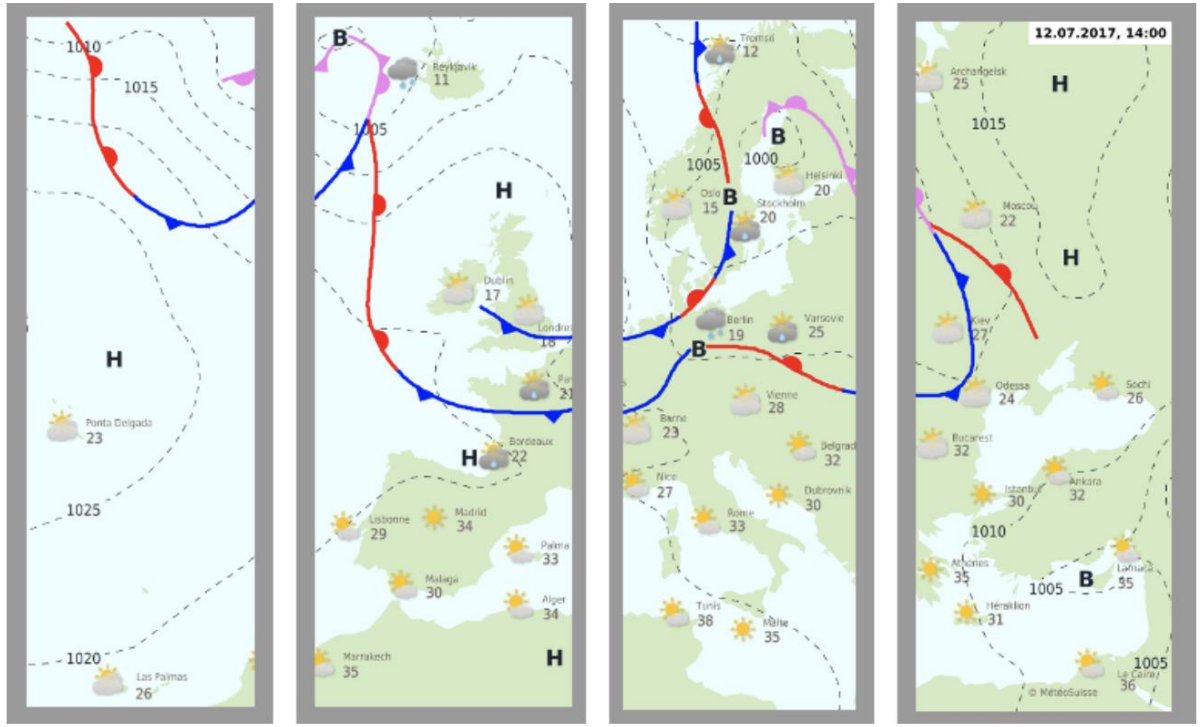
General concepts

- Parallel processing in shared memory



General concepts

- Parallel processing in distributed memory

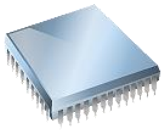


Performance and Scaling

What is Computational Performance?

- In computing, performance is defined by 2 factors:
  - Computational requirements (to be performed).
  - Computational resources (what is the cost of doing so).
- Computational problems translate into requirements.
- Computational resources act as tradeoffs.

$$Performance \sim \frac{1}{Solution\ Resources}$$



Hardware

Time

Energy

Money

What is Parallel Performance/Performance?

- We are interested in knowing performance problems when using parallel computing environments.
- Performance is the reason for parallelism:
  - If performance is not better, parallelism is not necessary.
- Parallel processing includes many techniques and technologies:
  - Hardware, networks, operating systems, libraries, programming languages, compilers, algorithms, tools, etc..
- Parallelism must deliver better performance
  - How? How much better?

Expected performance

- If each processor works at k MFLOPS and there are p processors, then will we have k*p MFLOPS performance?
- If a task takes 100 seconds on 1 processor, shouldn't it take 10 seconds on 10 processors?
- Many causes affect the performance of a parallel algorithm.
- It is necessary to understand all these causes.
- Solution to one problem might create another.
- Scaling is a desired feature in parallelism.

