10. Parallel performance

[1]Load balancing, speedup, and parallel efficiency

- **Parallelism is a cooperative work involving several tasks, with the main aim to complete a job/project**
- **The entire job cannot be completed until the slowest subtask is finished**
- **To this end, distributing/balancing the work load, that is, load balancing is important**
- **Let T_1 denote the (single core) time of execution of the best sequential solution of a scientific computing problem**
- **Suppose that the problem is divided into a set of P parallel tasks and is executed in a parallel computing environment and the ith task is executed in time $t_i, i = 1, \cdots, P$ on the environment**

[1] M. Ganesh, MATH440/540, SPRING 2018

- [2]It is reasonable to assume that

$$T_1 \leq \sum_{i=1}^{P} t_i$$

- The average execution time T_{ave} for this set of parallel tasks is

$$T_{ave} := \frac{1}{P} \sum_{i=1}^{P} t_i$$

- The parallel execution time T_P for the set is defined as

$$T_P := \mathbf{max}\{t_i : 1 \leq i \leq P\}$$

- The load balance β of this set of parallel tasks is

$$\beta := \frac{T_{ave}}{T_P}$$

- The ideal case of load balancing is $\beta = 1$.

[2] M. Ganesh, MATH440/540, SPRING 2018

- [3]The term

$$T_{rel} = \frac{T_P - T_{ave}}{T_P} = 1 - \beta$$

  measures the relative time difference between the longest task and the average task

- The set of tasks is said to be load balanced if $T_{rel} \approx 0$

- In practice, load balancing is achieved by balancing the amount of work to be done. This is because, usually, we can measure the amount of work to be done in advance as opposed to execution time in advance

- The main goal in parallel scientific computing is to create:

  - ⋆ largest possible embarrassingly parallel sections;
  - ⋆ largest possible granularity;
  - ⋆ smallest possible amount of nonlocal memory references;
  - ⋆ close to ideal load balancing among allocated cores

[3] M. Ganesh, MATH440/540, SPRING 2018

- **[4]It may not be possible to increase granularity and decrease communication simultaneously**
- **Increasing the granularity often is equivalent to decreasing the parallelism**
- **As the granularity gets smaller, the load balancing problem often becomes more difficult**
- **Thus the optimization problem for parallel computing can be quite complex and will require significant compromise**
- **A simple way to evaluate parallelization is to see if the time of execution is decreasing as a function of number of the number of cores**
- **A more critical way to evaluate the performance of parallelization is to consider speedup**

[4] M. Ganesh, MATH440/540, SPRING 2018

- [5]In HPC, performance is inversely proportional to time
- Hence the speedup of a parallel algorithm for a scientific computing problem, using P processors is:
  - ⋆ the ratio of parallel performance c/T_P divided by the sequential performance c/T_1, where
  - ⋆ T_P is the execution time for the parallel solution of the problem (that includes the full communication time to solve the problem)
  - ⋆ T_1 is time of execution of the best sequential solution to the problem, and
  - ⋆ c is the proportionality constant
- Thus, the speedup, S_P, using P processors is defined as the ratio of the sequential and parallel execution times

$$S_P = \frac{T_1}{T_P}$$

[5] M. Ganesh, MATH440/540, SPRING 2018

- [6]**If $S_P \approx P$, the speedup is refer to as linear (or perfect) speedup**
- **If $S_P > P$, the speedup is super-linear**
- **If $S_P < 1$, the speedup is called slowdown!**
- **For many large scale scientific computing problem, it is impossible to compute T_1, for example due to memory or time constraints**
- **In this case, speedup is defined with respect to the minimum number of processors, say Q, using which the problem can be solved. The relative speedup is defined as**

$$S_P^Q = \frac{T_Q}{T_P}$$

- **In such cases, the ideal speedup number is P/Q**

[6] M. Ganesh, MATH440/540, SPRING 2018

- [7]The parallel efficiency, E_P, of an algorithm using P cores is defined as the ratio of the speedup and P:

$$E_P = \frac{S_P}{P}$$

- Optimal parallel efficiency is $E_P = 1$. Super-optimal parallel efficiency is $E_p > 1$ and sub-optimal parallel efficiency is $E_p < 1$

- Use of E_P for cost analysis:

  - ⋆ Suppose that we need to pay \$$D$ per time unit (same as that used to measure T_1, T_P) for using each core
  - ⋆ The cost C_1 of using a single core to solve the scientific computing problem considered earlier is $C_1 =$
  - ⋆ The cost C_P of using P core to solve the problem is

$$C_P = \qquad\qquad = \qquad\qquad = \qquad\qquad =$$

[7] M. Ganesh, MATH440/540, SPRING 2018

- [8]**Hence purely from cost analysis point of view, even for optimal parallel efficiency algorithm, serial computing is much better!, especially considering the cost of setting up and maintaining multi-node clusters. (Note: $E_p = 1$ is the best we can expect in most cases.)**
- **However, it is important to note that the main purpose is the speedup (solving in hours/days rather than months/years) and the ability to simulate complex computer models**
- **As we discussed before, there are several factors that limit the parallel performance in HPC, leading to $E_P < 1$**
- **Is it possible to quantify such limitations, at least in relation to serial parts of algorithms/codes?**
- **There are several situations where some parts of an algorithm need to be sequential/serial. For example,**
  - ⋆ **to simulate space-time models, using time-stepping algorithm (where spatial solution at time t_n should be known before computing spatial part of the solution at time t_{n+1}, for $n = 0, 1, 2, \cdots$)**
  - ⋆ **or a programmer (or a group) unable to write efficient parallel codes to implement a fully parallel algorithm or ....**

