Speedup and efficiency of computational parallelization: A unifying approach and asymptotic analysis

Guido Schryen[a,*]

[a]*Department of Management Information Systems, Paderborn University, Warburger Strasse 100, Paderborn 33098, Germany*

ARTICLE INFO

Keywords:

Performance analysis

Speedup

Efficiency

Scalability

Asymptotic analysis

ABSTRACT

In high performance computing environments, we observe an ongoing increase in the available number of cores. For example, the current TOP500 list reveals that nine clusters have more than 1 million cores. This development calls for re-emphasizing performance (scalability) analysis and speedup laws as suggested in the literature (e.g., Amdahl's law and Gustafson's law), with a focus on asymptotic performance. Understanding speedup and efficiency issues of algorithmic parallelism is useful for several purposes, including the optimization of system operations, temporal predictions on the execution of a program, the analysis of asymptotic properties, and the determination of speedup bounds. However, the literature is fragmented and shows a large diversity and heterogeneity of speedup models and laws. These phenomena make it challenging to obtain an overview of the models and their relationships, to identify the determinants of performance in a given algorithmic and computational context, and, finally, to determine the applicability of performance models and laws to a particular parallel computing setting. In this work, I provide a generic speedup (and thus also efficiency) model for homogeneous computing environments. My approach generalizes many prominent models suggested in the literature and allows showing that they can be considered special cases of a unifying approach. The genericity of the unifying speedup model is achieved through parameterization. Considering combinations of parameter ranges, I identify six different asymptotic speedup cases and eight different asymptotic efficiency cases. Jointly applying these speedup and efficiency cases, I derive eleven scalability cases, from which I build a scalability typology. Researchers can draw upon my suggested typology to classify their speedup model and to determine the asymptotic behavior when the number of parallel processing units increases. Also, the description of two computational experiments demonstrates the practical application of the model and the typology. In addition, my results may be used and extended in future research to address various extensions of my setting.

1. Introduction

Parallel computing has become increasingly important for solving hard computational problems in a variety of scientific disciplines and industrial fields. The large diversity and deployment of parallel computing across disciplines, including artificial intelligence, arts and humanities, computer science, digital

[*]Corresponding author

guido.schryen@uni-paderborn.de (G. Schryen)

ORCID(s):

agriculture, earth and environmental sciences, economics, engineering, health sciences, mathematics, and natural sciences, is mirrored in usage statistics published by supercomputer clusters (e.g., [28, 23]). This ongoing progress in computational sciences through parallelization has been fostered through the end of exponential growth in single processor core performance [16] and the availability of high performance computing (HPC) infrastructures, tools, libraries, and services as commodity goods offered by computing centers of universities, public cloud providers, and open source communities. The development of parallel computing has been accompanied by the study of its performance. Generally speaking, performance in parallel computing refers to the behavior of a parallel computing system in processing specified tasks with respect to the amount of resources, such as parallel computing units, that are used or available. It encompasses a variety of metrics and concepts, including speedup, efficiency, load balancing, and communication overhead, among others. Performance has also been studied as the amount of parallel computing resources grows to infinity (asymptotic performance), resulting in a variety of speedup laws. For an introduction to performance in parallel computing, see, for example, [18, ch. 5].

Beyond these developments, the number of cores available as parallel processing units has increased substantially over the past years. While the statistics of the TOP500 list (as of June 2022) shows values of 35,339.2 (10th percentile), 67,328 (median), and 225,465.6 (90th percentile), the corresponding values of the lists as of June 2017 and June 2012 amount to (16,545.6; 36,000; 119,808) and (6,776; 13,104; 37,036.8), respectively [38]. In addition, in contrast to the lists of 2012 and 2017, which both include only one site with more than 1 million cores, the current list shows that nine clusters have more than 1 million cores. This enormous growth in the number of cores which are available for parallel processing calls for re-emphasizing asymptotic performance analysis (e.g., [10, 3]) and speedup laws as suggested in the literature (e.g., Amdahl's law [4] and Gustafson's law [19]).

In general, studying performance of algorithmic parallelism is useful for several purposes; these include optimizing system operations via design-time and run-time management (e.g., [20, 44, 40, 43, 11]), making temporal predictions on the execution of a program (e.g., [32, 1]), and analyzing asymptotic speedup and efficiency properties as well as determining speedup and efficiency bounds (e.g., [35, 10]). In this article, I focus on the two latter purposes, which have been addressed only rarely in the literature.

Analyzing performance of parallel algorithms is challenging as it needs to account for diversity in several regards. For example, existing speedup models and laws make different assumptions with respect to the homogeneity/heterogeneity of parallel processing units, variations of workloads, and methodological characteristics and application fields of algorithms (e.g., optimization, data analytics, simulation). This heterogeneity has resulted in a landscape of many speedup models and laws, which, in turn, makes it difficult to obtain an overview of the models and their relationships, to identify the determinants of performance in a given algorithmic and computational context, and, finally, to determine the applicability of performance models and laws to a particular parallel computing setting.

My focus lies on the development of a generic and unifying speedup and efficiency model for homogeneous parallel computing environments. I consider a range of determinants of speedup covered in the literature and prove that existing speedup laws can be derived from special cases of my model. My model depends neither on specific system architectures, such as symmetric multiprocessing (SMP) systems or graphics processing units (GPU), nor on software properties, such as critical regions; I rather perform a theoretical analysis although I also conduct computational experiments to demonstrate the application of the model. I further focus on the analysis of asymptotic properties of the suggested model to study speedup and efficiency limits and bounds in the light of a computing future with an increasing number of parallel processing units.

My results contribute to research on the performance (in terms of scalability) of computational parallelization in homogeneous computing environments in several regards: (1) I suggest a generic speedup and efficiency model which accounts for a variety of conditions under which parallelization occurs so that it is broadly applicable. This wide scope allows conducting performance analysis in many of those cases which are not covered by existing models and laws with restrictive assumptions. (2) I generalize the fragmented landscape of speedup and efficiency models and results, and I provide a unifying speedup and efficiency model which allows overcoming the perspective of conflicting speedup models by showing that these models can be interpreted as special cases of a more universal model. (3) From my asymptotic analysis, I derive a typology of scalability (speedup and efficiency), which researchers may use to classify their speedup model and/or to determine the asymptotic behavior of their particular application. I also provide a theoretical basis for explaining sublinear, linear and superlinear speedup and efficiency and for deriving speedup and efficiency bounds in the presence of an enormous growth of the number of available parallel processing units. (4) I demonstrate the practical application of the speedup and efficiency model and the typology with computational experiments on matrix multiplication and lower-upper matrix decomposition. To sum up, I consolidate prior research on performance in homogeneous parallel computing environments and I provide a theoretical understanding of quantitative effects of various determinants of asymptotic performance in parallel computing.

The remainder of the article is structured as follows: In Section 2, I provide a brief overview of the foundations of speedup and efficiency analysis in parallel computing. I proceed in Section 3 with the suggestion of a generic speedup and efficiency model. In Section 4, I perform a mathematical analysis of my model in order to determine theoretical speedup and efficiency limits. I describe the computational experiments in Section 5. In Section 6, I discuss the application of the proposed model and scalability typology, and I consider parallelization overhead. Finally, I provide conclusions of my research in Section 7.

2. Foundations of speedup and efficiency analysis

The main purpose of parallel computation is to take advantage of increased processing power to solve problems faster or to achieve better solutions. The former goal is referred to as *scalability*, and scalability measures fall into two main groups: *speedup* and *efficiency*. Speedup $S(N)$ is defined as the ratio of sequential computation time $T(1)$ to parallel computation time $T(N)$ needed to process a task with given workload when the parallel algorithm is executed on N parallel processing units (PUs) (e.g., cores in a multicore processor architecture); i.e.,

$$S(N) := \frac{T(1)}{T(N)},\ T : \mathbb{N} \rightarrow R^{>0} \tag{1}$$

The sequential computation time $T(1)$ can be measured differently, leading to different interpretations of speedup [6]: When $T(1)$ refers to the fastest serial time achieved on any serial computer, speedup is denoted as *absolute*. Alternatively, it may also refer to the time required to solve a problem with the parallel program on one of the parallel PUs. This type of speedup is qualified as *relative*. In this work, I focus on relative speedup.

As speedup relates the time required to process a given workload on a single PU to the time required to process the same workload on N PUs, you need to determine this workload. It is usually divided into two sub-workloads, the sequential workload and the parallelizable workload. While the former is inherently sequential and necessarily needs to be executed on a single PU, the latter can be executed in parallel on several PUs. Independent of the number of available parallel PUs N, the time required to solve a task is the sum of the time to handle the sequential workload and the time to handle the parallelizable workload of the given task. When only a single PU is available, the time for the sequential workload s and for the parallelizable workload p are usually normalized by setting $s + p = 1$; i.e., s and p represent the sequential and the parallelizable fractions of the overall execution time.

For some applications, it is useful to consider a fixed workload (e.g., when solving an instance of an optimization problem), which is independent of the number of parallel PUs (N) available, and then to analyze how computation of the fixed workload on a single PU can be speeded up by using multiple PUs. Speedup models of this type are referred to as *fixed-size models*, such as Amdahl's law [4]. For other applications (e.g., when analyzing data), is more appropriate to use the availability of N PUs to solve tasks with workloads which increase depending on N. Then, scalability analysis deals with investigating how computation of the variable workload on one PU can be speeded up by using multiple PUs. Speedup models of that type are referred to as *scaled-size models*, such as Gustafson's law [19].

With varying number of PUs N, both the sequential and parallelizable workload may be considered scalable. It is common in the literature to introduce two workload scaling functions $f(\cdot), g(\cdot)$ with $f, g : \mathbb{N} \rightarrow R^{>0}$ for the sequential and parallelizable workload, respectively; i.e.; the (normalized) time to process the sequential and the parallelizable workload on a single PU are $s \cdot f(N)$ and $p \cdot g(N)$, respectively. Thus, the (normalized) time to process the overall workload on a single PU amounts to $s \cdot f(N) + p \cdot g(N)$. Usually, it is assumed that $f(1) = g(1) = 1$ so that $T(1) = s + p = 1$ holds; however, my workload scaling functions do not require to meet this assumption.[1] An example of using a scaling function for the sequential workload can be found in the scaled speedup model suggested by Schmidt et al. [34, p. 31ff]. While scaling functions for sequential workloads can be found only rarely, scaling functions for parallelizable workloads are much more common; see, for example, the scale-sized speedup model of Gustafson [19], the memory-bound speedup model of Sun and Ni [36, 37, 35], the generalized scaled speedup model of Juurlink and Meenderinck [22] and the scaled speedup model of Schmidt et al. [34, p. 31ff]. A discussion of the relationship between problem size and the number of PUs N can be found in [39, p. 32f].

[1] The option to have values $f(1) \neq 1$ and/or $g(1) \neq 1$ allows scaling both fractions s and p, which may be useful when an overall workload to be executed on a machine A (with $p + s = 1$) is now executed on a different machine B on which the times to execute the serial and the parallelizable workload are scaled at either the same or different rates.

While the time required to process the sequential workload is independent of the number of PUs N, the time required to process the parallelizable workload depends on N as this workload can be processed in parallel. Usually, the parallelizable workload is considered to be equally distributed on N PUs, resulting in the (normalized) time $\frac{p \cdot g(N)}{N}$ to handle the parallelizable workload. However, there are tasks possible when the time required to handle the parallelizable workload is affected due to its actual parallel execution; for example, when a mathematical optimization problem, such as a mixed-inter linear program (MILP), is solved with a parallelized branch-and-bound algorithm, then good bounds may be found early so that the branch-and-bound tree does not grow as large as with the sequential execution of the branch-and-bound algorithm. This effect may result in a denominator function which is not identical to N and allows explaining superlinear speedup as it has been observed in the literature (e.g., [33, 13, 17]). I account for this effect with a scaling function $h(\cdot)$, with $h : \mathbb{N} \to R^{>0}$.

Finally, processing one single large task on several parallel PUs involves some sort of overhead, which is often rooted in initialization, communication, and synchronization efforts [41, 21, 15]. I account for the additional time required for these efforts with an overhead function $z(\cdot)$, $z : \mathbb{N} \to R^{>0}$.

The abovementioned workloads and temporal effects are visualized in Figure 1. The resulting general speedup equation is the given by

$$S(N) = \frac{T(1)}{T(N)} = \frac{s \cdot f(1) + p \cdot g(1)}{s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N)}, N \in \mathbb{N} \tag{2}$$

Note that the speedup equation given in (2) is a generalization of several well known speedup models, including those used in Amdahl's law [4] (set $f(N) = g(N) = 1, h(N) = N, z(N) = 0$), Gustafson's law [19] (set $f(N) = 1, g(N) = h(N) = N, z(N) = 0$), and the generalized scaled speedup model [22] (set $f(N) = 1, g(N) = N^{0.5}, h(N) = N, z(N) = 0$).

Based upon speedup $S(N)$, efficiency $E(N)$ relates speedup to the number of parallel PUs used to achieve this speedup, and it is defined by

$$E(N) = \frac{S(N)}{N} = \frac{T(1)}{N \cdot T(N)} = \frac{s \cdot f(1) + p \cdot g(1)}{N \cdot (s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N))}, N \in \mathbb{N} \tag{3}$$

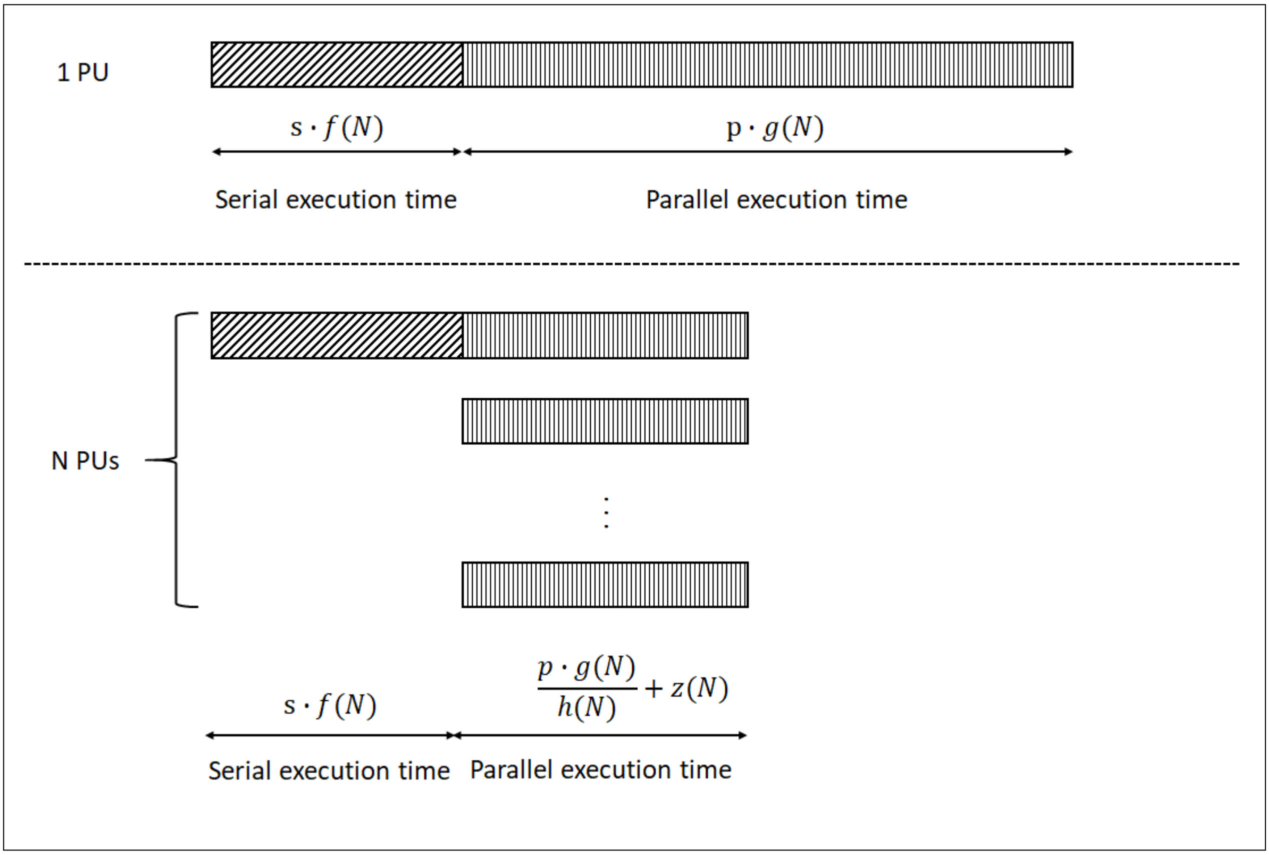

Figure 1: Workloads and temporal effects of parallelization

3. A generic speedup and efficiency model

Based upon the general speedup equation (2), I derive a generic speedup and efficiency model, which I use in the remainder of this article to analyze its asymptotic behavior. The generic speedup model uses power functions for $f(\cdot), g(\cdot), h(\cdot)$ and ignores any overhead induced through parallelization. The use of power functions is widely adopted in the literature, included in many prominent speedup models [4, 19, 34, 37, 22] and is based on the assumption that many algorithms have a polynomial complexity in terms of computation and memory requirement [35, p. 184]. As I focus on the analysis of the asymptotic behavior, I always take the highest degree term. The motivation for neglecting any parallelization overhead (i.e., $z(N) = 0 \; \forall N \in \mathbb{N}$), as it is done in many, if not most speedup and efficiency models in the literature, is manifold: First, the overhead is often unknown. Second, omitting an overhead term simplifies computations and provides a basis for developing laws which include an overhead function $z(N) \neq 0$. Third, speedup and efficiency values determined without considering overhead represent upper bounds for practically achievable speedup and efficiency values when overhead occurs.