"Embarrassing" parallel computation

- Embarrassing parallel computation is one that can be obviously divided into independent parts that run simultaneously.
- In many it is not necessary for interaction between processors.
- In others, distribution of results between processors may be necessary.
- Algorithms with this type of parallel computing have the potential to achieve maximum acceleration on parallel platforms.
- If solving a sequential problem takes T time, T/P time could potentially be achieved with P processors.

Scaling

- An algorithm can scale to use many processors.
- How to evaluate scaling?
- Benchmarking:
  - ✓ If we double the number of processors, is it linear scaling?.
- The key is to apply performance metrics.

Performance Metrics

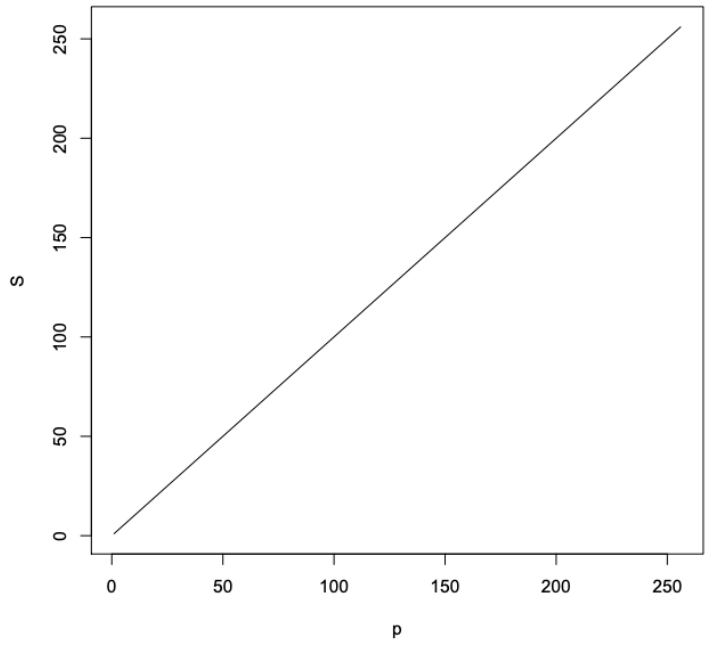
Performance metrics

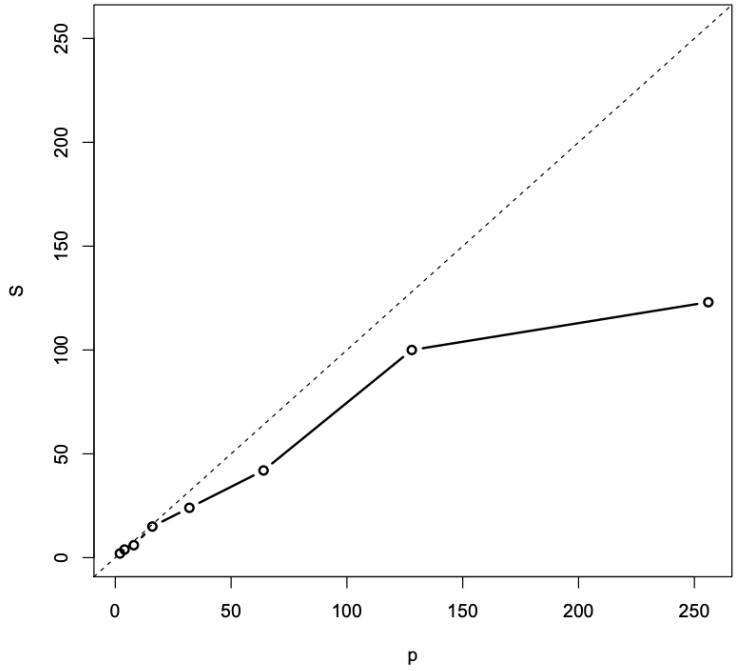
- Evaluation.
  - Sequential runtime (T_{sec}) is a function of:
    - The size of the problem and architecture.
  - Parallel runtime (T_{par}) is a function of:
    - The size of the problem and parallel architecture.
    - Number of processors used in execution.
  - Parallel performance is mainly affected by:
    - Algorithm + architecture.
- Scaling
  - It is the ability of a parallel algorithm to achieve performance improvements in proportion to the number of processors and the size of the problem..