[8] M. Ganesh, MATH440/540, SPRING 2018

[9]Amdahl's Law : Theoretical maximum speedup

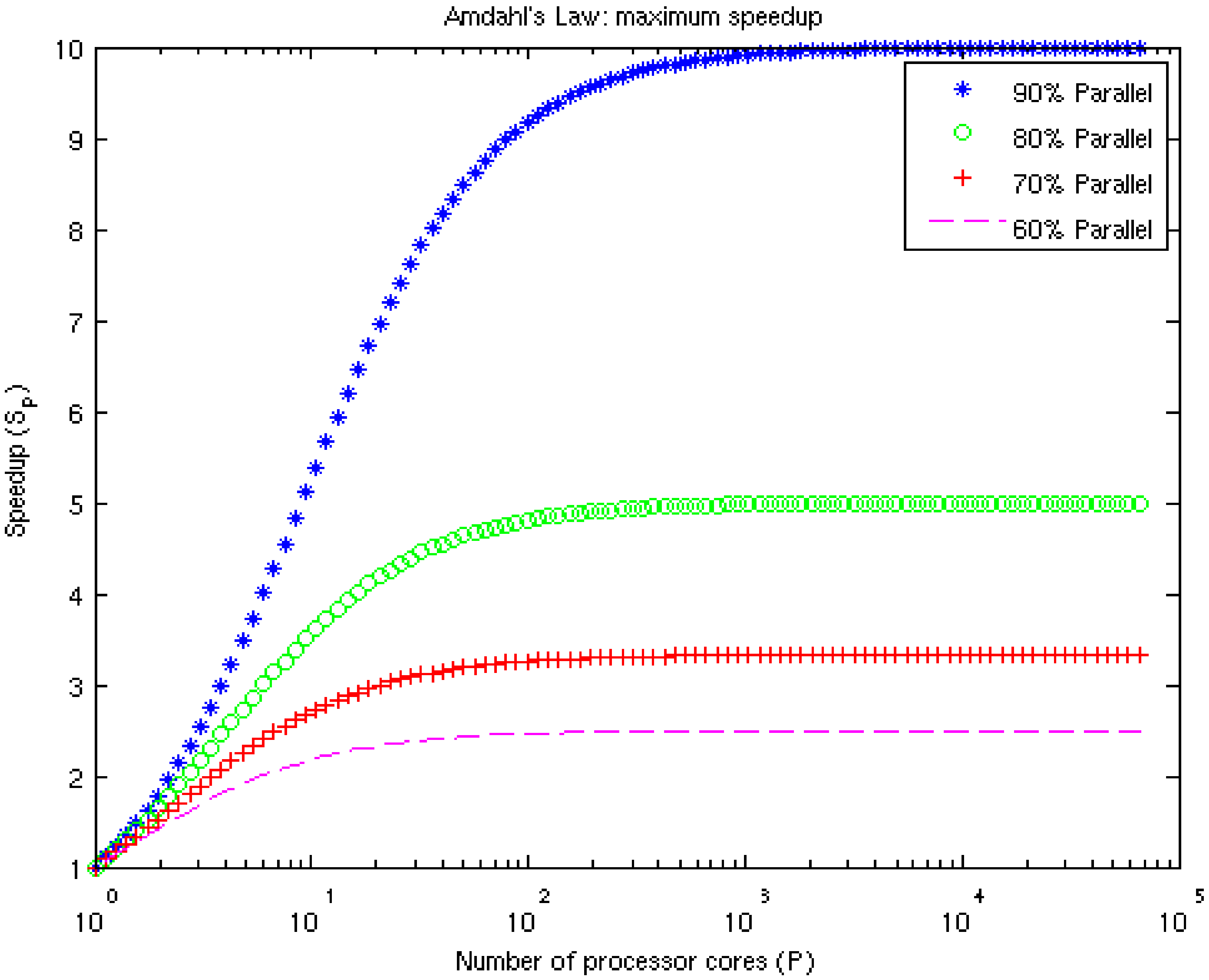
- **Suppose that a parallel code takes time T_1 to execute on one core**
- **Suppose that the sequential fraction of the code is f**
- **Suppose that the sequential part of the code requires time fT_1 to compute, independent of the number, say P, of processors used to execute the code**
- **Suppose that the reminder (that is, $1-f$ part) of the code requires *at least* time $(1-f)T_1/P$ to execute on P processor cores**
- $T_P = T_{COMP} + T_{COMM} = fT_1 + (1-f)T_1/P + T_{COMM}$, **with** $T_{COMM} > 0$
- **Exercise : Prove Amdahl's Law for maximum theoretical speedup for a code (with f of its parts sequential) using P cores**

$$S_P \leq \frac{1}{f + (1-f)/P}$$

[9] M. Ganesh, MATH440/540, SPRING 2018

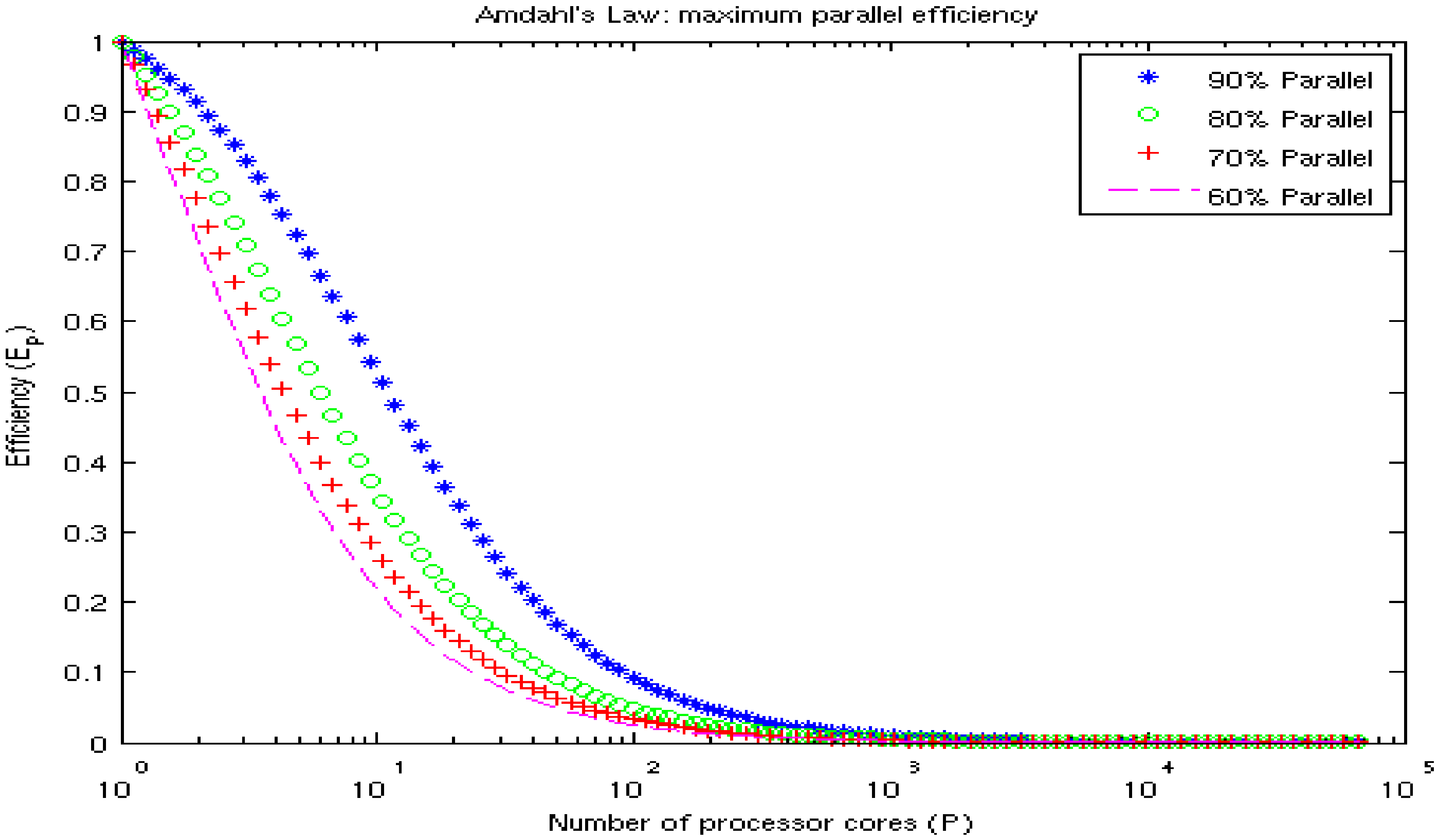
- **Exercise: What is the maximum speedup of a code with 5% of its part sequential, executed on a 1000 processor cores? (What if it ran on 100 cores? What if the 5% is reduced to 1%? Conclusions?)**