I use the following power functions:

$$f(N) := c_f N^{\alpha_f} \qquad g(N) := c_g N^{\alpha_g} \qquad h(N) := c_h N^{\alpha_h}, \qquad c_f, c_g, c_h > 0,\ \alpha_f, \alpha_g, \alpha_h \geq 0 \tag{4}$$

and yield the following generic speedup equation (for $N > 1$)[2]

$$S(N) = \frac{s \cdot f(N) + p \cdot g(N)}{s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N)} = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g \cdot N^{\alpha_g}}{c_h \cdot N^{\alpha_h}}} = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}} \tag{5}$$

and the following efficiency equation (for $N > 1$):

$$E(N) = \frac{S(N)}{N} = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{(\alpha_f + 1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h + 1)}} \tag{6}$$

The generic speedup equation given in (5) generalizes several well-known speedup equations and laws suggested in the literature (see Table 1).

4. Theoretical speedup and efficiency limits

4.1. Asymptotic speedup

As I am interested in asymptotic speedup, I determine limits for $N \to \infty$. I rewrite the generic speedup equation (5) as follows:

$$S(N) = \underbrace{\frac{s \cdot c_f \cdot N^{\alpha_f}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}}}_{(I)} + \underbrace{\frac{p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}}}_{(II)} \tag{7}$$

[2] The applicability of this speedup equation to $N = 1$ would require setting $c_h = 1$. In order to allow using an arbitrary coefficient in the power function $h(\cdot)$, I require N to be larger than 1.

Table 1
Instantiations of generic speedup model

Parameter values	Speedup equation	Speedup model or law
$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = \alpha_g = 0$	$S(N) = \frac{s+p}{s+\frac{p}{N}}$	*Amdahl's law* [4]
$c_f = c_g = c_h = \alpha_g = \alpha_h = 1, \alpha_f = 0$	$S(N) = \frac{s+p\cdot N}{s+p}$	*Gustafson's law* [19]
$c_h = \alpha_h = 1$	$S(N) = \frac{s\cdot c_f\cdot N^{\alpha_f}+p\cdot c_g\cdot N^{\alpha_g}}{s\cdot c_f\cdot N^{\alpha_f}+p\cdot c_g\cdot N^{(\alpha_g-1)}}$	*Scaled speedup model* [34] (under the assumption that the sequential and parallel workloads are given by power functions $c_f \cdot N^{\alpha_f}$ and $c_g \cdot N^{\alpha_g}$, resp.)
$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0$	$S(N) = \frac{s+p\cdot N^{\alpha_g}}{s+p\cdot N^{(\alpha_g-1)}}$	*Sun and Ni's law* [35, 36, 37] (under the assumption that the parallel workload is given by a power function N^{α_g})
$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g = \frac{1}{2}$	$S(N) = \frac{s+p\cdot N^{\frac{1}{2}}}{s+\frac{p}{N^{\frac{1}{2}}}}$	*Generalized scaled speedup model* [22]

For term (I), I yield the following limits (the proof can be obtained from equations (50)-(53) in Appendix A):

$$\lim_{N\to\infty}(I) = \begin{cases} 1, \ \alpha_f > \alpha_g - \alpha_h & (8) \\ \dfrac{s\cdot c_f}{s\cdot c_f + \frac{p\cdot c_g}{c_h}}, \ \alpha_f = \alpha_g - \alpha_h & (9) \\ 0, \ \alpha_f < \alpha_g - \alpha_h & (10) \end{cases}$$

For term (II), I yield the following limits (the proof can be obtained from equations (54) - (62) in Appendix A):

$$\lim_{N\to\infty}(II) = \begin{cases} \dfrac{p \cdot c_g}{s \cdot c_f},\ \alpha_g = \alpha_f, \alpha_h > 0 & (11) \\ \dfrac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}},\ \alpha_g = \alpha_f, \alpha_h = 0 & (12) \\ 0,\ \alpha_g < \alpha_f & (13) \\ \infty\ (\Theta(N^{\alpha_h})),\ \alpha_g > \alpha_f, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f \geq 0 & (14) \\ \infty\ (\Theta(N^{(\alpha_g - \alpha_f)})),\ \alpha_g > \alpha_f, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f < 0 & (15) \\ c_h,\ \alpha_g > \alpha_f, \alpha_h = 0 & (16) \end{cases}$$

Aggregating the above given limits for terms (I) and (II), yields the following limits for the speedup given in equations (5) and (7):

$$\lim_{N\to\infty} S(N) = \begin{cases} \dfrac{p\cdot c_g}{s\cdot c_f} + 1 = \dfrac{p\cdot c_g + s\cdot c_f}{s\cdot c_f},\ \alpha_g = \alpha_f, \alpha_h > 0 & (17) \\ (\Leftrightarrow 0 = \alpha_g - \alpha_f < \alpha_h) & \\ \dfrac{s\cdot c_f}{s\cdot c_f + \frac{p\cdot c_g}{c_h}} + \dfrac{p\cdot c_g}{s\cdot c_f + \frac{p\cdot c_g}{c_h}} = \dfrac{s\cdot c_f + p\cdot c_g}{s\cdot c_f + \frac{p\cdot c_g}{c_h}},\ \alpha_g = \alpha_f, \alpha_h = 0 & (18) \\ (\Leftrightarrow 0 = \alpha_g - \alpha_f = \alpha_h) & \\ 1 + 0 = 1,\ \alpha_g < \alpha_f & (19) \\ (\Leftrightarrow \alpha_g - \alpha_f < 0) & \\ \infty\ (\Theta(N^{\alpha_h})),\ \alpha_g > \alpha_f, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f \geq 0 & (20) \\ (\Leftrightarrow 0 < \alpha_h \leq \alpha_g - \alpha_f) & \\ \infty\ (\Theta(N^{(\alpha_g - \alpha_f)})),\ \alpha_g > \alpha_f, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f < 0 & (21) \\ (\Leftrightarrow 0 < \alpha_g - \alpha_f < \alpha_h) & \\ 0 + c_h = c_h,\ \alpha_g > \alpha_f, \alpha_h = 0 & (22) \\ (\Leftrightarrow 0 = \alpha_h < \alpha_g - \alpha_f) & \end{cases}$$

I now briefly discuss each of the six equations and refer to these as *speedup cases*; a visual illustration of the speedup cases can be retrieved from Figure 2.

Case A_S: The speedup limit given in eq. (17) represents an upper bound for $S(N)\ \forall N > 1$ (see Appendix A) and refers to situations in which the number N of available PUs affects the time required to address the parallelizable workload (due to $\alpha_h > 0$). While the speedup limit holds for any $c_h, \alpha_h > 0$, it seems reasonable to assume that $h(N) = c_h \cdot N^{\alpha_h} \geq 1$ holds as increasing the number of PUs from $N = 1$ to $N > 1$ should not lead to an increase of time required to execute the parallelizable workload. However, the speedup limit in this case does not depend on the values of c_h and α_h. Also, this case assumes that the scaling functions $f(N) = c_f \cdot N^{\alpha_f}$ and $g(N) = c_g \cdot N^{\alpha_g} = c_g \cdot N^{\alpha_f}$ change the serial and parallelizable workloads using the same factor N^{α_f}. It should be noticed that *case A_S* results in *Amdahl's law* [4] when setting $c_f = c_g = c_h = \alpha_h = 1, \alpha_f = \alpha_g = 0$; then, the limit on speedup amounts to $\frac{1}{s}$. As the speedup model of Amdahl's law is a special case of the memory-bound model

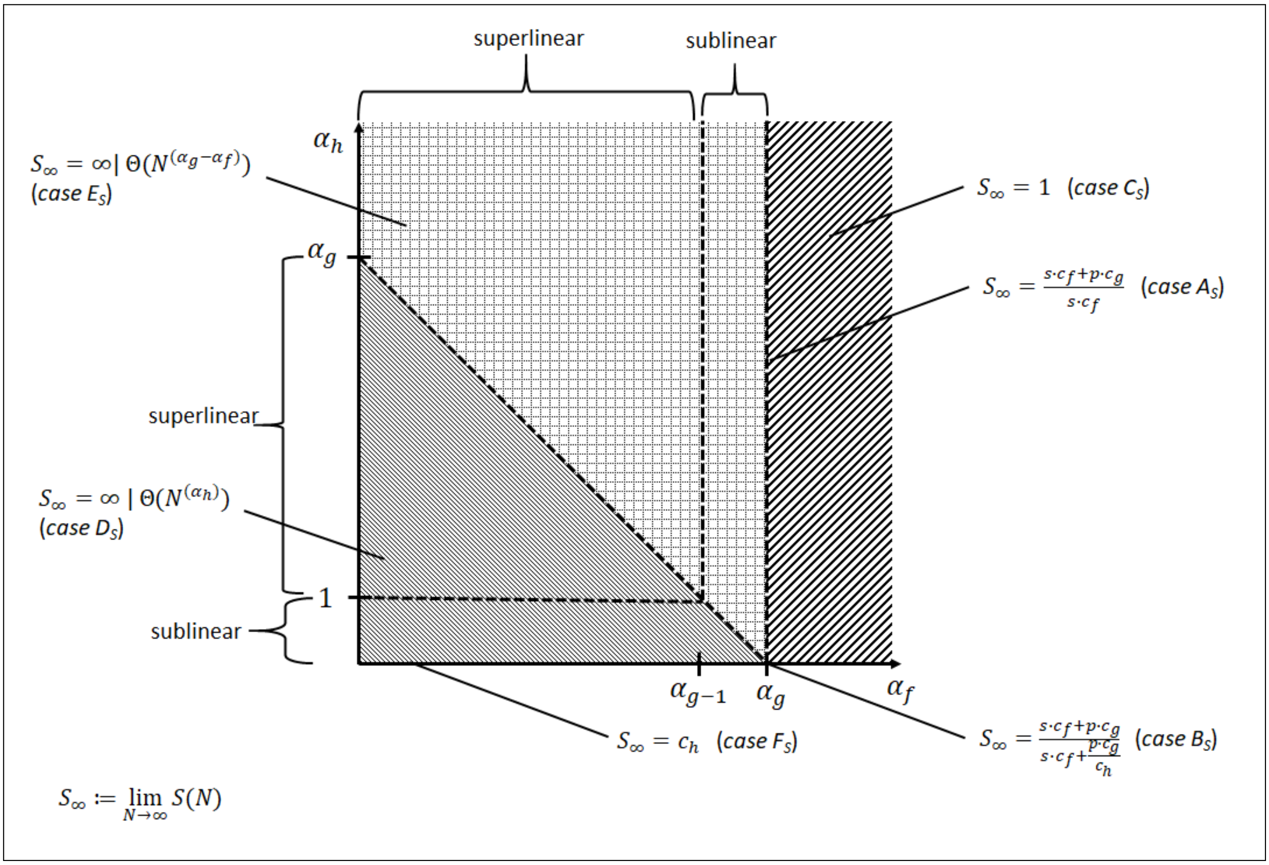
superlinear
sublinear
α_h
$S_\infty = \infty | \Theta(N^{(\alpha_g - \alpha_f)})$ (case E_S)
$S_\infty = 1$ (case C_S)
α_g
$S_\infty = \frac{s \cdot c_f + p \cdot c_g}{s \cdot c_f}$ (case A_S)
superlinear
$S_\infty = \infty \mid \Theta(N^{(\alpha_h)})$ (case D_S)
1
sublinear
α_{g-1}
α_g
α_f
$S_\infty = c_h$ (case F_S)
$S_\infty = \frac{s \cdot c_f + p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}$ (case B_S)
$S_\infty := \lim_{N \to \infty} S(N)$

Figure 2: Overview of speedup limits

suggested in [36, 37] (*Sun and Ni's law*) under the assumption that the parallelizable workload in the memory-bound model is given by a power function N^{α_g}, *case* A_S partly covers the abovementioned model. This case also (partly) covers the scaled workload model of Schmidt et al. [34] under the assumption that the sequential and parallelizable workloads are given by power functions $c_f \cdot N^{\alpha_f}$ and $c_g \cdot N^{\alpha_g}$, resp., with $\alpha_f = \alpha_g$.

Case B_S: The speedup limit given in eq. (18) represents an upper bound for $S(N)$ $\forall N > 1$ (see Appendix A). It refers to the same situation as described in *case* A_S with the modification that, here, N does not affect the time required to address the parallelizable workload (due to $\alpha_h = 0$); that time is rather modified through a division by the scalar c_h; i.e., when executing the parallelizable workload in parallel, the corresponding time changes are determined by a constant factor α_h. It seems reasonable to assume that $c_h \geq 1$ holds in this case (cmp. the analogeous discussion of *case* A_S), with a resulting speedup limit of $\lim_{N \to \infty} S(N) \geq 1$. While *case* B_S seems not useful under the premise that the parallelizable workload is infinitely parallelizable, it becomes useful when this assumption is replaced by the expectation that a given parallelizable workload can be executed in parallel only on a limited number N_{max} PUs; then, α_h may represent this limitation. For a discussion of limited parallelization, see, for example, [12, p. 772ff] and [3, p. 141ff]).

Case C_S: When the increase of serial workload is asymptotically higher than that of parallelizable workload ($\alpha_f > \alpha_g$), speedup converges against 1 (see eq. (19)) as a lower bound regardless of the values of c_h and α_h; i.e., in this case, any parallelization does not reduce the overall execution time asymptotically due to the “dominant"’ increase of the serial workload. Case C_S (partly) covers the scaled workload model of Schmidt et al. [34] under the assumption that the sequential and parallelizable workloads are given by power functions $c_f \cdot N^{\alpha_f}$ and $c_g \cdot N^{\alpha_g}$, resp., with $\alpha_f < \alpha_g$.