Performance metrics

- T_1 is the runtime on a processor
- T_p is the runtime in p processors
- S_p is the acceleration
- $S_p = \frac{T_1}{T_p}$
- E_p is efficiency
- $E_p = \frac{S_p}{p}$
- C_p is the cost
- $C_p = p \times T_p$

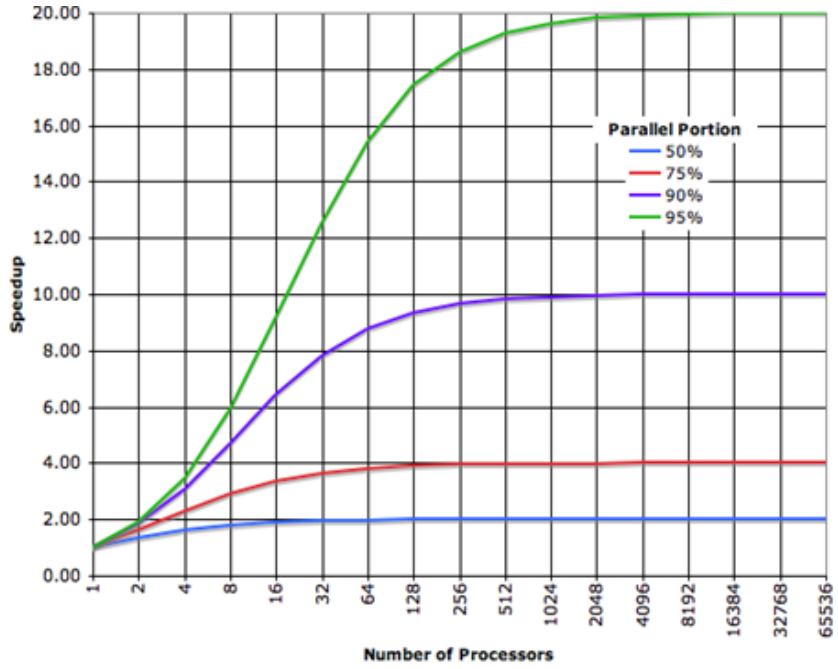
Ideal acceleration versus reality S_p



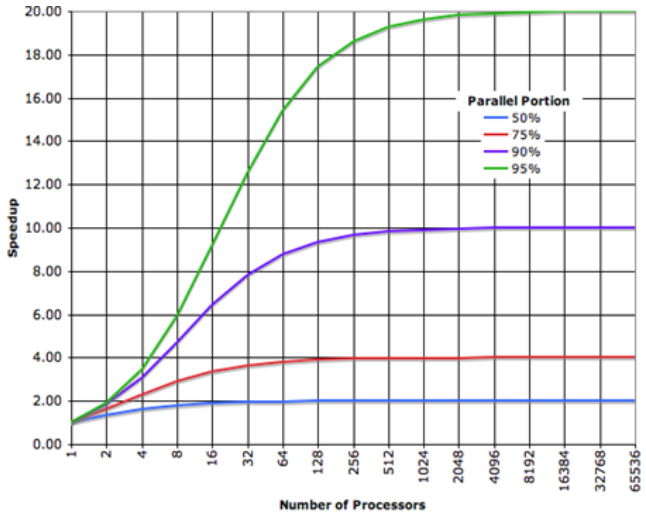


Performance Metrics: Amdahl's Law

- Let f be the fraction of a program that is sequential
- $1 - f$ is the fraction that can be parallelized
- T_1 is the runtime on a processor
- T_p is the runtime in p processors
- S_p is the acceleration
  - $S_p = T_1/T_p$
  - $= T_1/(fT_1 + \frac{(1-f)T_1}{p})$
  - $= 1/(f + \frac{(1-f)}{p})$
- Si $p \to \infty$
  - $S_\infty = 1/f$



Performance Metrics: Amdahl's Law



- When to apply Amdahl's Law?.
  - When the size of the problem is fixed.
  - Strong scaling ($p \to \infty$, $S_p = S_\infty \to 1/f$).
  - Acceleration limit is determined by the degree of sequential execution, not the number of processors!
  - is this good? Why?.
  - Perfect efficiency is very difficult to achieve.
- See Amdahl paper attached to the course platform.