- **Solution:**


Amdahl's Law: maximum speedup
Speedup (S_p)
10
9
8
7
6
5
4
3
2
1
10^0
10^1
10^2
10^3
10^4
10^5
Number of processor cores (P)
✱ 90% Parallel
○ 80% Parallel
+ 70% Parallel
– – 60% Parallel

- [10]**Since parallel efficiency** $E_P = S_P/P$, **Amdahl's Law implies that**

$$E_P \leq 1/[1 + (P-1)f], \qquad E_P \approx 1 - (P-1)f, \quad \textbf{if } (P-1)f \quad \textbf{is small}$$



[10] M. Ganesh, MATH440/540, SPRING 2018

- [11]**Load balancing also plays an important role in efficiency**
- **Recall that the load balance β, using P parallel tasks (or cores) with t_i the execution time for task (core) i is**

$$\beta = \frac{T_{ave}}{T_P} = \frac{\frac{1}{P}\sum_{i=1}^{P} t_i}{T_P} \geq \frac{\frac{1}{P}T_1}{T_P} = \frac{1}{P}S_P = E_P$$

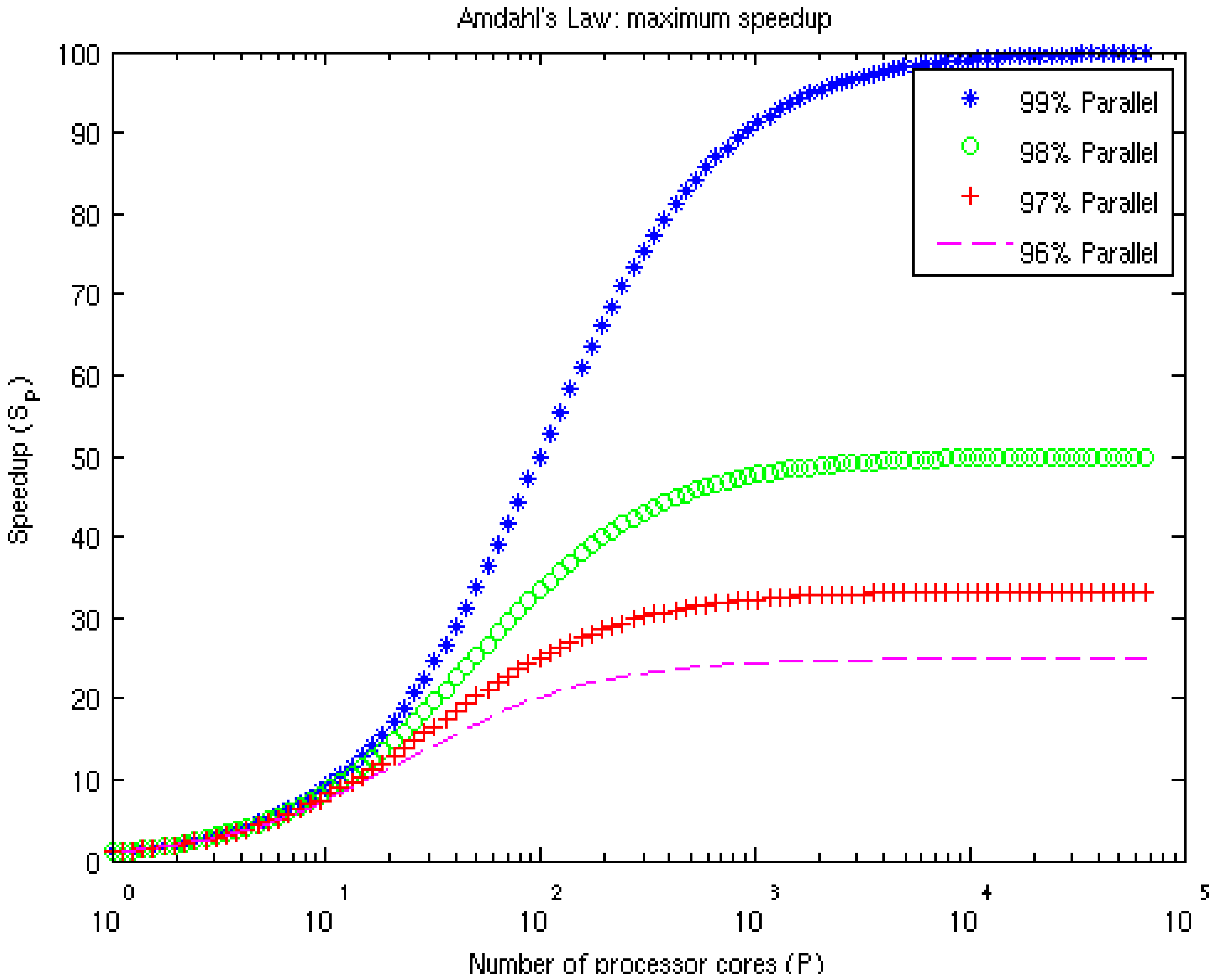
- **Thus efficiency of the calculation can never exceed the load balance**
- **Bad load balance leads to lack of synchronization. This limits the parallel performance**
- **In some cases, purely parallel computations (such embarrassingly parallel tasks) may not even be initiated due to idle wait time required due to bad load balance**