Case D_S: This case covers situations in which speedup is not limited and increases asymptotically with $\Theta(N^{\alpha_h})$ for (i) $\alpha_g > \alpha_f$, (ii) $\alpha_h > 0$, and (iii) $\alpha_g - \alpha_h \geq \alpha_f$ (see eq. (20)); (i) ensures that the parallelizable workload increases faster than the sequential workload, (ii) ensures that parallelization asymptotically reduces the time required to execute the parallelizable workload, and (iii) ensures that the temporal effect induced through the joint growth of the parallelizable workload and its actual parallel execution ($N^{(\alpha_g - \alpha_h)}$) is not weaker than the temporal effect induced through the growth of the sequential workload (N^{α_f}). Depending on the value of α_h, speedup asymptotically grows sublinearly ($0 < \alpha_h < 1$), linearly ($\alpha_h = 1$), or superlinearly ($\alpha_h > 1$). It should be noticed that *case D_S* results in *Gustafson’s law* [19] when setting $c_f = c_g = c_h = \alpha_g = \alpha_h = 1, \alpha_f = 0$; then, the speedup asymptotically grows linearly. *Case D_S* (partly) cover *Sun and Ni’s* law when setting $c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g \geq 1$ (under the assumption that the parallelizable workload in this model is given by a power function N^{α_g}). Finally, case D_S (partly) covers the scaled workload model of Schmidt et al. [34] under the assumption that the sequential and parallelizable workloads are given by power functions $c_f \cdot N^{\alpha_f}$ and $c_g \cdot N^{\alpha_g}$, resp., with $\alpha_g - \alpha_f \geq 1$.

Interestingly, *case D_S* may help explain superlinear speedup as is has been observed in research on mathematical optimization (at least for a limited range of N) [30, 7, 33, 17], for example.

Case E_S: Similar to *case D_S*, *case E_S* refers to situations in which speedup is not limited and increases asymptotically, but now with $\Theta(N^{(\alpha_g - \alpha_f)})$ for (i) $\alpha_g > \alpha_f$, (ii) $\alpha_h > 0$, and (iii) $\alpha_g - \alpha_h < \alpha_f$ (see eq. (21)). The conditions under which *case* E_S differ from those in *case* D_S only with regard to (iii); i.e., here, the temporal effect induced through the joint growth of the parallelizable workload and its actual parallel execution ($N^{(\alpha_g-\alpha_h)}$) is weaker than the temporal effect induced through the growth of the sequential workload (N^{α_f}). Now, the difference ($\alpha_g - \alpha_f$) determines the asymptotic growth of speedup: it asymptotically grows sublinearly ($0 < \alpha_g - \alpha_f < 1$), linearly ($\alpha_g - \alpha_f = 1$), or superlinearly ($\alpha_g - \alpha_f > 1$).

Similarly to *case* D_S, *case* E_S (partly) includes speedup models and laws suggested in the literature: *Case* E_S (partly) covers *Sun and Ni's law* when setting $c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g < 1$ (under the assumption that the parallelizable workload in this model is given by a power function N^{α_g}). With $\alpha_g = \frac{1}{2}$, Sun and Ni's model becomes the "generalized scaled speedup" model suggested in [22]; thus, *case* E_S also covers the generalized scaled speedup model. Finally, case E_S also (partly) includes the model of Schmidt et al. [34] under the assumption that the sequential and parallelizable workloads are given by power functions $c_f \cdot N^{\alpha_f}$ and $c_g \cdot N^{\alpha_g}$, resp., with $0 < \alpha_g - \alpha_f < 1$); then, speedup grows asymptotically sublinearly with $\Theta(N^{(\alpha_g-\alpha_f)})$.

As *case* D_S, also *case* E_S may help explain superlinear speedup.

Case F_S: This case covers situations in which speedup converges to c_h for (i) $\alpha_g > \alpha_f$ and (ii) $\alpha_h = 0$ (see eq. (22)). For $c_h \geq 1$, the limit c_h is a lower bound; for $c_h \leq 1$, the limit c_h is an upper bound. Condition (i) ensures that the parallelizable workload increases faster than the sequential workload, and with condition (ii) I assume that N does not affect the time required to address the parallelizable workload (due to $\alpha_h = 0$). As with *case* B_S, *case* F_S is useful with the expectation that a given parallelizable workload can be executed in parallel only on a limited number N_{max} of PUs.

4.2. Asymptotic efficiency

In order to determine theoretical efficiency limits, I proceed analogously to the determination of speedup limits. I rewrite the generic efficiency equation (eq. (6)) as follows:

$$E(N) = \underbrace{\frac{s \cdot c_f \cdot N^{\alpha_f}}{s \cdot c_f \cdot N^{(\alpha_f+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g-\alpha_h+1)}}}_{(I')} + \underbrace{\frac{p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{(\alpha_f+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g-\alpha_h+1)}}}_{(II')} \tag{23}$$

For term (I'), I yield the following limit (see equations (63) - (65) in Appendix B):

$$\lim_{N\to\infty}(I') = 0 \tag{24}$$

For term (II'), I yield the following limits (the proof can be obtained from equations (66) - (78) in Appendix B):

$$\lim_{N\to\infty}(II') = \begin{cases} 0,\ \alpha_f > \alpha_g - 1 & (25) \\ (\Leftrightarrow \alpha_g - \alpha_f < 1) & \\ 0,\ \alpha_f = \alpha_g - 1, 0 \le \alpha_h < 1 & (26) \\ (\Leftrightarrow 0 \le \alpha_h < \alpha_g - \alpha_f = 1) & \\ \dfrac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}},\ \alpha_f = \alpha_g - 1, \alpha_h = 1 & (27) \\ (\Leftrightarrow \alpha_g - \alpha_f = \alpha_h = 1) & \\ \dfrac{p \cdot c_g}{s \cdot c_f},\ \alpha_f = \alpha_g - 1, \alpha_h > 1 & (28) \\ (\Leftrightarrow \alpha_g - \alpha_f = 1 < \alpha_h) & \\ 0,\ \alpha_f < \alpha_g - 1, 0 \le \alpha_h < 1 & (29) \\ (\Leftrightarrow 0 < \alpha_h < 1 < \alpha_g - \alpha_f) & \\ c_h,\ \alpha_f < \alpha_g - 1, \alpha_h = 1 & (30) \\ (\Leftrightarrow 1 = \alpha_h < \alpha_g - \alpha_f) & \\ \infty\ (\Theta(N^{\alpha_g - \alpha_f - 1})),\ \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f > \alpha_g - \alpha_h & (31) \\ (\Leftrightarrow 1 < \alpha_g - \alpha_f < \alpha_h) & \\ \infty\ (\Theta(N^{\alpha_h - 1})),\ \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f \le \alpha_g - \alpha_h & (32) \\ (\Leftrightarrow 1 < \alpha_h \le \alpha_g - \alpha_f) & \end{cases}$$

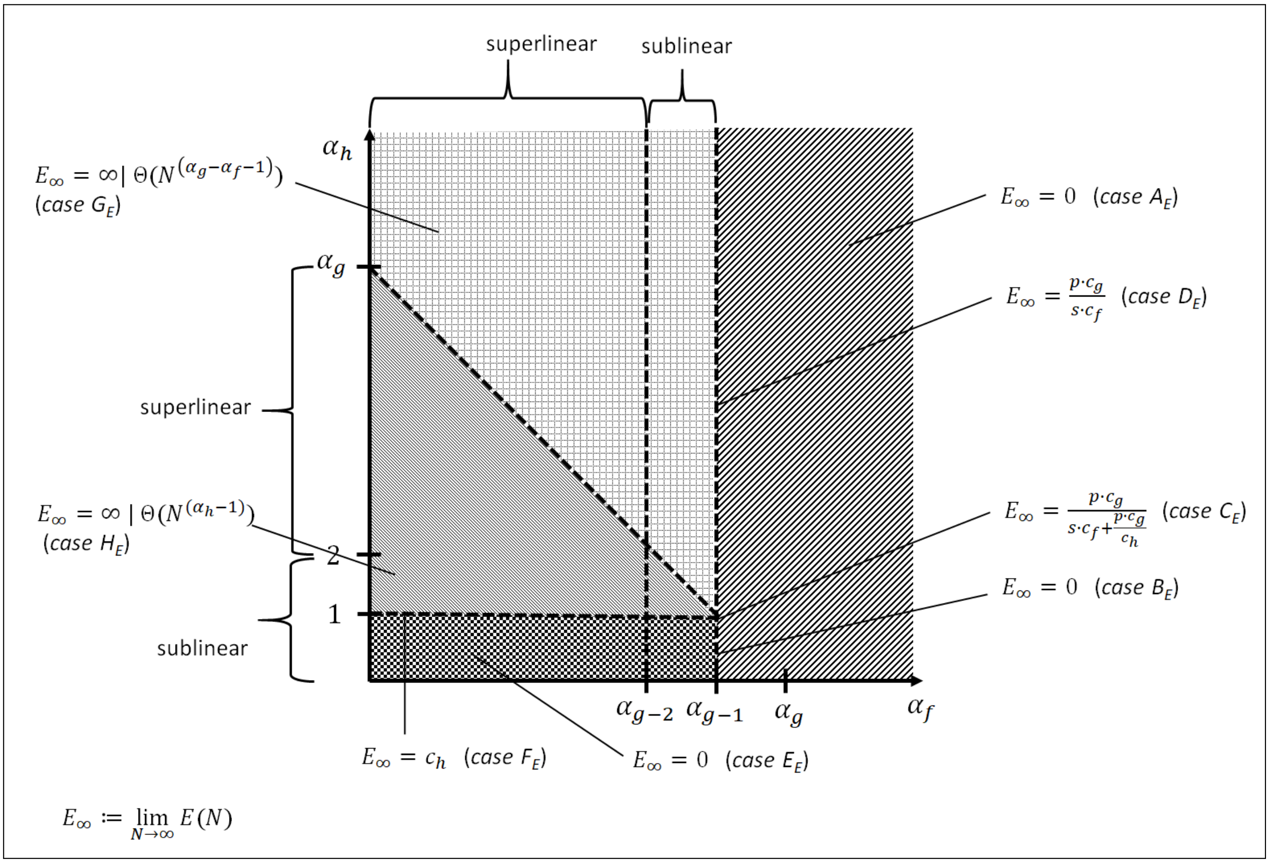
superlinear
sublinear
α_h
$E_\infty = \infty |\ \Theta(N^{(\alpha_g-\alpha_f-1)})$
(case G_E)
α_g
superlinear
$E_\infty = \infty \mid \Theta(N^{(\alpha_h-1)})$
(case H_E)
2
1
sublinear
$E_\infty = 0$ (case A_E)
$E_\infty = \frac{p\cdot c_g}{s\cdot c_f}$ (case D_E)
$E_\infty = \frac{p\cdot c_g}{s\cdot c_f+\frac{p\cdot c_g}{c_h}}$ (case C_E)
$E_\infty = 0$ (case B_E)
α_{g-2}
α_{g-1}
α_g
α_f
$E_\infty = c_h$ (case F_E)
$E_\infty = 0$ (case E_E)
$E_\infty \coloneqq \lim_{N\to\infty} E(N)$

Figure 3: Overview of efficiency limits

With $\lim_{N\to\infty}(I') = 0$, I yield the following limits for efficiency equations (6) and (23):

$$\lim_{N\to\infty} E(N) = \lim_{N\to\infty} \left[(I') + (II')\right] = \lim_{N\to\infty} (II') \tag{33}$$

I now briefly discuss each of the eight equations and refer to these as *(efficiency) cases*; a visual illustration of the efficiency cases can be retrieved from Figure 3 which, unsurprisingly, shows structural similarities with the visual representation of speedup limits due to the relationship between efficiency and speedup as given by $E(N) = \frac{S(N)}{N}$.

Case A_E: The efficiency limit given in eq. (25) equals zero regardless of the value of α_h and apparently represents a lower bound for $E(N)\ \forall N > 1$. This case refers to a situation in which the increase of the serial workload is asymptotically higher than that of a(n) (adjusted) parallelizable workload (adjusted based upon a decrease of the number of PUs by 1) ($\alpha_f > \alpha_g - 1$).

Case B_E: The efficiency limit given in eq. (26) equals zero when the value of α_h is upper bounded by 1. Again, it apparently represents a lower bound for $E(N)\ \forall N > 1$. This case refers to a situation in which the ratio of the increases of serial workload and adjusted parallelizable workload converges against the constant $\frac{c_f}{c_g}$ and in which the increase of time reduction of executing the parallelizable workload is sublinear in N ($\alpha_h < 1$).

Case C_E: The efficiency limit given in eq. (27) describes a situation that differs from that in *case* B_E only in that the increase of time reduction of executing the parallelizable workload is now linear in N ($\alpha_h = 1$). Then, the limit of efficiency is given by a constant larger than 0 assuming that the parallelizable workload is positive ($p > 0$).

Case D_E: The efficiency limit given in eq. (28) describes a situation that differs from that in *case* B_E only in that the increase of time reduction of executing the parallelizable workload is now superlinear in N ($\alpha_h > 1$). Despite this increase of time reduction and due to the relatively large increase of the serial workload compared to that of the parallelizable workload ($\alpha_f = \alpha_g - 1$), the limit of efficiency is still given by a constant (larger than 0) assuming that the parallelizable workload is positive ($p > 0$).

Case E_E: The efficiency limit given in eq. (29) describes a situation that is similar to that of *case* B_E. While $\alpha_h < 1$ holds again, the (adjusted and the non-adjusted) parallelizable workload grows faster than the serial workload ($\alpha_f < \alpha_g - 1$). However, as the increase of time reduction of executing the parallelizable workload is sublinear in N ($\alpha_h < 1$), efficiency converges against 0.

Case F_E: In contrast to *case* E_S, the efficiency limit given in eq. (30) describes a situation in which the increase of time reduction of executing the parallelizable workload is linear in N ($\alpha_h = 1$). Then, efficiency asymptotically equals a constant larger than 0 (assuming $p > 0$).

Case G_E: One situation in which efficiency is unbounded is described in eq. (31), where the (adjusted and the non-adjusted) parallelizable workload grows faster than the serial workload ($\alpha_f < \alpha_g - 1$) and the increase of time reduction of executing the parallelizable workload is superlinear in N ($\alpha_h > 1$). When also $\alpha_f > \alpha_g - \alpha_h$ holds, efficiency grows asymptotically with $\Theta(N^{\alpha_g - \alpha_f - 1})$; i.e., the asymptotic growth does not depend on α_h. For $\alpha_g - \alpha_f > 1$, $\alpha_g - \alpha_f = 1$ and $\alpha_g - \alpha_f < 1$, efficiency is superlinear, linear and sublinear, respectively.

Case H_E: A second situation in which efficiency is unbounded is described in eq. (32), where the (adjusted and the non-adjusted) parallelizable workload grows faster than the serial workload ($\alpha_f < \alpha_g - 1$) and the increase of time reduction of executing the parallelizable workload is superlinear in N ($\alpha_h > 1$). When also $\alpha_f \leq \alpha_g - \alpha_h$ holds, efficiency grows asymptotically with $\Theta(N^{\alpha_h - 1})$; i.e., the asymptotic growth does now depend on α_h. For $\alpha_h > 1$, $\alpha_h = 1$ and $0 < \alpha_h < 1$, efficiency is superlinear, linear and sublinear, respectively.

4.3. Asymptotic scalability

In the previous subsections, I identify *speedup cases* and *efficiency cases*. Now, I consider speedup cases and efficiency cases jointly, which results in various speedup-efficiency cases. I refer to these cases as *scalability cases*, which are defined by both speedup and efficiency limits (see Table 2 and Figure 4). I assign to each scalability case a scalability type, which describes both speedup (as first parameter) and efficiency (as second parameter), using the following semantics:

- β_h, γ_h: fixed values which depend on the reduced parallel workload scaling function h
- $\infty_{f,g}$: unbounded and monotonically increasing; the extent of increase depends on workload scaling functions f and g
- $\beta_{s,f,g}, \gamma_{s,f,g}$: fixed values which depend on the sequential part s (note: $p = 1 - s$) and the workload scaling functions f and g
- $\beta_{s,f,g,h}, \gamma_{s,f,g,h}$: fixed values which depend on the sequential part s (note: $p = 1 - s$), the workload scaling functions f and g, and the reduced parallel workload scaling function h
- ∞_h: unbounded and monotonically increasing; the extent of increase depends on reduced parallel workload scaling function h

Each scalability type refers to exactly one scalability case and set of conditions (see Table 2).