Performance Metrics: Gustafson-Barsis Law

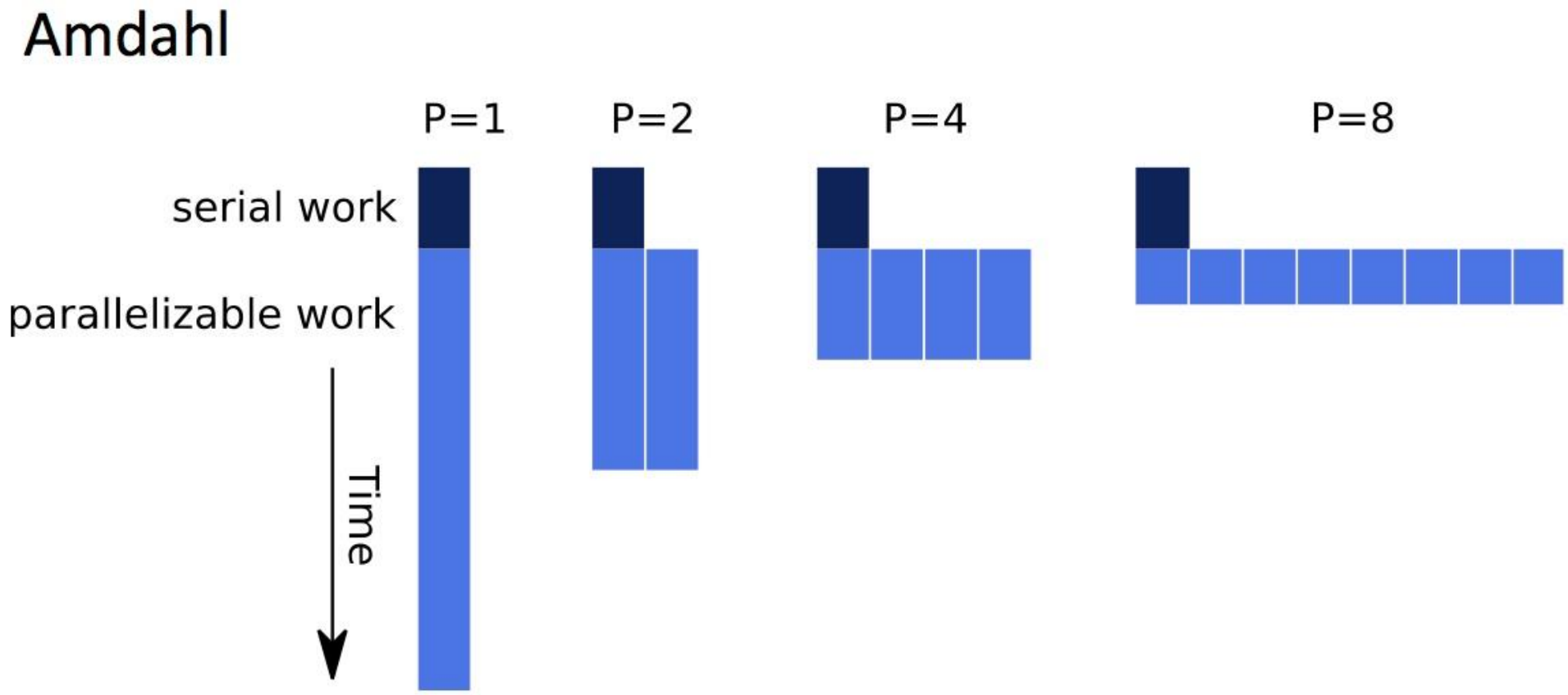
- Assume that parallel time is kept constant.
  - $T_p = C = \left(f + (1 - f)\right) * C$
  - f_{sec} is the fraction of Tp in sequential execution
  - f_{par} is the fraction of Tp running parallel.
  - What is the runtime on a processor?
  - Si $C = 1$, then
  - $T_s = f_{sec} + p(1 - f_{sec}) = 1 + (p - 1)f_{par}$.
- What is the acceleration in this case?
- $S_p = \frac{T_s}{T_p} = \frac{T_s}{1}$
  - $S_p = 1 + (p - 1)f_{par}$