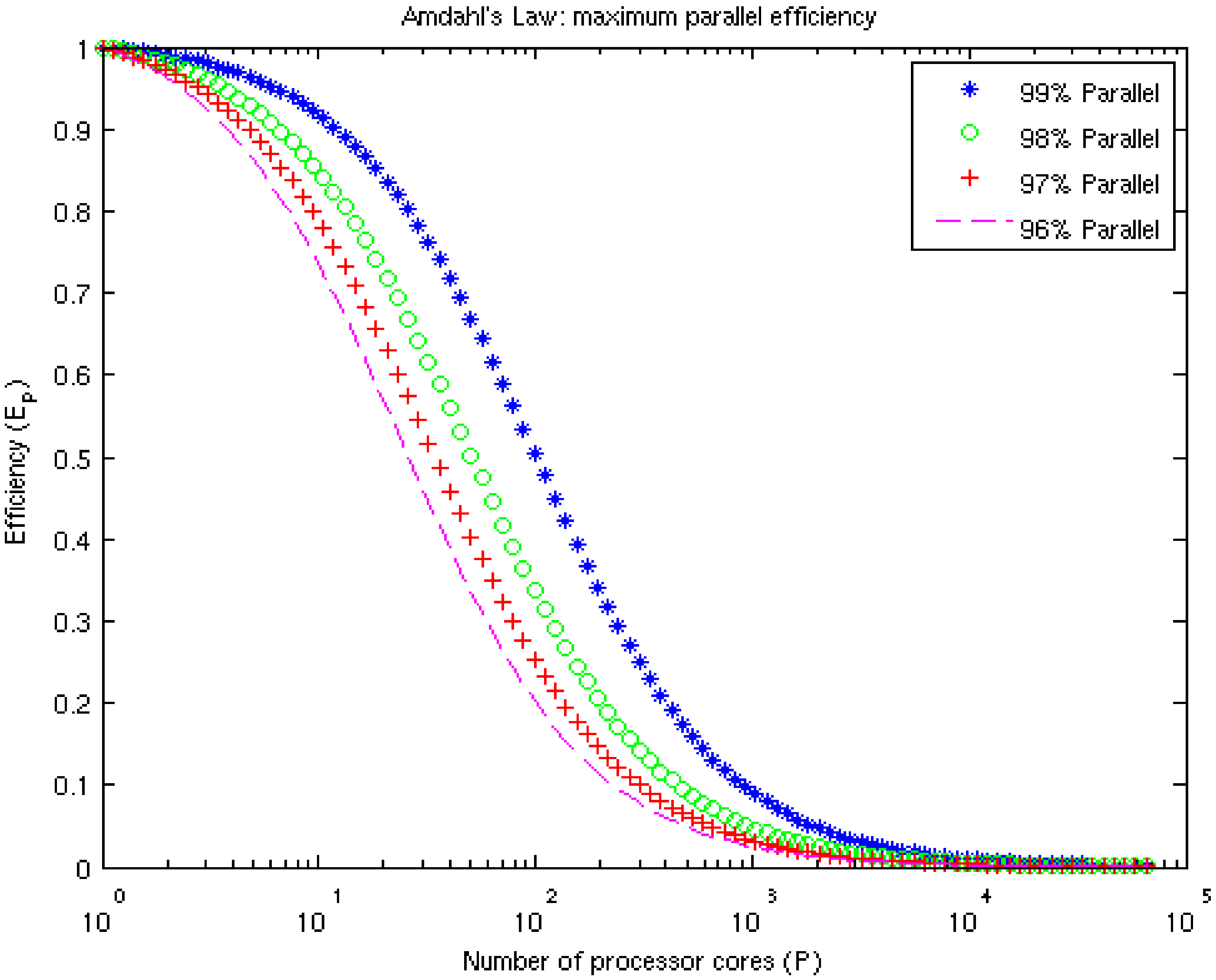
[11] M. Ganesh, MATH440/540, SPRING 2018

[12]Scalability

- **Consider Amdahl's Law based estimate for performance of a very highly parallel code (i.e., a code with less than 5% sequential fraction)**
- **After viewing the following two figures for the highly efficient parallel code, comment if the following concerns/questions are valid for an application oriented efficient parallel code:**
  - ⋆ **Is it possible to achieve speedup of 200?**
  - ⋆ **Is it possible to keep parallel efficiency positive as $P \to \infty$?**
- **In fact Alan Karp asked the speedup of 200 question in 1985. (Karp offered $\$100$ price to achieve this speedup within a decade.)**
- **This led to more complex challenge and Gordon Bell Award for progress in parallel scientific computing. (Price money $\$10,000$ for achievement in scalability, time, and price/performance.)**
- **The research paper by Gustafson et al., published in *SIAM J. Sci. Stat. Comp. Vol. 9, 609-638, 1988*, received both the Karp price and the Gordon Bell Award in 1988**

[12] M. Ganesh, MATH440/540, SPRING 2018


Amdahl's Law: maximum speedup
Speedup (S_P)
0
10
20
30
40
50
60
70
80
90
100
10^0
10^1
10^2
10^3
10^4
10^5
Number of processor cores (P)
99% Parallel
98% Parallel
97% Parallel
96% Parallel


Amdahl's Law: maximum parallel efficiency
Efficiency (E_p)
Number of processor cores (P)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10^0
10^1
10^2
10^3
10^4
10^5
99% Parallel
98% Parallel
97% Parallel
96% Parallel

- [13]**Alternative form of Amdahl's Law: Gustafson-Barsis's Law**

- **Amdahl's Law based speedup: Take a serial computation and estimate how quickly that could be executed on multiple cores**

- **Gustafson-Barsis's Law does the opposite: Start with a parallel computation and estimate how much faster the parallel computation is than the same computation executed on a single core**

- **Recall details used for derivation of Amdahl's Law:**
  - ⋆ **Speedup is** $S_P = \frac{T_1}{T_P}$
  - ⋆ T_1 **denotes the (single core) time of execution of the best sequential solution of a scientific computing problem**
  - ⋆ $T_P = T_{COMP,P} + T_{COMM,P} = fT_1 + (1-f)T_1/P + T_{COMM,P}$ **is the execution time for the parallel solution of the problem, using** P **cores**
  - ⋆ $T_{COMP,P}$ **and** $T_{COMM,P}$ **respectively are the parallel computation and overhead/communication time taken by a code to solve a parallel scientific computing problem using** P **cores**
  - ⋆ f **is the sequential fraction of the code**

[13] M. Ganesh, MATH440/540, SPRING 2018

- [14]**We write** $T_1 = fT_1 + (1-f)T_1$
- **Note that** fT_1 **is the execution time for the sequential part of the code**
- **We have** $T_P = fT_1 + (1-f)T_1/P + T_{COMM,P}$, **with** $T_{COMM,P} > 0$
- **Let** s **denote the** ***fraction of time*** **spent in the** ***parallel computation*** **performing inherently sequential operations**
- **That is, let**
$$s = \frac{fT_1}{fT_1 + (1-f)T_1/P} = \frac{f}{f + (1-f)/P}$$
- **Gustafson-Barsis's Law is based on scaled-speedup,** $\widehat{SS}_P$, **defined by**
$$\widehat{SS}_P = P + (1-P)s$$

[14] M. Ganesh, MATH440/540, SPRING 2018

[15]Gustafson-Barsis's Law :

$$S_P \leq P + (1-P)s = \widehat{SS}_P$$

Proof:

[15] M. Ganesh, MATH440/540, SPRING 2018

- [16]**Recall:**
  - ⋆ **Amdahl's Law estimate for speedup:**

$$S_P \leq \frac{1}{f + (1-f)/P}$$

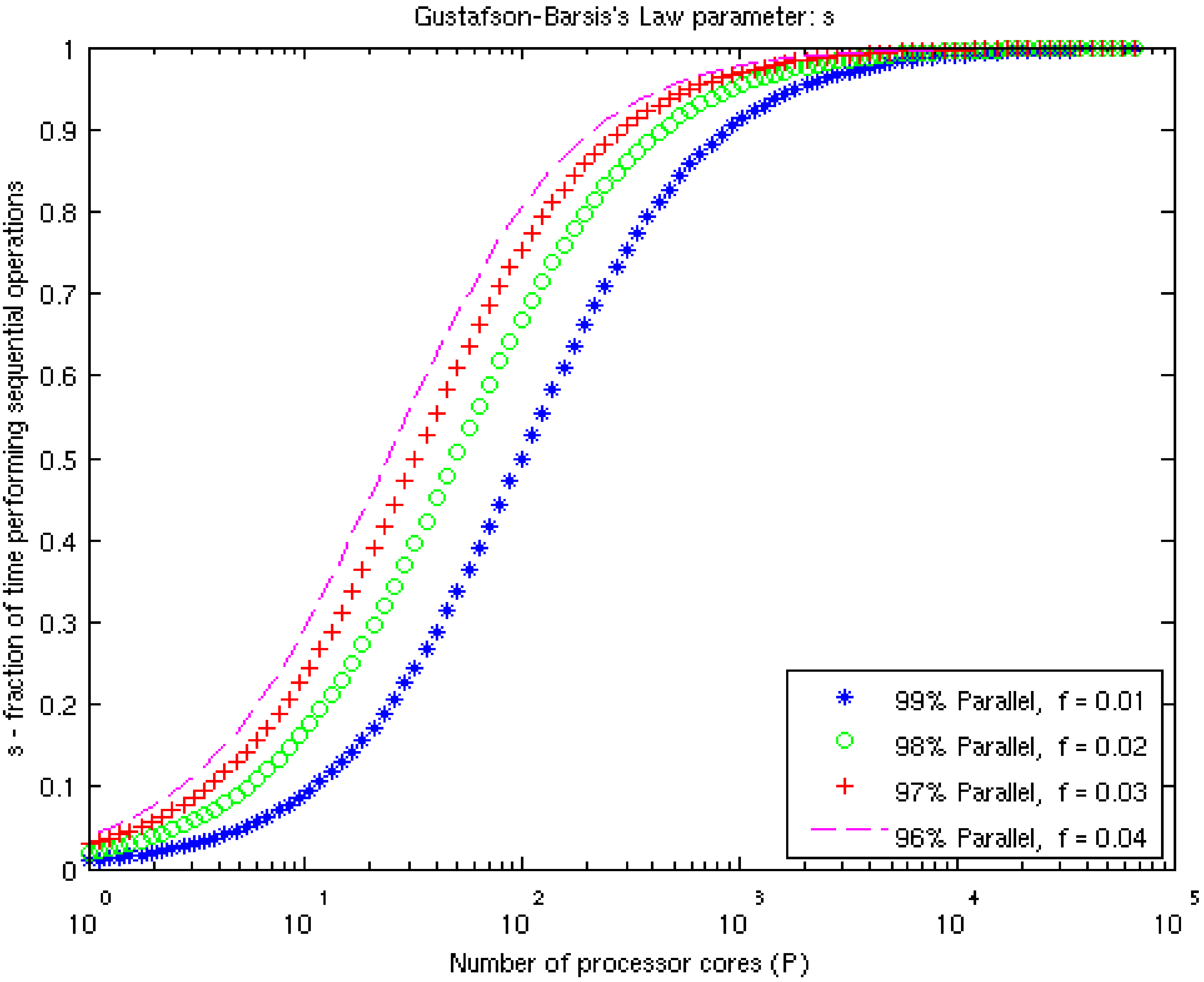
  - ⋆ **Gustafson-Barsis's Law estimate for speedup:**

$$S_P \leq P + (1-P)s = \widehat{SS}_P, \qquad s = \frac{f}{f + (1-f)/P}$$

- **Since the proof of Gustafson-Barsis's Law is similar to that used to derive Amdahl's law, do we expect substantially improved speedup plot, compared to the earlier one? No (see next figure or plot $\widehat{SS}_P$)**
- **However, the scaled-speedup estimate helps to identify that we need $(1-P)s \to 0$ as $P \to \infty$, to attain almost linear speedup**
- Exercise: **Show that the Amdahl's law estimate for speedup is same as that of Gustafson-Barsis's Law speedup. Following the format used for Amdahl's law demonstration plots, plot Gustafson-Barsis's estimate for various fixed values of f.**

[16] M. Ganesh, MATH440/540, SPRING 2018

-


Gustafson-Barsis's Law parameter: s
s - fraction of time performing sequential operations
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
10^0
10^1
10^2
10^3
10^4
10^5
Number of processor cores (P)
99% Parallel, f = 0.01
98% Parallel, f = 0.02
97% Parallel, f = 0.03
96% Parallel, f = 0.04

- [17]**In the last figure we observed that s (the *fraction of time* spent in the *parallel computation* performing inherently sequential operations) tends to 1 as the number of processors increase**

- **This is not a surprise because in all previous figures in this section, we chose f to be independent of the number of processors used:**

$$\lim_{P\to\infty} s = \lim_{P\to\infty} \frac{f}{f+(1-f)/P} = 1$$

- **However, for several practical applications/algorithms, the values of f and s decrease as the complexity (or data size) of the problem increases**

- **This is because, in practice, the number of processors and the size of f (and hence s) depend on the data size**

- **Before we investigate data dependent estimates, we consider examples related to s and an experimental way to determine f**

[17] M. Ganesh, MATH440/540, SPRING 2018

[18]**Exercise**:

- **Suppose that a parallel code executed on** 1000 **processors requires one full day to run the job. Careful benchmarking of the execution of the code shows that one hour of the computation is spent on executing serial portions of the computations on a single processing core. What is the scaled-speedup of the code?**
- **Solution**:

[18] M. Ganesh, MATH440/540, SPRING 2018

[19]**Exercise:**

- **Currently a cluster XX is equipped with 2144 cores. Suppose that you have a research model/application that requires at least five thousand cores to simulate (within a reasonable amount of time) and demonstrate that the model/application is very important.**

  Suppose that $\$3$ million is required to add another 3000 cores to XX.

  Suppose that an agency is ready to loan $\$3$ million to make XX a 5144 core cluster. Further, you are given a challenge that the loan can be considered as a donation provided that there will be a demonstration of scaled-speedup of 5000 for a few applications on the upgraded XX.

  To get the donation, what types of codes will you consider?