Table 2: Scalability cases

Cases			Limit values		Type	Conditions
Scalability	Speedup	Efficiency	Speedup*	Efficiency**		
A_{SC}	C_S	A_E	1	0	$(1/0)$	$\alpha_g - \alpha_f < 0$
B_{SC}	A_S	A_E	$\beta := \frac{s \cdot c_f + p \cdot c_g}{s \cdot c_f} > 1$	0	$(\beta_{s,f,g}/0)$	$0 = \alpha_g - \alpha_f < \alpha_h$
C_{SC}	B_S	A_E	$\beta := \frac{s \cdot c_f + p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}$	0	$(\beta_{s,f,g,h}/0)$	$0 = \alpha_g - \alpha_f = \alpha_h$
D_{SC}	F_S	E_E, B_E	$\beta := c_h$	0	$(\beta_h/0)$	$0 = \alpha_h < \alpha_g - \alpha_f$
E_{SC}	D_S	E_E, B_E	$\infty\,(\Theta(N^{\alpha_h}))$	0	$(\infty_h/0)$	$0 < \alpha_h < 1 \leq \alpha_g -$ α_f
F_{SC}		H_E		$\infty\,\left(\Theta(N^{(\alpha_h - 1)})\right)$	(∞_h/∞_h)	$1 < \alpha_h \leq \alpha_g - \alpha_f$
G_{SC}		C_E		$\gamma := \frac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}$	$(\infty_h/\gamma_{s,f,g,h})$	$\alpha_h = \alpha_g - \alpha_f = 1$
H_{SC}		F_E		$\gamma := c_h$	(∞_h/γ_h)	$1 = \alpha_h < \alpha_g - \alpha_f$
I_{SC}	E_S	G_E	$\infty\,\left(\Theta(N^{(\alpha_g - \alpha_f)})\right)$	$\infty\,\left(\Theta(N^{(\alpha_g - \alpha_f - 1)}\right)$	$(\infty_{f,g}/\infty_{f,g})$	$1 < \alpha_g - \alpha_f < \alpha_h$
J_{SC}		A_E		0	$(\infty_{f,g}/0)$	$0 < \alpha_g - \alpha_f <$ $\min\left\{1, \alpha_h\right\}$
K_{SC}		D_E		$\gamma := \frac{p \cdot c_g}{s \cdot c_f}$	$(\infty_{f,g}/\gamma_{s,f,g})$	$1 = \alpha_g - \alpha_f < \alpha_h$

*: Values for cases A_S and B_S are upper bounds, value for case C_S is lower bound, value for case F_S is lower bound ($c_h \geq 1$) or upper bound ($c_h \leq 1$).

**: Values for cases A_E to G_E are lower bounds, the value for case H_E is an upper bound (for sufficiently large values of N ($c_h \cdot N \geq 1$)).

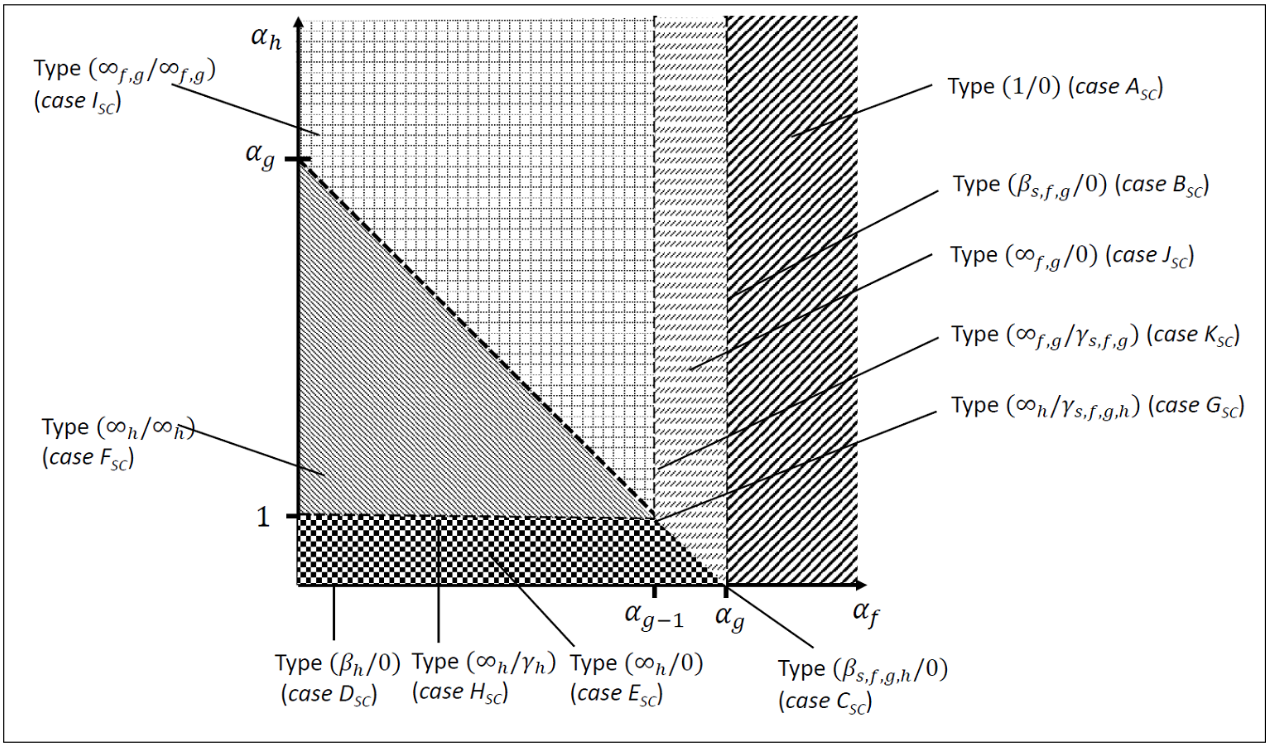
α_h
α_g
1
α_{g-1}
α_g
α_f
Type $(\infty_{f,g}/\infty_{f,g})$ (*case* I_{SC})
Type $(1/0)$ (*case* A_{SC})
Type $(\beta_{s,f,g}/0)$ (*case* B_{SC})
Type $(\infty_{f,g}/0)$ (*case* J_{SC})
Type $(\infty_{f,g}/\gamma_{s,f,g})$ (*case* K_{SC})
Type $(\infty_h/\gamma_{s,f,g,h})$ (*case* G_{SC})
Type (∞_h/∞_h) (*case* F_{SC})
Type $(\beta_h/0)$ (*case* D_{SC})
Type (∞_h/γ_h) (*case* H_{SC})
Type $(\infty_h/0)$ (*case* E_{SC})
Type $(\beta_{s,f,g,h}/0)$ (*case* C_{SC})

Figure 4: Overview of scalability cases

For the discussion of speedup, efficiency and scalability results derived in the preceding section, I recall the meaning of parameters α_f, α_g and α_h since their values determine the (speedup, efficiency and scalability) case of a particular parallel algorithm: The parameters α_f, α_g and α_h affect the serial workload, the parallelizable workload and the actual reduction of parallelizable workload through parallelization, respectively, depending on the number of PUs N; they are given by $f(N) := c_f N^{\alpha_f}$, $g(N) := c_g N^{\alpha_g}$ and $h(N) := c_h N^{\alpha_h}$, respectively. I also recall the abovementioned speedup and efficiency equations:

$$S(N) = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}} \text{ (see eq. (5))}$$

$$E(N) = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{(\alpha_f+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h + 1)}} \text{ (see eq. (6))}$$

I now discuss each of the eleven scalability cases A_{SC} to K_{SC}. As the definition of scalability cases (and types) is based upon combinations of speedup cases and efficiency cases, the characteristics of scalability cases (and types) can be derived from the above descriptions of the characteristics of speedup and efficiency cases.

Case A_{SC}: With speedup case C_S, the increase of serial workload is asymptotically higher than that of parallelizable workload ($\alpha_f > \alpha_g$). Then, speedup converges against 1. Case C_S induces efficiency case A_E so that the resulting asymptotic efficiency equals 0. The scalability type is $(1, 0)$. Overall, parallelization does not scale at all and parallelization efforts do not make much sense.

Case B_{SC}: With speedup case A_S, the scaling functions f and g change the serial and parallelizable workloads using the same factor N^{α_f} with possibly different values c_f and c_g; furthermore, case A_S refers to situations in which the number N of available PUs affects the time required to address the parallelizable workload (due to $\alpha_h > 0$). Then, speedup converges against a constant $\beta_{s,f,g} = \frac{s \cdot c_f + (1-s) \cdot c_g}{s \cdot c_f} > 1$. Speedup case A_S leads to efficiency case A_E; i.e., efficiency converges against zero. Scalability case B_{SC}, which refers to scalability type $(\beta_{s,f,g}, 0)$, includes *Amdahl's law* [4] and partly *Sun and Ni's law* [36, 37] (see the discusssion of speedup case A_S).

Case C_{SC}: This scalability case is similar to the scalability case B_{SC} and differs from it only as α_h equals zero; i.e., N does not affect the time required to address the parallelizable workload. With speedup case B_S and resulting efficiency case A_E, the associated scalability type is $(\beta_{s,f,g,h}, 0)$, with speedup $\beta_{s,f,g,h} = \frac{s \cdot c_f + (1-s) \cdot c_g}{s \cdot c_f + \frac{(1-p) \cdot c_g}{c_h}}$. As discussed above, speedup case B_S, and thus scalability case C_{SC}, are not useful under the premise that the parallelizable workload is infinitely parallelizable, but it becomes useful when a given parallelizable workload can be executed in parallel only on a limited number of PUs.

Case D_{SC}: Scalability case D_{SC}, which corresponds to scalability type $(\beta_h/0)$, includes speedup case F_S, in which the parallelizable workload increases faster than the sequential workload ($\alpha_g > \alpha_f$) and N does not affect the time required to address the parallelizable workload ($\alpha_h = 0$). Then, speedup converges to c_h. When speedup case F_S applies, either efficiency case E_E or B_E applies with efficency converging towards zero in both cases. As with scalability case C_{SC}, case D_{SC} is useful with the expectation that a given parallelizable workload can be executed in parallel only on a limited number of PUs.

Case E_{SC}: This scalability case refers to a situation in which (i) the parallelizable workload increases at least one magnitude faster than the sequential workload ($\alpha_g \geq \alpha_f + 1$) and (ii) the number of available PUs affects the time required to address the parallelizable workload with unlimited and sublinear growth ($0 < \alpha_h < 1$). This scalability case is linked to speedup case D_S and one of the efficiency cases E_E or B_E, resulting to scalability type $(\infty_h, 0)$. Case E_{SC} involves a speedup growth of $\Theta(N^{\alpha_h})$; i.e., speedup convergence is determined by the reduced parallel workload scaling function h. Due to the condition $\alpha_h < 1$, this growth is sublinearly in N and efficiency converges towards zero.

Case F_{SC}: This case describes to a situation in which the parallelizable workload increases more than one magnitude faster than the sequential workload ($\alpha_g \geq \alpha_f + 1$) and the number of available PUs affects the time required to address the parallelizable workload with superlinear growth ($1 < \alpha_h \leq (\alpha_g - \alpha_f)$)). Under such conditions, speedup case D_S and efficiency case H_E apply, resulting in the scalability type (∞_h, ∞_h), where both speedup and efficiency are unlimited, speedup grows superlinearly with $\Theta(N^{\alpha_h})$, and efficiency grows sublinearly (when $1 < \alpha_h < 2$), linearly (when $\alpha_h = 2$), or superlinearly (when $2 < \alpha_h$).

Case G_{SC}: This scalability case describes a situation in which the parallelizable workload increases one magnitude faster than the sequential workload ($\alpha_g = \alpha_f + 1$) and the number of available PUs affects the time required to address the parallelizable workload with linear growth ($\alpha_h = 1$). Under such conditions, speedup case D_S and efficiency case C_E apply, resulting in the scalability type $(\infty_h, \gamma_{s,f,g,h})$, where speedup is unlimited and grows linearly and where efficiency converges against a constant γ that depends upon functions f, g, h and upon s ($\gamma_{s,f,g,h} = \frac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}$).

Scalability case G_{SC} covers *Gustafson's law* [19], where $\alpha_g = \alpha_h = 1, \alpha_f = 0$. It also (partly) covers the model of Schmidt et al. [34].

Case H_{SC}: Scalability case H_{SC} differs from scalability case G_{SC} only in the regard that the parallelizable workload increases more than one magnitude faster than the sequential workload ($\alpha_g > \alpha_f + 1$). Similiar to case G_{SC}, the scalability type is (∞_h, γ_h) but here γ_h is set to c_h (efficiency case F_E).

Analogously to scalability case G_{SC}, also case H_{SC} (partly) covers the model of Schmidt et al. [34]. In addition, case H_{SC} also (partly) covers *Sun and Ni's law* when $c_f = c_g = c_h = 1$ holds and when the parallelizable workload in this model is given by a power function N^{α_g}.

Case I_{SC}: This scalability case applies when (i) the parallelizable workload increases at least one magnitude faster than the sequential workload ($\alpha_g > \alpha_f + 1$) and (ii) the number of available PUs affects the time required to address the parallelizable workload with unlimited and superlinear growth ($1 < \alpha_g - \alpha_f < \alpha_h$). Under these conditions, speedup case E_S and efficiency case G_E apply, leading to scalability type $(\infty_{f,g}, \infty_{f,g})$; i.e., both speedup and efficiency are unbounded and grow superlinearly.

Case J_{SC}: The conditions under which this case apply differ from those of case I_{SC} in that the parallelizable workload increases less than one magnitude faster than the sequential workload ($0 < \alpha_g - \alpha_f < 1$). Then, speedup case E_S and efficiency case A_E apply, leading to scalability type $(\infty_{f,g}, 0)$; i.e., while speedup is unbounded and grow sublinearly, efficiency converges against zero.

With $\alpha_f = 0, \alpha_h = 1$, it partly covers *Sun and Ni's law*, and with $\alpha_f = 0, \alpha_g = \frac{1}{2}, \alpha_h = 1$, it covers the speedup model of Juurlink and Meenderinck [22].

Case K_{SC}: When (i) the parallelizable workload increases one magnitude faster than the sequential workload ($\alpha_g = \alpha_f + 1$) and (ii) the number of available PUs affects the time required to address the parallelizable workload with unlimited and superlinear growth ($1 < \alpha_h$), scalability case K_{SC} applies with speedup case E_S and efficiency case D_E; then, speedup is unlimited and grows linearly, efficiency converges against a constant $\gamma = \frac{p \cdot c_g}{s \cdot c_f}$, and scalability type $(\infty_{f,g}, \gamma_{s,f,g})$ applies.

5. Computational experiments

I demonstrate the application of the generic speedup and efficiency model and the scalability typology with two computational experiments. The first experiment targets strong scalability and performs parallel matrix multiplication with fixed workload (in terms of matrix sizes) in the Amdahl setting. The second experiment is more sophisticated and targets lower-upper (LU) decomposition as factorization of a matrix as the product of a lower triangular matrix and an upper triangular matrix; here I consider a variable workload (scaled-size model) that increases with the number of available PPUs. Both types of tasks occur in many problems in numerical analysis and linear algebra. The description of these experiments is intended to illustrate the proposed model and to give examples of how the polynomial functions of the generic speedup and efficiency model can be determined in practice.

I ran all experiments on a compute node (2x AMD Milan 7763, 2.45 GHz, up to 3.5 GHz, 2x 64 cores, 256 GiB main memory) of the HPC cluster NOCTUA2 provided by the Paderborn Center for Parallel Computing of Paderborn University[3]. The source code was written in C++, using the OpenMP application programming interface for parallelization, and compiled with GCC (version 12.2.0)[4].