Performance Metrics: Gustafson-Barsis Law

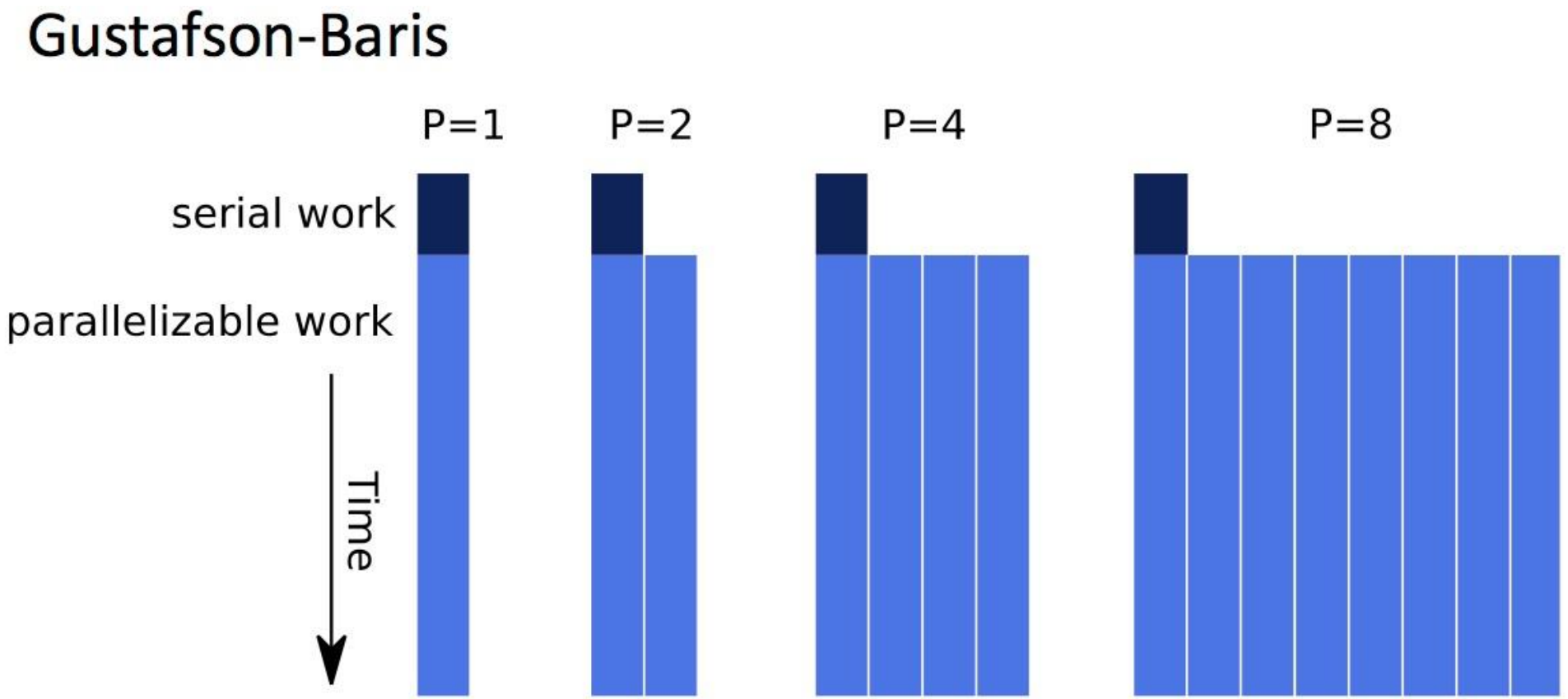
- When to apply the Gustafson-Barsis Law?.
  - When the size of the problem may grow while the number of processors also increases
  - Weak scaling ($S_p = 1 + (p-1)f_{par}$)
  - Acceleration function includes the number of processors!
  - Can maintain or increase parallel efficiency while the problem scales.

- See Gustafson-Barsis paper attached to the course platform.