- **Solution:**

[19] M. Ganesh, MATH440/540, SPRING 2018

- [20]Note that in the derivations of Amdahl's Law and Gustafson-Barsis's Law, overhead/communication time, $T_{COMM,P}$, was ignored, by using upper bounds, leading to overestimates
- Speedup and efficiency using $P > 1$ processing cores crucially depend on the overhead resulting from processor communication and synchronization (induced by, for example, message latency, network or bus contention, or network bandwidth limitations), and redundant computations. We denoted the total overhead by $T_{COMM,P}$
- Based on Amdahl's Law and Gustafson-Barsis's Law, it is important to determine the serial fraction of the code to estimate the speedup
- In practice, it is useful to have a metric to compute the serial fraction of the code, that also depends on the overhead $T_{COMM,P}$
- Next we consider a metric to experimentally determine serial fraction of the code, that includes the overhead $T_{COMM,P}$

[20] M. Ganesh, MATH440/540, SPRING 2018

[21]The Karp-Flatt Metric : Measurement of serial fraction

- **We recall the notation used for a parallel computation with P cores:**

$$T_1 = fT_1 + (1-f)T_1, \qquad T_P = fT_1 + (1-f)T_1/P + T_{COMM,P}, \tag{3.1}$$

where f is the sequential fraction of the parallel computation

- **If we assume the overhead $T_{COMM,P} = 0$, the measurement of f based on S_P is easy, leading to the Karp-Flatt Metric (without overhead):**

$$f = \frac{\frac{1}{S_P} - \frac{1}{P}}{1 - \frac{1}{P}}$$

- **Proof :**

[21] M. Ganesh, MATH440/540, SPRING 2018

- [22]**We define the experimentally determined serial fraction, exp_f, of the parallel computation using the total amount of idle time (in $P-1$ cores) and the overhead time:**

$$exp_f = f + \frac{PT_{COMM,P}}{(P-1)T_1} = \frac{(P-1)fT_1 + PT_{COMM,P}}{(P-1)T_1} \tag{3.2}$$

- **The Karp-Flatt Metric (with overhead) to compute exp_f is:**

$$exp_f = \frac{\frac{1}{S_P} - \frac{1}{P}}{1 - \frac{1}{P}} \tag{3.3}$$

- **Proof :**

[22] M. Ganesh, MATH440/540, SPRING 2018

- [23]**Exercise: Consider the following benchmarking of a code on two to eight processing cores and the corresponding speedup:**

P	**2**	**3**	**4**	**5**	**6**	**7**	**8**
S_P	**1.82**	**2.50**	**3.08**	**3.57**	**4.00**	**4.38**	**4.71**

  - ⋆ **What is the primary reason for the speedup** 4.71 **with** 8 **cores?**
  - ⋆ **What is the best possible speedup you expect with** 1000 **cores?**

- **Solution:**

[23] M. Ganesh, MATH440/540, SPRING 2018

-

- [24]**Exercise: Consider the following data that is closely related to the data in the last example:**

P	**2**	**3**	**4**	**5**	**6**	**7**	**8**
S_P	**1.87**	**2.61**	**3.23**	**3.73**	**4.14**	**4.46**	**4.71**

What is the primary reason for the speedup 4.71 **with** 8 **cores?**

- **Solution:**

[24] M. Ganesh, MATH440/540, SPRING 2018

- **The Karp-Flatt Metric (without and with overhead cost)** $f, exp_f = \left[\frac{1}{S_P} - \frac{1}{P}\right] / [1 - \frac{1}{P}]$ **suggests that**

$$f, exp_f \to \frac{1}{S_P}, \qquad \text{as} \quad P \to \infty$$

- **That is the speedup is inversely proportional to the serial fraction. For S_P to increase as the number of processors increase, it is necessary f, exp_f should depend on P and should decrease.**

- **Exercise Using $T_P = T_{COMP,P} + T_{COMM,P}$ and $T_1 \leq PT_{COMP}$, prove that**

$$S_P \leq P\left(1 + \frac{T_{COMM,P}}{T_{COMP,P}}\right)^{-1}, \qquad E_P \leq \left(1 + \frac{T_{COMM,P}}{T_{COMP,P}}\right)^{-1}. \qquad (3.4)$$

- **Solution:**

- **Further, for the special case $T_1 = PT_{COMP,P}$, the inequalities in (??) become equalities. That is,**

$$S_P = \qquad\qquad\qquad , \qquad E_P = \tag{3.5}$$

- [25]**Example:**
  Consider an algorithm for a problem that is characterized by the data size N (such as summing up N values), implemented in a parallel code and executed using P cores
  - ⋆ **Let $T_{COMM,P}$ be proportional to the number of cores P and $T_{COMP,P}$ is proportional to N/P**
  - ⋆ **Let $T_{COMM,P} = c_1 P$, $T_{COMP,P} = c_2 \frac{N}{P}$, for some unknown constants c_1, c_2**
  - ⋆ **Further assume that $T_1 = PT_{COMP,P}$**

  Then, using (??) and the unknown proportionality constant $c = c_1/c_2$, the speedup and efficiency of the code are given by

$$S_P = \qquad\qquad = \qquad\qquad , \qquad E_P = \tag{3.6}$$

[25] M. Ganesh, MATH440/540, SPRING 2018

Peak performance using P processing cores - Gordon Bell Award

- **1988: 1 GFLOPS with $P = 8$ (Application: Structures)**
- **1990: 14 GFLOPS with $P = 2,048$ (Application: Seismic)**
- **1992: 5.4 GFLOPS with $P = 512$ (Application: Gravitation)**
- **1994: 143 GFLOPS with $P = 1,904$ (Application: Structures)**
- **1996: 111 GFLOPS with $P = 160$ (Application: CFD)**
- **1998: 1,020 GFLOPS with $P = 1,536$ (Application: Magnetism)**
- **2000: 1,349 GFLOPS with $P = 96$ (Application: Gravitation)**
- **2002: 26,500 GFLOPS with $P = 5,120$ (Application: Climate)**
- **2004: 15, 200 GFLOPS with $P = 4,096$ (Application: CFD)**
- **2006: 207, 000 GFLOPS with $P = 131,072$ (Application: Elect. Struct.)**

Price $ per MFLOPS performance - Gordon Bell Award

- **1989: $2,500 (Application: Reservoir modeling)**
- **1999: $6.90 (Application: Quantum molecular dynamics)**
- **2009+: Less than one cent! (thanks to GPU, CELL, PHI)**

- [26]In the last exercise, using (??), if we carefully choose the number of processing cores P to depend on the data size N of the problem, we can achieve the speedup to be proportional to the number of cores and the efficiency can be made positive for all N and hence $P(N)$
- In particular, if we choose $P(N) = \sqrt{N}$ in (??) we get

$$S_P = \qquad\qquad\qquad\qquad , \qquad\qquad\qquad E_P = \qquad\qquad\qquad > 0 \tag{3.7}$$

- For such problems with appropriate data dependent choice of cores, the sequential fraction f of the code decreases as the data size N increases, because f is inversely proportional to S_P
- Thus in practice, we need to consider all quantities introduced so far in this section to depend on the data size of a chosen problem (denoted throughout by N)

[26] M. Ganesh, MATH440/540, SPRING 2018

- [27]**Thus we replace the notation**

$$T_1, T_P, T_{COMP,P}, T_{COMM,P}, S_P, E_P, f, s, \widehat{SS}_P$$

with

$$T_1(N), T_P(N), T_{COMP,P}(N), T_{COMM,P}(N), S_P(N), E_P(N), f(N), s(N), \widehat{SS}_P(N)$$

- **In particular, the data-dependent speedup, $S_P(N)$, and the data-dependent parallel efficiency, $E_P(N)$, using P processing cores for a problem with data size N are defined as**

$$S_P(N) = \frac{T_1(N)}{T_P(N)}, \qquad E_P(N) = \frac{S_P(N)}{P} = \frac{T_1(N)}{PT_P(N)}$$