5.1. Parallel matrix multiplication with fixed workload

I perform parallel matrix multiplication without any optimization, i.e. rows and columns of the first and second matrices are multiplied in pairs (scalar product). In my speedup and efficiency model I set $c_f = c_g = c_h = \alpha_h = 1, \alpha_f = \alpha_g = 0$. This setting results in the following speedup and efficiency equations:

$$S(N) = \frac{1}{s + \frac{p}{N}}, \quad E(N) = \frac{1}{N \cdot s + p} \tag{34}$$

This instantiation of the generic speedup and efficiency model corresponds to scalability class B_{SC} and scalability type ($\beta_{s,f,g} = 1/s, 0$) (see Table 2); i.e., asymptotic speedup and asymptotic efficiency are given by $1/s$ and 0, respectively. I set the sizes of the two integer matrices A and B to $[1.28 \cdot 10^{10}, 300]$ and $[200, 1.28 \cdot 10^{10}]$, respectively, and initialized both matrices with random values in the interval $[-1000, 1000]$. The large sizes of the matrices are motivated by the fact that parallelization may then be useful. I also set the number of rows to a multiple of the maximum number of cores available on the target machine (128), to allow an equal distribution of the parallelizable workload when 128 (or lower powers of 2) cores are used. This justifies setting $h(N) := N$. I determined the sequential and parallelizable workloads $s = 0.023595$ and $p = 0.976405$, respectively, by executing the code on a single core. Not surprisingly, in a matrix multiplication setting, the serial workload is quite small. Note, however, that even for a value as low as $s = 0.023595$, the speedup is limited by $1/s \approx 42$.

Figure 5a shows that the theoretical speedup, which considers the existence of a serial part $s > 0$ but ignores any computational overhead induced by parallelization, converges to the speedup limit given by $1/s \approx 42$. The computational speedup, given by $T(1)/T(N)$ for any N, shows the actual speedup achieved. The gap between the two speedup lines is due to the overhead of parallelization. Figure 5b shows both the theoretical and the computational efficiency, defined as the ratios of the above speedup values and the

[3] `https://pc2.uni-paderborn.de/hpc-services/available-systems/noctua2`

[4] I used the following compiler options: *-march=native -m64 -fPIC -O3 -fopenmp*

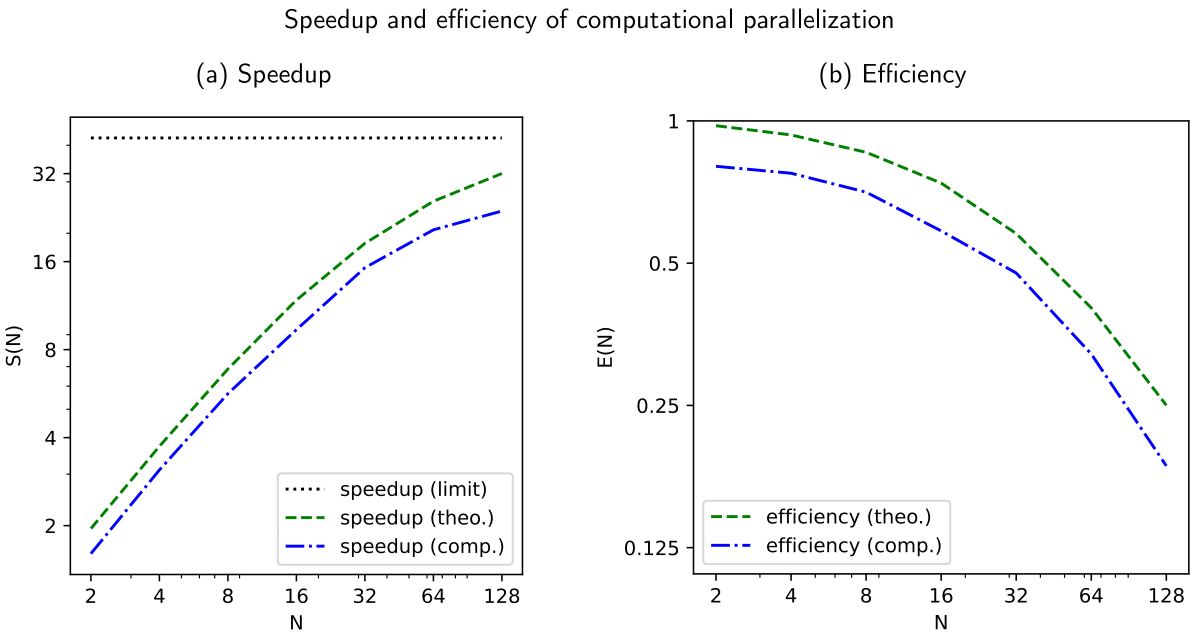

Figure 5: Computational results of matrix multiplication with a fixed workload

number of parallel PUs N. While the decrease of the theoretical efficiency and its convergence to 0 is due to the presence of a serial part $s > 0$, the gap between the two efficiency curves is, as in the speedup case, due to the computational overhead induced by parallelization. The data for this experiment can be found in Table 4 in Appendix C.1.

5.2. Parallel LU decomposition with variable workload

I perform LU decomposition of an matrix without pivoting. A description of the algorithm is shown in Figure 6. The LU decomposition algorithm is only partially parallel. The outer loop (iteration over i) cannot be parallelized, since computations in iteration i $(i > 1)$ require computations in iteration $(i-1)$ to be completed. In contrast, the inner loops (iterating over j and l) allow parallelization. In my implementation, the first inner loop (iteration over j) is parallelized. To account for variable workload, I assume that the number of rows in the matrix A grows linearly with the number of PPUs.[5].

As in the first experiment, I assume that the serial workload remains constant as the number of PPUs varies; i.e., I assume $f(N) = 1$ $(N \geq 1)$. However, unlike the setting in the first experiment, the workload size now depends on the number of parallel PPUs N; i.e., one must determine an appropriate polynomial function $g(N) = c_g \cdot N^{\alpha_g}$. It is also reasonable to assume that the parallelizable workload $p \cdot g(N)$ cannot be partitioned into N independent sub-workloads of equal size as the number of iterations in the j-loop is

[5] Obviously, the size of A in terms of the number of entries grows quadratically with the number of available PPUs.

```
1.  Input: $A \in \mathbb{N}^{z \times z}$, $\mathbb{N} := \{1, 2, \dots\}$
2.  $L \leftarrow$ identity matrix $I_n$
3.  $U \leftarrow A$
4.  for $i = 1$ to $z - 1$ do
5.    for $j = i + 1$ to $z$ do
6.      $L_{j,i} \leftarrow \frac{U_{j,i}}{U_{i,i}}$
7.      for $l = i + 1$ to $z$ do
8.        $U_{j,l} \leftarrow U_{j,l} - L_{j,i} \times U_{i,l}$
9.      end for
10.     for $l = 1$ to $i$ do
11.       $U_{j,l} \leftarrow 0$
12.     end for
13.   end for
14. end for
15. Output: $L, U \in \mathbb{N}^{z \times z}$, $A = L \cdot U$
```

Figure 6: LU decomposition algorithm (without pivoting)

usually not a multiple of the number of available PPUs; i.e., the workload is not “perfectly” parallelizable and one needs to determine an appropriate polynomial function $h(N) = c_h \cdot N^{\alpha_h}$. I use analytical and numerical approaches to determine g and h.

To determine g, I first determine the number of (basic) calculations $\hat{g}(z)$ (as performed in lines 6, 8, and 11 of the LU decomposition algorithm shown above) as a function of the number of matrix rows z. As shown in Appendix C.2, one obtains

$$\hat{g}(z) = \frac{1}{3}\left(z^3 - z\right) \tag{35}$$

Assuming that a calculation requires t_c units of computation time, the execution of (lines 4-14 of) the LU decomposition algorithm requires $\hat{g}(z) \cdot t_c$ units of computation time. As I assume that the number of rows/columns z of the matrix A (“problem size”) grows linearly with the number of PPUs N, z is given by $z = z_1 \cdot N$, where z_1 is the number of rows when only a single processing unit is available. This leads to

$$\hat{g}(z) = \hat{g}(N, z_1) = \hat{g}(z_1 \cdot N) = \frac{1}{3}\left(\left(z_1 \cdot N\right)^3 - \left(z_1 \cdot N\right)\right) = \frac{1}{3}\left(z_1^3 \cdot N^3 - z_1 \cdot N\right) \tag{36}$$

For $N = 1$ we get

$$\hat{g}(1, z_1) = \frac{1}{3}\left(z_1^3 - z_1\right), \tag{37}$$

which is the number of calculations when only a single processor is available (i.e., $N = 1$). I can now determine a normalized workload scaling function $\overline{g}_{z_1}$ given by

$$\overline{g}_{z_1}(N) := \frac{\hat{g}(N, z_1)}{\hat{g}(1, z_1)} = \frac{z_1^3 \cdot N^3 - z_1 \cdot N}{z_1^3 - z_1} = \frac{z_1^3}{z_1^3 - z_1} \cdot N^3 - \frac{z_1^3}{z_1^3 - z_1} \cdot N = \Theta(N^3) \tag{38}$$

Note that $\overline{g}$ is normalized as $\overline{g}_{z_1}(1) = 1$. For the analysis of the asymptotic speedup, I do not need to consider the linear term any further, so I use as workload scaling function

$$g_{z_1}(N) = \frac{z_1^3}{z_1^3 - z_1} \cdot N^3 \tag{39}$$

Using the generic representation of the parallel workload scaling function $g(N) = c_g \cdot N^{\alpha_g}$, I get $c_g = z_1^3 / \left(z_1^3 - z_1\right)$ and $\alpha_g = 3$.

I now determine the scaling function h, which describes the extent to which the execution time of the parallelizable workload $p \cdot g_{z_1}(N)$ can theoretically be reduced by parallelization. While the outer loop cannot be parallelized (see explanation above), the first inner loop (iteration over variable j) can be parallelized because the computations in the iterations do not affect each other. Assuming that this loop is parallelized, $(z - i)$ iterations can be executed in parallel, for each $i = 1, \dots, (z - 1)$. Suppose $z = 128$ and $N = 64$ PPUs are available. Then, for example, in the 63th iteration of the outer loop ($i = 63$), $(z - i) = 65$ iterations of the first inner loop ($j = 64, \dots, 128$) can be distributed to $N = 64$ PPUs. While 64 of the 65 iterations can be distributed evenly across $N = 64$ PPUs, the remaining iteration is executed on a single PPU while the other 63 PPUs remain idle. Obviously, the time required to execute the workload of this iteration of the outer loop is not divided by the number of available PPUs $N = 64$. Therefore, $h(N) < N$ $(N > 1)$.

The proof of equation (35) (see Appendix C.2) makes use of the fact that the number of calculations of the parallelizable part is given by

$$\hat{g}(z) = \sum_{i=1}^{z-1} \left[(z - i) \cdot (z - i + 1)\right]. \tag{40}$$

Note that parallelizing the first inner loop means that $(z-i)$ blocks (of $(z-i+1)$ calculations each) can be executed in parallel by N PPUs. Following my explanations of parallelization above, using N PPUs requires a computation time equal to the computation time required in a serial execution for $\hat{g}_{reduced}(z)$ calculations, with

$$\hat{g}_{reduced}(z) = \sum_{i=1}^{z-1} \left[\left\lceil \frac{(z-i)}{N} \right\rceil \cdot (z-i+1) \right]. \tag{41}$$

Assuming again that $z = N \cdot z_1$, then the computation time of the parallelizable workload executed on a single PPU is reduced by using N PPUs by the factor $\hat{h}(z) = \hat{h}(N, z_1)$ given by

$$\hat{h}(N, z_1) = \hat{h}(z) = \frac{\hat{g}(z)}{\hat{g}_{reduced}(z)} = \frac{\hat{g}(N, z_1)}{\hat{g}_{reduced}(N, z_1)}. \tag{42}$$

Since it is difficult to find a closed-form expression of $\hat{g}_{reduced}$ and to derive the function $\hat{h}$ from it analytically, one could use a statistical approach to derive, for a given z_1, a polynomial function $h(N, z_1) = c_h \cdot N^{\alpha_h}$ ($c_h > 0, \alpha_h \geq 0$) that approximates $\hat{h}$. For $z_1 \in \{10^i,\ i \in \{2,3,4,5\}\}$, Tables 5-8 in Appendix C.2 show data points for $N = 2^i,\ i \in \{1, \dots 20\}$. Note that z_1 is the number of rows/columns of the matrix to be decomposed when only a single PU is available, and that c_h and α_h must be determined depending on z_1.

The computations show that for any value of z_1, the value of $\hat{h}(N, z_1)$ is close to N. It is reasonable to assume that, for any z_1, as N goes to infinity, $(\hat{h}(N, z_1) - N)$ converges to 0, or $(\hat{h}(N, z_1)/N)$ converges to 1, although I do not prove this here. However, from an asymptotic point of view, for any z_1 and any $c_h > 0, \alpha_h < 1$, $h(N, z_1) = c_h \cdot N^{\alpha_h}$ would converge to 0. Therefore, I do not need to do a statistical analysis here and set $c_h = \alpha_h = 1$; i.e., $h(N, z_1) = h(N) = N$.

In summary, the scaling functions for the given LU decomposition algorithm are given by

$$\begin{aligned}
f(N) &= 1 && (c_f = 1, \alpha_f = 0) \\
g_{z_1}(N) &= \frac{z_1^3}{z_1^3 - z_1} \cdot N^3 && \left(c_g = \frac{z_1^3}{z_1^3 - z_1}, \alpha_g = 3 \right) \\
h(N) &= N && (c_h = \alpha_h = 1),
\end{aligned} \tag{43}$$

which yields as speedup equation (cmp. equation (5))

$$S_{z_1}(N) = \frac{s + p \cdot \frac{z_1^3}{z_1^3 - z_1} \cdot N^3}{s + p \cdot \frac{z_1^3}{z_1^3 - z_1} \cdot N^2} \tag{44}$$

Based on the proposed scalability typology shown in Table 2, the LU decomposition algorithm is of scalability case H_{SC} with scalability type (∞_h, γ_h); the speedup increases asymptotically with $\Theta(N)$ $(\lim_{N\to\infty}(S_{z_1}(N) - N) = 0)$ and the efficiency converges to $c_h = 1$. Note that these results do not depend on the value of z_1.

In the computational experiment, I initialize the matrix to be decomposed with (random and integer) values in the interval $[1, 1000]$ to avoid any numerical problems, since the LU decomposition does not involve any pivoting. I set $z_1 := 100$; thus, the number of rows and columns of the matrix to be decomposed is given by $z = N \cdot z_1 = 100 \cdot N$ $(N = 1, 2, 4, \dots, 128)$. The first experiments showed that, contrary to the theoretical analysis, the speedup tends to converge to 1 with increasing values of N. This effect is due to the phenomenon that the amount of work done in the parallel region (lines 6-12 in the pseudocode shown in Figure 6) is small relative to the overhead of creating and managing threads. To avoid this effect, I extended the LU decomposition algorithm to decompose a set of m matrices A_d $(d = 1, \dots, m)$ of equal size; this can be easily implemented by running the code in lines 6-12 for each of the m matrices, so that the workload in the parallel region increases by a factor of m. Note that from a theoretical point of view, this modification does not affect the speedup and efficiency bounds, while it allows to demonstrate the speedup increase with increasing values of N in computational experiments. A second problem arose when determining the sequential and parallelizable fractions of the total execution time s and p, respectively: since the matrix size (in terms of the number of rows/columns) grows linearly with N, and the number of computations to be performed in the LU decomposition algorithm grows asymptotically with $\Theta(n^3)$ (see equation (38)), the matrices to be decomposed become large. When only a single PU is used, the matrix to be decomposed must be relatively small in order to perform the experiments in a reasonable amount of time. As a consequence, the total execution time for decomposing a (100×100) matrix is small (11 ms), so the serial time s is 0 when traced computationally, since the numerical accuracy is too low. However, when s is close to 0, it hardly affects the theoretical (serial and parallel) computation times and speedups (see equation (44)); it also does not affect the speedup and efficiency bounds. Since in every parallel program at least a small part of the code is executed serially, I set $s := 0.01$.