Amdahl vs Gustafson-Barsis



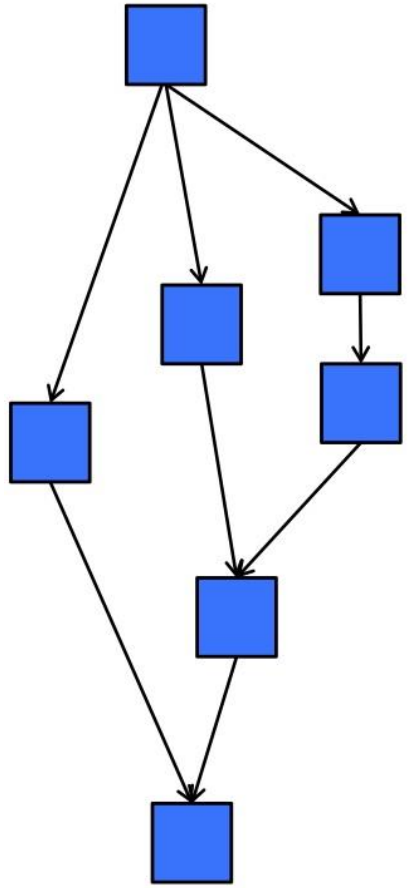
Amdahl vs Gustafson-Barsis



Execution Models

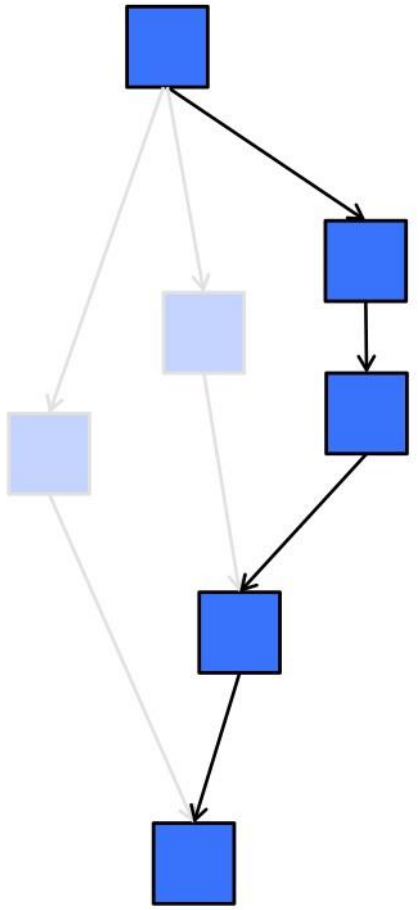
Execution models: DAG

- Assume a program as a directed acyclic graph (DAG) of tasks
  - A task cannot run until all its inputs are available
  - Inputs come from outpus of other previously executed tasks
  - DAG explicitly displays task dependency.
- Consider a "greedy" task scheduler to assign tasks to processors.
  - There should be no idle processors as long as there are tasks to run.



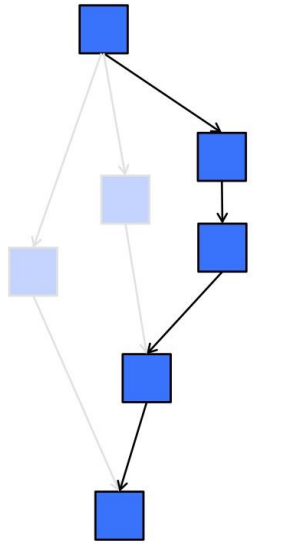
Execution models: DAG

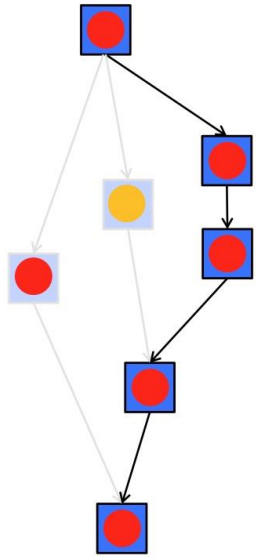
- Example:
  - Each task takes 1 unit of time
  - DAG has 7 tasks
  - $T_1 = 7$
    - All tasks must be executed
    - Tasks are executed in serial order
    - Can tasks be executed in any order?
  - $T_\infty = 5$
    - Time on the critical path
    - In this case, it is the longest path of tasks with linear dependencies.