- **An algorithm for a problem with data size N is said to be scalable, if there is a minimal efficiency $\epsilon > 0$ (independent of N) such that *for any* data size N chosen for the problem, there is a number of processing cores $P(N)$ (with $P(N) \to \infty$ as $N \to \infty$) such that**

$$E_{P(N)}(N) \geq \epsilon > 0$$

[27] M. Ganesh, MATH440/540, SPRING 2018

- [28]**Exercise**:
  Consider a class of algorithms with data size N to be implemented on a cluster using P processing cores. Assume that the algorithms are such that

  $$T_1(N) = c_1 P T_{COMP,P}, \qquad \frac{T_{COMM,P}(N)}{T_{COMP,P}(N)} = c_2 N^{-k} P^{\ell}, \qquad \text{for some } c_1, c_2, k, \ell > 0$$

  (In the last example, $k = 1, \ell = 2$.) Are these algorithms scalable? If yes, what are the choices of $P(N)$, for a minimal efficiency $\epsilon > 0$.

- **Solution**:

[28] M. Ganesh, MATH440/540, SPRING 2018

- [29]**For scalable algorithms, $S_{P(N)}(N) \geq \epsilon P(N)$, for some fixed $\epsilon > 0$**
- **Thus for scalable algorithms, the speedup is an increasing function of the problem size. This is also known as Amdahl's effect**
- **The scalability of an algorithm is a measure of its ability to increase performance as the number of processing cores increases**
- **For scalable algorithms, the speedup is at least linear with respect to the number of the processing cores $P(N)$**
- **For scalable algorithms, the sequential fraction $f(N)$ is a decreasing function of the processing cores $P(N)$**
- **Suppose that a parallel code, executed for various data size parameter N and processing cores P, exhibits efficiency $E_P(N)$**
- **Next we consider a metric for N as the number of cores P increase:**
- **In order for the code to maintain the same level of efficiency as the number of processors P increase, is there metric on N?**

[29] M. Ganesh, MATH440/540, SPRING 2018

[30]The Isoefficiency Metric

- **Let $C = \frac{E_P(N)}{1-E_P(N)}$. For a parallel code to maintain the same level of efficiency as the number of processors P increase (that is to keep C as a constant, independent of N) N must be increased so that**

$$T_1(N) \geq C\widetilde{T}_P(N),$$

where $\widetilde{T}_P(N)$ is the total amount of time spent by all processing cores doing work not by the sequential part of the code, defined by

$$\widetilde{T}_P(N) = (P-1)f(N)T_1(N) + PT_{COMM,P}(N)$$

- **Proof:**

[30] M. Ganesh, MATH440/540, SPRING 2018

[31]Memory limited scalability

- **In various present HPC cluster systems, a processing core can access memory only within the node in which it is located**
- **The amount RAM available per processing core in a node of a supercomputer (say BLUE-AuN) is not substantially higher than the RAM available per core in a HPC workstation.**
- **For example, each AuN node has 64 GB RAM and 16 cores and each Sayers Lab machine has 16 GB RAM and 8 cores!**
- **Hence it may not be possible implement a scalable algorithm, for a problem size N with $P(N)$ chosen to satisfy the scalability condition, as nodes may not be able to accommodate the divided data within its limited RAM, required perform computations**
- **This leads to the concept of scalable with respect to memory**

[31] M. Ganesh, MATH440/540, SPRING 2018

- [32]Consider a parallel algorithm for a problem with data size N, to be executed on HPC cluster, using P processing cores in $\widetilde{P}$ nodes
- Let $Mem(P, N)$ be the amount of memory (say in GB) required per processing core to execute the algorithm with data size N and P processing cores
- So the total amount of memory required to execute the algorithm within the $\widetilde{P}$ nodes is $P \times Mem(P, N)$ GB
- If p_i cores are used in the i-th node, then the i-th node should have at least $p_i \times Mem(P, N) GB$ memory, $i = 1, \cdots, \widetilde{P}$
- An algorithm for a problem with data size N is said to be scalable with respect memory, if there is a minimal efficiency $\epsilon > 0$ and a fixed constant Mem_{max} (maximum memory number), both independent of N, such that *for any* data size N chosen for the problem, there is a number of processing cores $P(N)$ (with $P(N) \to \infty$ as $N \to \infty$) such that

  $$E_{P(N)}(N) \geq \epsilon > 0, \qquad\qquad Mem(P(N), N) \leq Mem_{max}$$

  as the data size N is allowed to grow arbitrarily large

[32] M. Ganesh, MATH440/540, SPRING 2018

- [33]**Exercise**:
  Consider a class of algorithms with data size N to be implemented on a cluster using P processing cores. Assume that the algorithms are such that

  $$T_1(N) = c_1 P T_{COMP,P}, \qquad \frac{T_{COMM,P}(N)}{T_{COMP,P}(N)} = c_2 N^{-k} P^{\ell}, \qquad \textbf{for some } c_1, c_2, k, \ell > 0$$

  In addition, assume that $Mem(P, N) \leq c_3 \frac{N^m}{P^n}$. What is the required range of $P(N)$ for these these algorithms to be scalable with respect memory? Are these algorithms scalable with respect memory?

- **Solution**:

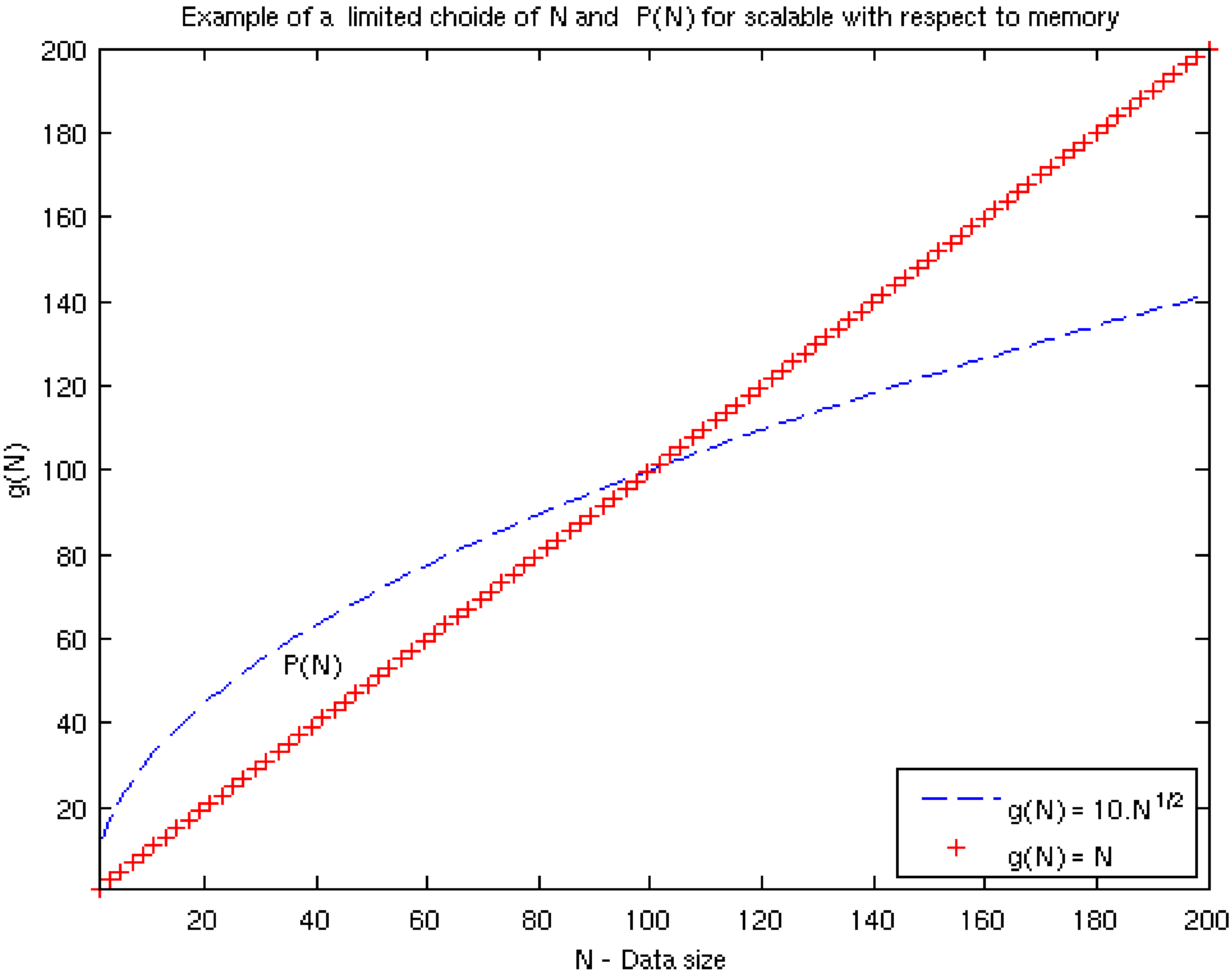
[33] M. Ganesh, MATH440/540, SPRING 2018

.

- **Find values of $c_1, c_2, c_3, k.\ell, m, n$ so that the range of $P(N)$ for the class of algorithms in the last exercies is given by**

$$N \leq P(N) \leq 10\sqrt{N}$$

- **Clearly, the above inequalities hold only for a limited choice of N (see the following figure for a visual range of N) and not for arbitrarily large values of N, required in the definition**



Example of a limited choide of N and P(N) for scalable with respect to memory
200
180
160
140
120
100
80
60
40
20
g(N)
20
40
60
80
100
120
140
160
180
200
N - Data size
P(N)
g(N) = 10.N 1/2
g(N) = N

- [34]**Even if an algorithm is scalable with respect to memory, it may not be scalable in a HPC cluster if Mem_{max} in the definition is larger than the maximum memory available in required computing nodes.**
- **In a HPC system, let Max_{mem} be the maximum amount of memory available in any computing node in the system**
- **Let $Mem(P, N)$ be the amount of memory required per processing core to execute the algorithm with data size N and P cores**
- **Let $Mem(P, N)$ be a decreasing function of P for any given N. That is for a fixed N, the amount of memory requirement in the algorithm decreases as the number of cores/nodes are increased**
- **The choice**

  $$P(N) = \min\{P : Mem(P, N) \le Max_{mem}\}$$

  lead to a scaling of the algorithm for the particular amount of memory Max_{mem}. This scaling is known as memory constrained scaling

[34] M. Ganesh, MATH440/540, SPRING 2018

[35]Less technical definitions of scalability

- **In this section, we first considered the definition of speedup for fixed size problems (that is the data size N is fixed, S_P and P are independent of N). This leads to the definition of strong scaling:**
- **An algorithm with a fixed data size is said to be strongly scalable if the execution time decreases in inverse proportion to the number of processing cores**
- **In the later part of this section, we considered the definition of speedup with respect to the data size N of the algorithm and that P is a function of N. This leads to the definition of weak scaling:**
- **An algorithm is said to be weakly scalable if the execution time remains constant, as the data size and the processing cores are increased in proportion and the data size per processing core is fixed**
- **There are various sub-class of weakly scalable algorithms, based on the last part of this section, such as the memory limited weakly scalable algorithms**

[35] M. Ganesh, MATH440/540, SPRING 2018

[36]Less technical approach to predict parallel performance

- **For strongly scalable algorithms or in general for efficient parallel scientific computing algorithms, it is natural to assume the following:**

  - ⋆ **The overhead time increases as the number of processing cores increases. (That is $T_{COMM,P}$ is an increasing function of P.)**
  - ⋆ **The parallel computation time decreases as the number of cores increases. (That is $T_{COMP,P}$ is an decreasing function of P.)**

- **Our interest, for a fixed problem with a fixed data size, is to determine if we gain anything by increasing the number of cores and, more importantly, when to stop increasing P, for parallel scientific computing algorithms with polynomial complexity.**

- **We assume that there exists a constant C (ideal case $C = 1$) such that**

$$T_1 = CPT_P = c\left[T_{COMP,P} + T_{COMM,P}\right] \tag{3.8}$$

[36] M. Ganesh, MATH440/540, SPRING 2018

- [37]**Taking natural** $\log$ **in (??), we get**

  $$\log T_P = \widetilde{C} - \log P, \qquad \textbf{for some fixed constant} \quad \widetilde{C} = \log T_1 - \log C$$

- **That is in a log-log graph, with** $x = \log P$ **and** $y = \log T_P$**, we get a linear plot with slope** -1

- **So, in general, for parallel performance, in logarithmic scale, curves that are decreasing are good**

- **Note that in the total computing time**

  $$T_P = T_{COMP,P} + T_{COMM,P},$$

  in most practical cases, we have the first component of RHS is decreasing but the second component of RHS is increasing

[37] M. Ganesh, MATH440/540, SPRING 2018

- [38]**A parallel performance based rule of thumb is to stop increasing the number P of processing cores when the slope of the curve for the total time in the $\log\log$ plot of P and T_P changes from negative to positive**

- **In particular, the rule of thumb is to stop increasing P when the increasing component of T_P meets the decreasing component of T_P in the $\log\log$ performance prediction graph. (That is at the point of intersection of the curves $T_{COMM,P}$ and $T_{COMP,P}$.)**

- **Exercise: Demonstrate the rule of thumb using various plots for a class of algorithms with fixed data size N and**

$$T_{COMP,P} = \frac{N^\alpha}{P^\beta}, \qquad T_{COMM,P} = cP^\gamma,$$

where the fixed constants α, β, γ and c are such that $T_{COMP,P}$ and $T_{COMM,P}$ are, respectively, decreasing and increasing functions of P

[38] M. Ganesh, MATH440/540, SPRING 2018