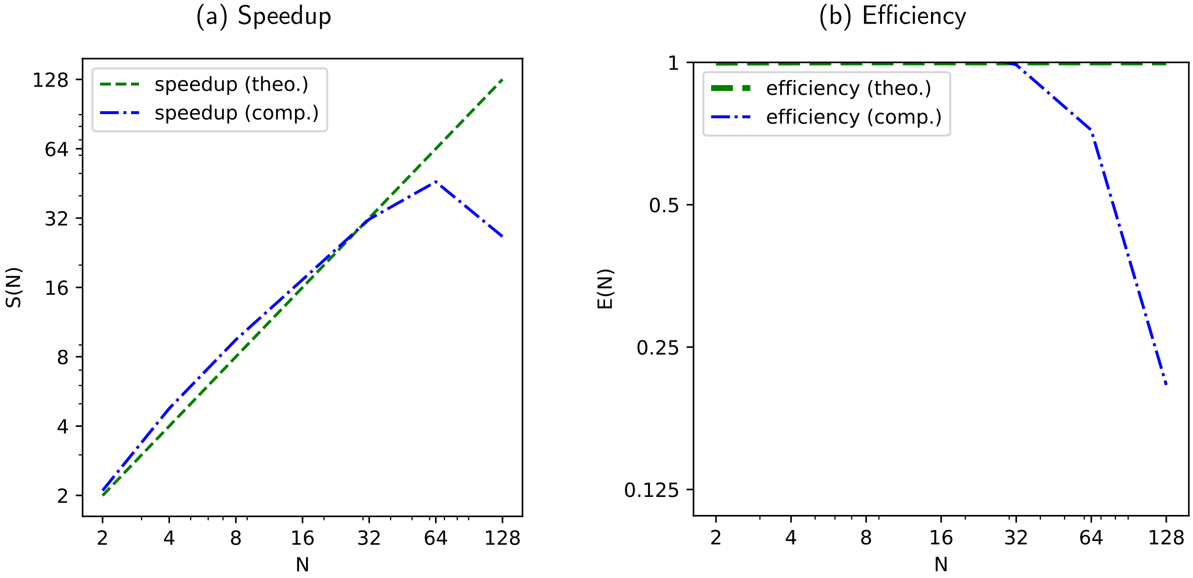

Figure 7: Computational results of LU matrix decomposition with variable workload

Figure 7 shows the speedup and efficiency achieved in the computational experiment. As suggested by the theoretical analysis, the speedup increases linearly and the efficiency is close to 1 (using up to $N = 32$ threads). Since both the serial and parallel execution times are relatively small (see the figures in Table 9 in Appendix C.2), the computational speedup seems to be slightly above the theoretical speedup (for $N = 2, 4, 8, 16$) due to numerical problems in determining execution times on the cluster with sufficient accuracy. For $N = 64$ the speedup still increases, but at a much lower rate (speedup ≈ 46) with an efficiency of about 0.72; for $N = 128$ the speedup even decreases (speedup ≈ 26.61) with an efficiency of about 0.21. It seems reasonable to expect that for $N > 128$ both speedup and efficiency decrease further. I did not investigate these effects because the computing node available for the experiments has a maximum of 128 cores. Obviously, and as expected, for an empirically large number of PPUs, theoretical speedup and efficiency bounds become much looser for computations. Therefore, in the next section, I discuss how the impact of parallelization overhead can be taken into account in future research when looking for tighter bounds.

6. Discussion

6.1. Application of model and typology

The scalability typology developed in the previous section allows researchers to determine the limits of speedup and efficiency of their applications and the extent to which computational parallelization scales for their needs. They also support researchers regarding their decision of how many parallel PUs to use in the presence of economic budget constraints. My typology shown in Table 2 provides a more comprehensive picture of scalability in homogeneous computing environments than speedup laws suggested in the literature (shown in Table 1), thereby widening the scope of applying scalability insights. At the same time, my typology includes all of the abovementioned speedup laws as illustrated in Table 3. In particular, Amdahl's law and Gustafson's law are consistent with our classification. These laws have been discussed in the literature as two different types of scaling: *strong scaling* focuses on the "Amdahl setting", where the total problem size remains fixed as more processors are added, and the goal is to run the same problem size faster. In contrast, *weak scaling* focuses on the "Gustafson setting", where the problem size per processor remains fixed as more processors are added, the total problem size is linear in the number of processors used, and the goal is to run larger problems in the same amount of time [5].

A key issue for researchers is the assignment of their particular application to a scalability type, which requires determining the sequential workload s and the power functions f, g, h (see Definition 4). In order to determine s, a straightforward approach is to execute the application on a single PU and measure the execution times t_{seq} ans t_{par} of the sequential and parallelizable workloads, resp., leading to $s = \frac{t_{seq}}{t_{seq}+t_{par}}$ and $p = 1 - s$. I used this approach in the two computational experiments. As the second experiment shows, this approach can lead to inaccurate results when the total execution time is small and numerical problems occur. As Table 2 shows, the value of s affects speedup and efficiency bounds in some scalability cases. Under such circumstances, it may be helpful to use "safe" lower and upper bounds on s, and to define intervals of speedup and efficiency bounds.

The determination of the power functions f, g and h can be much more challenging, depending on the algorithm used. While in the first experiment the functions could be determined straightforward, the second experiment shows that analytical, numerical and statistical approaches may be required to obtain reliable estimates of the functions. The application of such approaches may become quite tedious or even impossible due to the complexity of the parallelized algorithm. In this case, researchers are advised to use computational

Table 3
Scalability types of speedup models suggested in the literature

Speedup model	Scalability case	Scalability type	Conditions
Amdahl's law [4]	B_{SC}	$(\beta_{s,f,g} = \frac{1}{s}, 0)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = \alpha_g = 0$
Gustafson's law [19]	G_{SC}	$(\infty_h, \gamma_{s,f,g,h} = 1 - s)$	$c_f = c_g = c_h = \alpha_g = \alpha_h = 1, \alpha_f = 0$
Generalized scaled speedup model [22]	J_{SC}	$(\infty_{f,g}, 0)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g = \frac{1}{2}$
Sun and Ni's law [36, 37]	B_{SC}	$(\beta_{s,f,g} = \frac{1}{s}, 0)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g = 0$
	G_{SC}	$(\infty_h, \gamma_{s,f,g,h} = 1 - s)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g = 1$
	H_{SC}	$(\infty_h, \gamma_h = 1)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, \alpha_g > 1$
	J_{SC}	$(\infty_{f,g}, 0)$	$c_f = c_g = c_h = \alpha_h = 1, \alpha_f = 0, 0 < \alpha_g < 1$
Scaled speedup model [34]	A_{SC}	$(1, 0)$	$c_h = \alpha_h = 1, \alpha_g - \alpha_f < 0$
	B_{SC}	$(\beta_{s,f,g}, 0)$	$c_h = \alpha_h = 1, \alpha_g - \alpha_f = 0$
	G_{SC}	$(\infty_h, \gamma_{s,f,g,h})$	$c_h = \alpha_h = 1, \alpha_g - \alpha_f = 1$
	H_{SC}	$(\infty_h, \gamma_h = 1)$	$c_h = \alpha_h = 1, \alpha_g - \alpha_f > 1$
	J_{SC}	$(\infty_{f,g}, 0)$	$c_h = \alpha_h = 1, 0 < \alpha_g - \alpha_f < 1$

experiments to determine the functions. These issues limit the practical applicability and usefulness of the proposed model.

If the application involves data processing and analysis, the amount of data to be processed, and thus the parallelizable workload given by $p \cdot g(N)$, should be relatively easy to determine. Also, it seems reasonable to expect that the parallelizable workload can be almost equally distributed across the available PPUs ($h(N) \approx N$), unless the data processing requires taking into account data dependencies. In contrast, in numerical algorithms, which can be as simple as LU matrix decomposition as used in the second experiment, the determination of the functions g and h can become quite complicated. Also, in an optimization context, such as solving an instance of a mixed-integer linear program to optimality with a branch-and-bound algorithm, both the parallelizable workload $p \cdot g(N)$ and the effect of parallelization expressed by $h(N)$

may depend not only on N, which could be related to the size of the (optimization) problem instance to be solved, but also on the instance itself. For example, while some instances of a given problem may show (sub)linear speedup, other instances of the same size may benefit from superlinear speedup (see, for example, [33]).

6.2. Consideration of parallelization overhead

In the speedup and efficiency equations (5)-(6) and the resulting analysis, any overhead due to parallelization has been omitted for a variety of reasons. However, as can be seen from the results of the second computational experiment and widely acknowledged in the literature, parallelization overhead can cause large scalability degradation and have a significantly large impact on speedup and efficiency. Thus, the bounds may become loose.

Overhead can be caused by several phenomena, including the existence of critical regions (exclusive access for only one process), inter-process communication, the creation and management of threads, and sequential-to-parallel synchronization due to data exchange [41]. Such phenomena can be analyzed by considering an overhead function in the determination of parallel execution times, which are likely to depend, among other factors, on the number of parallel PUs.

In the literature, several ways of incorporating overhead functions into execution time evaluation have been proposed. One option is to include an additive overhead term in the speedup and efficiency functions (e.g., [15, 21, 29]); an alternative approach is to use a multiplicative term (coefficient function) to account for the increased workload of parallel execution due to parallelization overhead (e.g., [14, 36, 21]). Although I focus here on additive overhead functions, the key concepts, opportunities, and challenges for considering parallelization overhead also apply to multiplicative functions.

Our general speedup and efficiency equations (2)-(3) already account for additive overhead with the term $z(N)$. Assuming that parallelization overhead can also be determined by a polynomial function $z(N) := c_z \cdot N^{\alpha_z}$ ($c_z > 0, \alpha_z \geq 0$), my generic speedup and efficiency equations (5)-(6) would have to be changed to

$$S(N) = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)} + c_z \cdot N^{\alpha_z}} \tag{45}$$

and

$$E(N) = \frac{s \cdot c_f \cdot N^{\alpha_f} + p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{(\alpha_f+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g-\alpha_h+1)} + c_z \cdot N^{\alpha_z+1}}, \tag{46}$$

respectively.

In the study of Flatt and Kennedy [15], the authors suggest that an additive (and continuous) overhead function should satisfy some mathematical assumptions. Under these assumptions, some theoretical results can be derived. One important result is that when Amdahl's setting is extended to include overhead, the speedup has a unique maximum at $N \geq 1$. In Appendix D, I list the assumptions and prove that any polynomial overhead function $z(N) = c_z \cdot N^{\alpha_z} - c_z$ $(c_z, \alpha_z > 0, N \geq 1)$ satisfies all assumptions, so that the above result holds for modified equation (45) in the Amdahl setting, if we exclude the case $\alpha_z = 0$ and allow the expansion of $z(N)$ by subtracting the constant c_z. Flatt and Kennedy [15] also gives theoretical bounds on the (scaled) speedup with increasing problem size.

Huang et al. [21] also suggest using an additive overhead function under the Amdahl setting that accounts for the data transmission overhead in multi-core environments. They propose using an overhead function $z(i) = f_t + \frac{g_t}{i}$, where i is the number of communication links of a single core, and f_t and g_t are the sequential and parallel parts of the transmission, respectively. Applying their extended Amdahl model to a setting with area constraints [20], they look for zero points of the speedup derivative to identify the optimal speedup.

A key conclusion from these findings is that if an additive overhead function is considered in parallel execution time and performance analysis, then functions for execution time, speedup, and efficiency may become non-monotonous, and as a consequence, the determination of limits and bounds with asymptotic analysis needs to be replaced or complemented by an analysis of extreme points; i.e., an analysis that accounts for parallelization overhead should determine the optimal number of PPUs for a given metric such as parallel execution time, speedup, and efficiency. This approach would not only lead to better predictions, but would also immediately lead to a suggestion of the appropriate number of parallel PUs to choose. Using my general speedup and efficiency equations (2)-(3), this would lead to solving the optimization problems (47)-(49) for execution time, speedup, and efficiency, respectively.

$$\min_{N \in \mathbb{N}} \left(s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N) \right) \tag{47}$$

$$\max_{N\in\mathbb{N}} S(N) = \max_{N\in\mathbb{N}} \left(\frac{s \cdot f(1) + p \cdot g(1)}{s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N)} \right) \tag{48}$$

$$\max_{N\in\mathbb{N}} E(N) = \max_{N\in\mathbb{N}} \left(\frac{s \cdot f(1) + p \cdot g(1)}{N \cdot (s \cdot f(N) + \frac{p \cdot g(N)}{h(N)} + z(N))} \right) \tag{49}$$

The determination of an additive overhead function z should take into account the (architecture of the) parallel system, including the speed of the cores, the size and structure of the caches, and the operating system [8]. Also, it should consider the roots of the parallelization overhead. For example, Flatt and Kennedy [15] suggest using different overhead functions for i) scheduling in shared memory multiprocessors using critical regions, ii) synchronization in an array of processors arranged as a k-cube, and iii) synchronization in an array of processors connected by a logarithmic network. Additive terms were also used to account for the overhead associated with data preparation, communication, and synchronization [24, 3, 29, 41, 21].

In summary, a performance analysis that takes into account parallelization overhead needs to consider the root(s) of the parallelization overhead and the parallel system architecture applicable in a given context to determine the mathematical structure of the overhead function, its mathematical embedding in the computation of execution times (additive, multiplicative, etc.). From a methodological perspective, it requires searching for extreme points due to possible non-monotonicity of execution times, speedup and efficiency functions.

7. Conclusion

In this work, I provide a generic speedup (and thus also efficiency) model, which generalizes many prominent models suggested in the literature and allows showing that they can be considered special cases with different assumptions of a unifying approach. The genericity of the speedup model is achieved through parameterization. Considering combinations of parameter ranges, I identify six different asymptotic speedup cases and eight different asymptotic efficiency cases; these cases include sublinear, linear and superlinear speedup and efficiency. Based upon the identified speedup and efficiency cases, I derive eleven different scalability cases and types, to which instantiations of my generic speedup (and efficiency) model may lead.

Researchers can draw upon my suggested typology to classify their speedup model and/or to determine the asymptotic scalability of their application when the number of PPUs increases. Also, the description of two computational experiments demonstrates the practical application of the model and the typology.

My theoretical analysis is based upon several assumptions which are common in the literature (e.g., [37]). First, I assume that the overall workload only contains two parts, a sequential part and a perfectly parallelizable part, which can be executed in parallel on all available PPUs. In practice, the latter condition may not always hold but even then my speedup and efficiency results are useful as they can be used as upper bounds of achievable speedup and efficiency. Alternative models that do not require the above dichotomy distinction have been proposed in the literature, including parallelism/span-and-work models, multiple-fraction models, and roofline models (e.g., [12, p. 772ff], [3, p. 141ff]), [9]). Future theoretical analysis of speedup and efficiency limits may consider those types of models.

Second, while my generic speedup model includes a function for parallelization overhead, I assume that this overhead is negligible and omit this function from my analysis. However, I admit that parallelization overhead may cause large scalability degradation [21] and have considerably large effects on speedup and efficiency.

A discussion of the implications and research avenues for accounting for parallelization overhead is provided in Section 6.2. Future research can build on these ideas to obtain tighter bounds on speedup and efficiency, and to derive recommendations for the optimal choice of the number of PPUs to use.

Third, in my analysis I focus on homogeneous parallel computing environments. I acknowledge that, in modern parallel computing environments, parallel processing units are not necessarily equally potential in their computing capabilities and that a substantial body of literature on speedup in such heterogeneous computing environments exist; see, for example, the surveys on heterogeneous multicore environments of Al-Babtain et al. [2] and Al-hayanni et al. [3]. Many works suggest extensions of *Amdahl's law*, *Gustafson's law* and/or *Sun and Ni's Law* for such environments (e.g., [20, 22, 42, 31, 45, 26, 25]). Studies on speedup and efficiency properties of architecture-dependent laws are particularly helpful for the design of multi-core environments. Future work may extend my generic speedup model by concepts of different types of PPUs as suggested in the literature and adapt my theoretical analysis to heterogeneous settings.