Lower/Upper bound with "greedy" planning

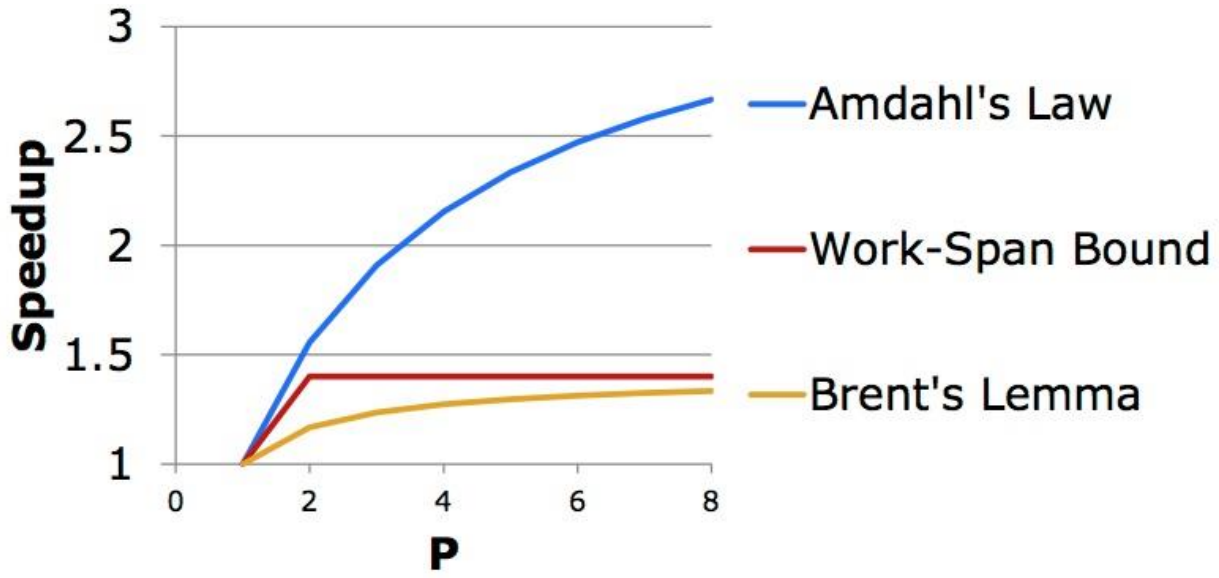
- Suppose there are only p processors
- It is possible to write a formula to reflect the lower bound of T_p.
  - $Max(\frac{T_1}{p}, T_\infty) \leq T_p$
- T_∞ is the best possible runtime
- Brent lemma for the upper bound
- Capture the additional cost of running other tasks that are not on the critical path
  - Assume that this can be done without large additional costs
  - $T_p \leq \frac{T_1 - T_\infty}{p} + T_\infty$

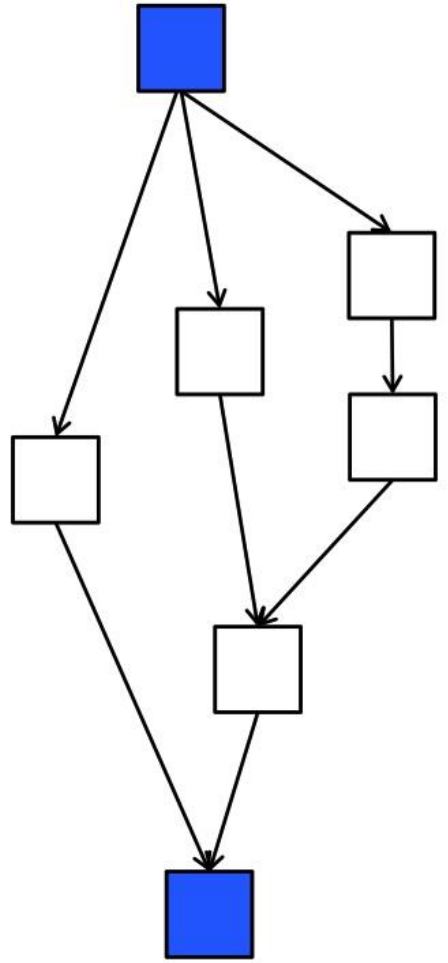




Lower/Upper bound with "greedy" planning

- Amdahl is optimistic
-





References

- Parallel Computing Center. University of Oregon http://ipcc.cs.uoregon.edu/index.html