Finally, merging the two abovementioned research streams leads to the consideration of speedup and efficiency in heterogeneous parallel computing environments under the consideration of overhead (functions). My model and theoretical analysis may be extended in both regards, drawing on prior work. For example, Huang et al. [21] suggest an extension of *Amdahl's law* and *Gustafson's law* in architecture-specific multi-core settings by considering communication overhead and area constraints; Pei et al. [29] extend *Amdahl's law* for heterogeneous multicore processors with the consideration of overhead through data preparation; and Morad et al. [27] analyze overhead as a result of synchronization, communication and coherence costs based upon Amdahl's model for asymmetric cluster chip multiprocessors.

With the suggestion of a generic and unifying speedup (and efficiency) model and its asymptotic analysis, I hope to provide a theoretical basis for and typology of scalability of parallel algorithms in homogeneous computing environments. Future research can draw upon and extend my research results to address various extensions of my setting.

Acknowledgment

I am especially grateful for the assistance of Philipp Speckenmeyer and Sascha Burmeister, who provided an excellent introduction to the use of the NOCTUA 2 computing cluster. They also helped me with many practical recommendations and some shell scripts. I would also like to thank Philip Speckenmeyer for his feedback on an early version of the manuscript.

References

[1] C. Ababei and M. G. Moghaddam. A survey of prediction and classification techniques in multicore processor systems. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1184–1200, 2018.

[2] B. M. Al-Babtain, F. J. Al-Kanderi, M. F. Al-Fahad, and I. Ahmad. A survey on amdahl's law extension in multicore architectures. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 3(3):30–46, 2013.

[3] M. A. N. Al-hayanni, F. Xia, A. Rafiev, A. Romanovsky, R. Shafik, and A. Yakovlev. Amdahl's law in the context of heterogeneous many-core systems–a survey. *IET Computers & Digital Techniques*, 14(4):133–148, 2020.

[4] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[5] B. Barney et al. Introduction to parallel computing. *Lawrence Livermore National Laboratory*, 6(13):10, 2010.

[6] R. S. Barr and B. L. Hickman. Reporting computational experiments with parallel algorithms: Issues, measures, and experts' opinions. *ORSA Journal on Computing*, 5(1):2–18, 1993.

[7] A. Borisenko, P. Kegel, and S. Gorlatch. Optimal design of multi-product batch plants using a parallel branch-and-bound method. In *International Conference on Parallel Computing Technologies*, pages 417–430. Springer, 2011.

[8] R. G. Brown. Amdahl's law & parallel speedup. USENIX website: https://www. usenix.org., Aug. 2000.

[9] A. S. Cassidy and A. G. Andreou. Beyond amdahl's law: An objective function that links multiprocessor performance gains to delay and energy. *IEEE Transactions on Computers*, 61(8):1110–1126, 2011.

[10] H. Che and M. Nguyen. Amdahl's law for multithreaded multicore processors. *Journal of Parallel and Distributed Computing*, 74(10): 3056–3069, 2014. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2014.06.012. URL https://www.sciencedirect.com/science/article/pii/S0743731514001142.

[11] J. Chen, C. Du, F. Xie, and B. Lin. Scheduling non-preemptive tasks with strict periods in multi-core real-time systems. *Journal of Systems Architecture*, 90:72–84, 2018.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[13] A. Dabah, I. Chegrane, S. Yahiaoui, A. Bendjoudi, and N. Nouali-Taboudjemat. Efficient parallel branch-and-bound approaches for exact graph edit distance problem. *Parallel Computing*, 114:102984, 2022.

[14] S. Eyerman and L. Eeckhout. Modeling critical sections in amdahl's law and its implications for multicore design. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 362–370, 2010.

[15] H. P. Flatt and K. Kennedy. Performance of parallel processors. *Parallel Computing*, 12(1):1–20, 1989.

[16] S. H. Fuller and L. I. Millett. *The future of computing performance: game over or next level?* National Academy Press, 2011.

[17] T. Gonggiatgul, G. Shobaki, and P. Muyan-Özçelik. A parallel branch-and-bound algorithm with history-based domination and its application to the sequential ordering problem. *Journal of Parallel and Distributed Computing*, 172:131–143, 2023. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2022.10.007. URL https://www.sciencedirect.com/science/article/pii/S0743731522002167.

[18] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to parallel computing*. Harlow, England : Pearson, Addison-Wesley, second edition 2. ed. edition, 2003. ISBN 978-0-201-64865-2, 0-201-64865-2.

[19] J. L. Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[20] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[21] T. Huang, Y. Zhu, M. Qiu, X. Yin, and X. Wang. Extending amdahl's law and gustafson's law by evaluating interconnections on multi-core processors. *The Journal of Supercomputing*, 66(1):305–319, 2013.

[22] B. H. Juurlink and C. H. Meenderinck. Amdahl's law for predicting the future of multicores considered harmful. *ACM SIGARCH Computer Architecture News*, 40(2):1–9, 2012.

[23] Jülich Supercomputing Centre. Research fields and geographic distribution of supercomputer users. https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/user-research-fields-distribution, 2022.

[24] X. Li and M. Malek. Analysis of speedup and communication/computation ratio in multiprocessor systems. In *Proceedings. Real-Time Systems Symposium*, pages 282–283. IEEE Computer Society, 1988.

[25] D. Moncrieff, R. E. Overill, and S. Wilson. Heterogeneous computing machines and amdahl's law. *Parallel Computing*, 22(3):407–413, 1996.

[26] A. Morad, T. Y. Morad, Y. Leonid, R. Ginosar, and U. Weiser. Generalized multiamdahl: Optimization of heterogeneous multi-accelerator soc. *IEEE Computer Architecture Letters*, 13(1):37–40, 2012.

[27] T. Y. Morad, U. C. Weiser, A. Kolodnyt, M. Valero, and E. Ayguade. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *IEEE Computer Architecture Letters*, 5(1):14–17, 2006.

[28] National Center for Supercomputing Applications. Pushing boundaries. farther. https://www.ncsa.illinois.edu/research/, 2021.

[29] S. Pei, M.-S. Kim, and J.-L. Gaudiot. Extending amdahl's law for heterogeneous multicore processor with consideration of the overhead of data preparation. *IEEE Embedded Systems Letters*, 8(1):26–29, 2016.

[30] J. L. Ponz-Tienda, A. Salcedo-Bernal, and E. Pellicer. A parallel branch and bound algorithm for the resource leveling problem with minimal lags. *Computer-Aided Civil and Infrastructure Engineering*, 32(6):474–498, 2017.

[31] A. Rafiev, M. A. Al-Hayanni, F. Xia, R. Shafik, A. Romanovsky, and A. Yakovlev. Speedup and power scaling models for heterogeneous many-core systems. *IEEE Transactions on Multi-scale computing systems*, 4(3):436–449, 2018.

[32] J. K. Rai, A. Negi, R. Wankar, and K. Nayak. Performance prediction on multi-core processors. In *2010 International Conference on Computational Intelligence and Communication Networks*, pages 633–637. IEEE, 2010.

[33] G. Rauchecker and G. Schryen. Using high performance computing for unrelated parallel machine scheduling with sequence-dependent setup times: Development and computational evaluation of a parallel branch-and-price algorithm. *Computers & Operations Research*, 104:338–357, 2019.

[34] B. Schmidt, J. Gonzalez-Dominguez, C. Hundt, and M. Schlarb. *Parallel programming: concepts and practice*. Morgan Kaufmann, 2017.

[35] X.-H. Sun and Y. Chen. Reevaluating amdahl's law in the multicore era. *Journal of Parallel and distributed Computing*, 70(2):183–188, 2010.

[36] X.-H. Sun and L. M. Ni. Another view on parallel speedup. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 324–333, 1990.

[37] X.-H. Sun and L. M. Ni. Scalable problems and memory-bounded speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993.

[38] TOP500.org. Top500. https://www.top500.org/lists/top500/, 2022.

[39] R. Trobec, B. Slivnik, P. Bulic, and B. Robic. *Introduction to Parallel Computing: From Algorithms to Programming on State-of-Art Platforms*. Springer, 2018.

[40] F. Xia, A. Rafiev, A. Aalsaud, M. Al-Hayanni, J. Davis, J. Levine, A. Mokhov, A. Romanovsky, R. Shafik, A. Yakovlev, et al. Voltage, throughput, power, reliability, and multicore scaling. *Computer*, 50(8):34–45, 2017.

[41] L. Yavits, A. Morad, and R. Ginosar. The effect of communication and synchronization on amdahl's law in multicore systems. *Parallel Computing*, 40(1):1–16, 2014.

[42] N. Ye, Z. Hao, and X. Xie. The speedup model for manycore processor. In *2013 International Conference on Information Science and Cloud Computing Companion*, pages 469–474, 2013. doi: 10.1109/ISCC-C.2013.146.

[43] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Computing Surveys (CSUR)*, 45(1):1–28, 2012.

[44] T. Zidenberg, I. Keslassy, and U. Weiser. Multiamdahl: How should i divide my heterogenous chip? *IEEE Computer Architecture Letters*, 11 (2):65–68, 2012.

[45] T. Zidenberg, I. Keslassy, and U. Weiser. Optimal resource allocation with multiamdahl. *Computer*, 46(07):70–77, 2013.

A. Calculations of speedup limits

We rewrite (I) as follows:

$$(I) = \frac{s \cdot c_f \cdot N^{\alpha_f}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}} = \frac{s \cdot c_f}{s \cdot c_f + \frac{p \cdot c_g}{c_h} \cdot N^{\alpha_g - \alpha_h - \alpha_f}} \tag{50}$$

and obtain

$$\lim_{N \to \infty}(I) = \begin{cases} 1,\ \alpha_f > \alpha_g - \alpha_h & (51) \\ \dfrac{s \cdot c_f}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}, \alpha_f = \alpha_g - \alpha_h & (52) \\ 0, \alpha_f < \alpha_g - \alpha_h & (53) \end{cases}$$

It should be noticed that, in contrast to the limits in equations (51) and (52), which show upper bounds, the limit in equation (53) represents a lower bound.

We rewrite (II) as follows:

$$(II) = \frac{p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{\alpha_f} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g - \alpha_h)}} = \begin{cases} \dfrac{p \cdot c_g}{s \cdot c_f \cdot N^{(\alpha_f - \alpha_g)} + \frac{p \cdot c_g}{c_h \cdot N^{\alpha_h}}},\ \alpha_f > \alpha_g & (54) \\ \dfrac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h \cdot N^{\alpha_h}}},\ \alpha_f = \alpha_g & (55) \\ \dfrac{p \cdot c_g \cdot N^{(\alpha_g - \alpha_f)}}{s \cdot c_f + \frac{p \cdot c_g}{c_h} \cdot c_h \cdot N^{(\alpha_g - \alpha_h - \alpha_f)}},\ \alpha_f < \alpha_g & (56) \end{cases}$$

For equation (54), we obtain

$$\lim_{N \to \infty}(II) = 0,\ \alpha_f > \alpha_g \tag{57}$$

For equation (55), we obtain

$$\lim_{N\to\infty}(II) = \begin{cases} \dfrac{p\cdot c_g}{s\cdot c_f},\ \alpha_f = \alpha_g, \alpha_h > 0 & (58) \\[2ex] \dfrac{p\cdot c_g}{s\cdot c_f + \frac{p\cdot c_g}{c_h}},\ \alpha_f = \alpha_g, \alpha_h = 0 & (59) \end{cases}$$

For equation (56), we obtain

$$\lim_{N\to\infty}(II) = \begin{cases} \lim\limits_{N\to\infty} \dfrac{p\cdot c_g \cdot N^{\alpha_h}}{\frac{s\cdot c_f}{N^{(\alpha_g-\alpha_h-\alpha_f)}} + \frac{p\cdot c_g}{c_h}} = \infty\ (\Theta(N^{\alpha_h})),\ \alpha_f < \alpha_g, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f \geq 0 & (60) \\[2ex] \lim\limits_{N\to\infty} \dfrac{p\cdot c_g \cdot N^{(\alpha_g-\alpha_f)}}{s\cdot c_f + \frac{p\cdot c_g}{c_h}\cdot N^{(\alpha_g-\alpha_h-\alpha_f)}} = \infty\ (\Theta(N^{(\alpha_g-\alpha_f)})),\ \alpha_f < \alpha_g, \alpha_h > 0, \alpha_g - \alpha_h - \alpha_f < 0 & (61) \\[2ex] \lim\limits_{N\to\infty} \dfrac{p\cdot c_g}{\frac{s\cdot c_f}{N^{(\alpha_g-\alpha_f)}} + \frac{p\cdot c_g}{c_h}} = c_h,\ \alpha_f < \alpha_g, \alpha_h = 0 & (62) \end{cases}$$

It should be noticed that, in contrast to the limits in equations (58), (59) and (62), which show upper bounds, the limit in equation (57) represents a lower bound.

B. Calculations of efficiency limits

$$\lim_{N\to\infty}(I') = \lim_{N\to\infty} \frac{s\cdot c_f}{s\cdot c_f\cdot N + \frac{p\cdot c_g}{c_h}\cdot N^{(\alpha_g-\alpha_h-\alpha_f+1)}} = \begin{cases} 0,\ \alpha_g - \alpha_h - \alpha_f + 1 > 0 & (63) \\[1ex] 0,\ \alpha_g - \alpha_h - \alpha_f + 1 = 0 & (64) \\[1ex] 0,\ \alpha_g - \alpha_h - \alpha_f + 1 < 0 & (65) \end{cases}$$

It should be noticed that the limits in equations (63) to (65) all show lower bounds.

We rewrite (II') as follows:

$$(II') = \frac{p \cdot c_g \cdot N^{\alpha_g}}{s \cdot c_f \cdot N^{(\alpha_f+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g-\alpha_h+1)}} = \begin{cases} \dfrac{p \cdot c_g \cdot N^{(\alpha_g-\alpha_f-1)}}{s \cdot c_f + \frac{p \cdot c_g}{c_h} \cdot N^{(\alpha_g-\alpha_h-\alpha_f)}}, & \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f \geq \alpha_g - \alpha_h \quad (66) \\ \dfrac{p \cdot c_g \cdot N^{(\alpha_h-1)}}{\frac{s \cdot c_f}{N^{(\alpha_g-\alpha_f-1)}} + \frac{p \cdot c_g}{c_h}}, & \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f < \alpha_g - \alpha_h \quad (67) \\ \dfrac{p \cdot c_g}{s \cdot c_f \cdot N^{(\alpha_f-\alpha_g+1)} + \frac{p \cdot c_g}{c_h} \cdot N^{(-\alpha_h+1)}}, & \text{else} \quad (68) \end{cases}$$

and obtain

$$\lim_{N\to\infty}(II') = \begin{cases} 0, \ \alpha_f > \alpha_g - 1, 0 \leq \alpha_h \leq 1 & (69) \\ 0, \ \alpha_f > \alpha_g - 1, \alpha_h > 1 & (70) \\ 0, \ \alpha_f = \alpha_g - 1, 0 \leq \alpha_h \leq 1 & (71) \\ \dfrac{p \cdot c_g}{s \cdot c_f + \frac{p \cdot c_g}{c_h}}, \ \alpha_f = \alpha_g - 1, \alpha_h = 1 & (72) \\ \dfrac{p \cdot c_g}{s \cdot c_f}, \ \alpha_f = \alpha_g - 1, \alpha_h > 1 & (73) \\ 0, \ \alpha_f < \alpha_g - 1, 0 \leq \alpha_h < 1 & (74) \\ c_h, \ \alpha_f < \alpha_g - 1, \alpha_h = 1 & (75) \\ \infty \ (\Theta(N^{\alpha_g-\alpha_f-1})), \ \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f > \alpha_g - \alpha_h & (76) \\ \infty \ (\Theta(N^{\alpha_h-1})), \ \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f = \alpha_g - \alpha_h & (77) \\ \infty \ (\Theta(N^{\alpha_h-1})), \ \alpha_f < \alpha_g - 1, \alpha_h > 1, \alpha_f < \alpha_g - \alpha_h & (78) \end{cases}$$

It should be noticed that the limits in equations (69) to (74) all show lower bounds while the limit in equation (75) is an upper bound.

C. Computational experiments

C.1. Experiment 1: Parallel matrix multiplication with fixed workload

Table 4
Computational results of matrix multiplication with fixed size of all matrices

N	T(N) [in ms]	S(N)		E(N)	
		Theor.	Comp.	Theor.	Comp.
1	1,529,020	–	–	–	–
2	953,760	1.953898	1.603150	0.976949	0.801575
4	493,262	3.735577	3.099813	0.933894	0.774953
8	270,447	6.865980	5.653677	0.858248	0.706710
16	163,341	11.817493	9.360908	0.738593	0.585057
32	100,269	18.481672	15.249180	0.577552	0.476537
64	74,392	25.739145	20.553555	0.402174	0.321149
128	64,154	32.027504	23.833588	0.250215	0.186200

C.2. Experiment 2: Parallel LU decomposition with variable workload

Using the algorithm shown in Figure 6, I determine the number of calculations along the outer loop that iterates over the variable i. In the first iteration ($i = 1$), the elements $L_{j,1}$ ($j = 2 \ldots z$) and the elements $U_{j,l}$ ($j, l = 2 \ldots z$) are calculated and assigned, for a total of $(z-1) \cdot z$ calculations. Similarly, in the second iteration ($i = 2$), the elements $L_{j,2}$ ($j = 3 \ldots z$) and the elements $U_{j,l}$ ($j, l = 3 \ldots z$) are calculated and assigned; i.e., a total of $(z-2) \cdot (z-1)$ calculations are required. In general, in iteration i, the number of required calculations equals $(z-i) \cdot (z-i+1)$. This yields

$$
\begin{aligned}
\hat{g}(z) &= \sum_{i=1}^{z-1} [(z-i) \cdot (z-i+1)] = \sum_{i=1}^{z-1} \left[z^2 - z \cdot i + z - z \cdot i + i^2 - i\right] \\
&= \sum_{i=1}^{z-1} \left[z^2 + z + i \cdot (-z - z - 1) + i^2\right] = \sum_{i=1}^{z-1} \left[z^2 + z\right] + \sum_{i=1}^{z-1} [(-2z-1) \cdot i] + \sum_{i=1}^{z-1} i^2 \\
&= (z-1) \cdot (z^2 + z) - (2z+1) \cdot \sum_{i=1}^{z-1} i + \sum_{i=1}^{z-1} i^2 \\
&= z^3 + z^2 - z^2 - z - (2z+1) \cdot \frac{z \cdot (z-1)}{2} + \frac{z \cdot (z-1) \cdot (2z-1)}{6} \\
&= z^3 - z + \frac{z \cdot (z-1) \cdot [(2z-1) \cdot 3 \cdot (2z+1)]}{6} \\
&= z^3 - z + \frac{(z^2 - z) \cdot (-4z-4)}{6} = z^3 - z + \frac{(z^2 - z) \cdot (-2z-2)}{3} \\
&= z^3 - z + \frac{(-2z^3 + 2z)}{3} = z^3 - z - \frac{2}{3} z^3 + \frac{2}{3} z = \frac{1}{3} z^3 + \frac{1}{3} z = \frac{1}{3}\left(z^3 - z\right)
\end{aligned}
$$

Table 5
Data points of LR decomposition (z_1 =100)

i	$N = 2^i$	z	$\hat{g}(N, z_1)$	$\hat{g}_{reduced}(N, z_1)$	$\hat{h}(N, z_1)$	$\hat{h}(N, z_1)/N$
1	2	200	$2.6666e + 06$	$1.33835e + 06$	1.99245	0.996227
2	4	400	$2.13332e + 07$	$5.3634e + 06$	3.97755	0.994388
3	8	800	$1.70666e + 08$	$2.14733e + 07$	7.94784	0.99348
4	16	$1,600$	$1.36533e + 09$	$8.59323e + 07$	15.8885	0.993029
5	32	$3,200$	$1.09227e + 10$	$3.43807e + 08$	31.7697	0.992805
6	64	$6,400$	$8.73813e + 10$	$1.37538e + 09$	63.5323	0.992693
7	128	$12,800$	$6.99051e + 11$	$5.50185e + 09$	127.057	0.992636
8	256	$25,600$	$5.59241e + 12$	$2.2008e + 10$	254.108	0.992608
9	512	$51,200$	$4.47392e + 13$	$8.80333e + 10$	508.208	0.992594
10	1024	$102,400$	$3.57914e + 14$	$3.52136e + 11$	1016.41	0.992587
11	2048	$204,800$	$2.86331e + 15$	$1.40855e + 12$	2032.81	0.992584
12	4096	$409,600$	$2.29065e + 16$	$5.6342e + 12$	4065.62	0.992582
13	8192	$819,200$	$1.83252e + 17$	$2.25368e + 13$	8131.23	0.992581
14	16384	$1.6384e + 06$	$1.46602e + 18$	$9.01473e + 13$	16262.4	0.992581
15	32768	$3.2768e + 06$	$1.17281e + 19$	$3.60589e + 14$	32524.9	0.992581
16	65536	$6.5536e + 06$	$9.3825e + 19$	$1.44236e + 15$	65049.8	0.992581
17	131072	$1.31072e + 07$	$7.506e + 20$	$5.76943e + 15$	130100	0.992581
18	262144	$2.62144e + 07$	$6.0048e + 21$	$2.30777e + 16$	260199	0.99258
19	524288	$5.24288e + 07$	$4.80384e + 22$	$9.23109e + 16$	520398	0.99258
20	$1.04858e + 06$	$1.04858e + 08$	$3.84307e + 23$	$3.69243e + 17$	$1.0408e + 06$	0.99258

In the context of determining the scaling function h, the Tables 5-8 show data points of N, $\hat{g}(N, z_1)$, $\hat{g}_{reduced}(N, z_1)$, $\hat{h}(N, z_1)$, and $(\hat{h}(N, z_1)/N)$ for various values of the number of rows/columns (z_1) of the input matrix.

Table 6
Data points of LR decomposition (z_1 =1,000)

i	$N = 2^i$	z	$\hat{g}(N, z_1)$	$\hat{g}_{reduced}(N, z_1)$	$\hat{h}(N, z_1)$	$\hat{h}(N, z_1)/N$
1	2	2,000	$2.66667e+09$	$1.33383e+09$	1.99925	0.999625
2	4	4,000	$2.13333e+10$	$5.33633e+09$	3.99775	0.999438
3	8	8,000	$1.70667e+11$	$2.13473e+10$	7.99475	0.999344
4	16	16,000	$1.36533e+12$	$8.53933e+10$	15.9888	0.999297
5	32	32,000	$1.09227e+13$	$3.41581e+11$	31.9768	0.999274
6	64	64,000	$8.73813e+13$	$1.36634e+12$	63.9528	0.999262
7	128	128,000	$6.99051e+14$	$5.4654e+12$	127.905	0.999257
8	256	256,000	$5.59241e+15$	$2.18616e+13$	255.809	0.999254
9	512	512,000	$4.47392e+16$	$8.74467e+13$	511.617	0.999252
10	1024	$1.024e+06$	$3.57914e+17$	$3.49787e+14$	1023.23	0.999252
11	2048	$2.048e+06$	$2.86331e+18$	$1.39915e+15$	2046.47	0.999251
12	4096	$4.096e+06$	$2.29065e+19$	$5.5966e+15$	4092.93	0.999251
13	8192	$8.192e+06$	$1.83252e+20$	$2.23864e+16$	8185.86	0.999251
14	16384	$1.6384e+07$	$1.46602e+21$	$8.95456e+16$	16371.7	0.999251
15	32768	$3.2768e+07$	$1.17281e+22$	$3.58182e+17$	32743.5	0.999251
16	65536	$6.5536e+07$	$9.3825e+22$	$1.43273e+18$	65486.9	0.999251
17	131072	$1.31072e+08$	$7.506e+23$	$5.73092e+18$	130974	0.999251
18	262144	$2.62144e+08$	$6.0048e+24$	$2.29237e+19$	261948	0.999251
19	524288	$5.24288e+08$	$4.80384e+25$	$9.16947e+19$	523895	0.999251
20	$1.04858e+06$	$1.04858e+09$	$3.84307e+26$	$3.66779e+20$	$1.04779e+06$	0.999251

Table 7
Data points of LR decomposition (z_1 =10,000)

i	$N = 2^i$	z	$\hat{g}(N, z_1)$	$\hat{g}_{reduced}(N, z_1)$	$\hat{h}(N, z_1)$	$\hat{h}(N, z_1)/N$
1	2	20,000	$2.66667e+12$	$1.33338e+12$	1.99992	0.999962
2	4	40,000	$2.13333e+13$	$5.33363e+12$	3.99978	0.999944
3	8	80,000	$1.70667e+14$	$2.13347e+13$	7.99948	0.999934
4	16	160,000	$1.36533e+15$	$8.53393e+13$	15.9989	0.99993
5	32	320,000	$1.09227e+16$	$3.41358e+14$	31.9977	0.999927
6	64	640,000	$8.73813e+16$	$1.36543e+15$	63.9953	0.999926
7	128	$1.28e+06$	$6.99051e+17$	$5.46174e+15$	127.99	0.999926
8	256	$2.56e+06$	$5.59241e+18$	$2.1847e+16$	255.981	0.999925
9	512	$5.12e+06$	$4.47392e+19$	$8.73879e+16$	511.962	0.999925
10	1024	$1.024e+07$	$3.57914e+20$	$3.49552e+17$	1023.92	0.999925
11	2048	$2.048e+07$	$2.86331e+21$	$1.39821e+18$	2047.85	0.999925
12	4096	$4.096e+07$	$2.29065e+22$	$5.59282e+18$	4095.69	0.999925
13	8192	$8.192e+07$	$1.83252e+23$	$2.23713e+19$	8191.39	0.999925
14	16384	$1.6384e+08$	$1.46602e+24$	$8.94852e+19$	16382.8	0.999925
15	32768	$3.2768e+08$	$1.17281e+25$	$3.57941e+20$	32765.5	0.999925
16	65536	$6.5536e+08$	$9.3825e+25$	$1.43176e+21$	65531.1	0.999925
17	131072	$1.31072e+09$	$7.506e+26$	$5.72705e+21$	131062	0.999925
18	262144	$2.62144e+09$	$6.0048e+27$	$2.29082e+22$	262124	0.999925
19	524288	$5.24288e+09$	$4.80384e+28$	$9.16328e+22$	524249	0.999925
20	$1.04858e+06$	$1.04858e+10$	$3.84307e+29$	$3.66531e+23$	$1.0485e+06$	0.999925

Table 8
Data points of LR decomposition (z_1 =100,000)

i	$N = 2^i$	z	$\hat{g}(N, z_1)$	$\hat{g}_{reduced}(N, z_1)$	$\hat{h}(N, z_1)$	$\hat{h}(N, z_1)/N$
1	2	$200,000$	$2.66667e+15$	$1.33334e+15$	1.99999	0.999996
2	4	$400,000$	$2.13333e+16$	$5.33336e+15$	3.99998	0.999994
3	8	$800,000$	$1.70667e+17$	$2.13335e+16$	7.99995	0.999993
4	16	$1.6e+06$	$1.36533e+18$	$8.53339e+16$	15.9999	0.999993
5	32	$3.2e+06$	$1.09227e+19$	$3.41336e+17$	31.9998	0.999993
6	64	$6.4e+06$	$8.73813e+19$	$1.36534e+18$	63.9995	0.999993
7	128	$1.28e+07$	$6.99051e+20$	$5.46137e+18$	127.999	0.999993
8	256	$2.56e+07$	$5.59241e+21$	$2.18455e+19$	255.998	0.999993
9	512	$5.12e+07$	$4.47392e+22$	$8.7382e+19$	511.996	0.999993
10	1024	$1.024e+08$	$3.57914e+23$	$3.49528e+20$	1023.99	0.999993
11	2048	$2.048e+08$	$2.86331e+24$	$1.39811e+21$	2047.98	0.999993
12	4096	$4.096e+08$	$2.29065e+25$	$5.59245e+21$	4095.97	0.999993
13	8192	$8.192e+08$	$1.83252e+26$	$2.23698e+22$	8191.94	0.999993
14	16384	$1.6384e+09$	$1.46602e+27$	$8.94792e+22$	16383.9	0.999993
15	32768	$3.2768e+09$	$1.17281e+28$	$3.57917e+23$	32767.8	0.999993
16	65536	$6.5536e+09$	$9.3825e+28$	$1.43167e+24$	65535.5	0.999993
17	131072	$1.31072e+10$	$7.506e+29$	$5.72667e+24$	131071	0.999993
18	262144	$2.62144e+10$	$6.0048e+30$	$2.29067e+25$	262142	0.999992
19	524288	$5.24288e+10$	$4.80384e+31$	$9.16267e+25$	524284	0.999993
20	$1.04858e+06$	$1.04858e+11$	$3.84307e+32$	$3.66507e+26$	$1.04857e+06$	0.999993

Table 9
Computational results of LU matrix decomposition with variable size of matrices

N	#(P)PUs	T(N) [in ms]	S(N)		E(N)	
			Theor.	Comp.	Theor.	Comp.
1	1	2	–	–	–	–
2	1	21	1.997481	2.100000	0.998741	1.050000
	2	10				
4	1	167	3.998107	4.771429	0.999527	1.192857
	4	35				
8	1	1,053	7.998896	9.486486	0.999862	1.185811
	8	111				
16	1	10,255	15.999408	17.293423	0.999963	1.080839
	16	593				
32	1	94,539	31.999695	31.628973	0.999990	0.988405
	32	2,989				
64	1	831,699	63.999844	46.016323	0.999998	0.719005
	64	18,074				
128	1	5,383,229	127.999924	26.609865	0.999999	0.207890
	128	202,302				

#(P)PUs: number of (parallel) processing units.

D. Mathematical requirements on overhead functions [15]

Flatt and Kennedy [15] formulate the following five requirements on a parallelization overhead function $z : \mathcal{R} \to \mathcal{R}$:

1. z is continuous and twice differentiable with respect to N.
2. $z(1) = 0$
3. $z'(N) > 0 \quad \forall N \geq 1$
4. $N \cdot z''(N) + 2 \cdot z'(N) > 0 \quad \forall N \geq 1$
5. There exists $N_1 \geq 1$ such that $z(N_1) = 1$.

Lemma D.1. *Each function $z(N) := c_z \cdot N^{\alpha_z} - c_z$ with $c_z, \alpha_z > 0$ meets the above requirements for $N \geq 1$.*

Proof. I prove any of the five conditions separately:

1. z is apparently continuous and twice differentiable with $z'(N) = c_z \cdot \alpha_z \cdot N^{\alpha_z - 1}$ and $z''(N) = c_z \cdot \alpha_z \cdot (\alpha_z - 1) \cdot N^{\alpha_z - 2} \quad \forall \alpha_z, c_z > 0$
2. $z(1) = c_z \cdot 1^{\alpha_z} - c_z = 0 \quad \forall \alpha_z, c_z > 0$
3. $z'(N) = c_z \cdot \alpha_z \cdot N^{\alpha_z - 1} > 0 \qquad \forall \alpha_z, c_z > 0 \quad \forall N \geq 1$
4. $N \cdot z''(N) + 2 \cdot z'(N) = N \cdot c_z \cdot \alpha_z \cdot (\alpha_z - 1) \cdot N^{\alpha_z - 2} + 2 \cdot c_z \cdot \alpha_z \cdot N^{\alpha_z - 1} = c_z \cdot \alpha_z \cdot (\alpha_z + 1) \cdot N^{\alpha_z - 1} > 0 \quad \forall \alpha_z, c_z > 0 \quad \forall N \geq 1$
5. Setting $z(N_1) = 1$ leads to $c_z \cdot N_1^{\alpha_z} - c_z = 1 \Leftrightarrow N_1^{\alpha_z} = \frac{c_z + 1}{c_z} \Leftrightarrow N_1 = \left(\frac{c_z + 1}{c_z}\right)^{\frac{1}{\alpha_z}} > 1 \qquad \forall c_z, \alpha_z > 0$

□