# TEMPUS: Activity 2

# Parallel Algorithms,
# Chapter 1 Fundamentals,
# *Section 4 Performance metrics and Scalability*

**István Erényi, KFKI-MSZKI**

**20th July 1997**

## Overview

# Performance metrics
# 1 Introduction

## Performance metrics

Sequential algorithms are evaluated in terms of execution time as a function of input size.

Execution of parallel algorithms depends on:

input size, *n*

architecture

number of processors, *p*.

A **parallel system** is a combination of the *algorithm*, and the *parallel architecture* on which it runs. Hence, we need metrics for the performance of parallel systems, that consider the underlying architecture.

The **scalability** of a parallel algorithm on an architecture is a measure of its ability to achieve performance proportional to the number of processors.

# Performance metrics
# 2 Definitions:

**Speedup** *($S_p$)***:** measure that expresses the relative benefit of the parallel solution. It is the ratio of the time to solve the problem on a single processor *($T_1$)* to the time to solve it on *p* processors *($T_p$)*.

$$S_p = T_1/T_p$$

*Superlinear speedup ($S_p$>p)* theoretically impossible, only if not identical conditions are considered (e.g. larger memory),

**Efficiency** ***($E_p$)*****:** measure for the fraction of time, processors are usefully employed:

$$E_p = S_p/p.$$

**Cost** ***($C_p$)*** or **processor-time product:**

$C_p = p\ T_p$

*Cost optimality* achieved if $E_p = 1$.

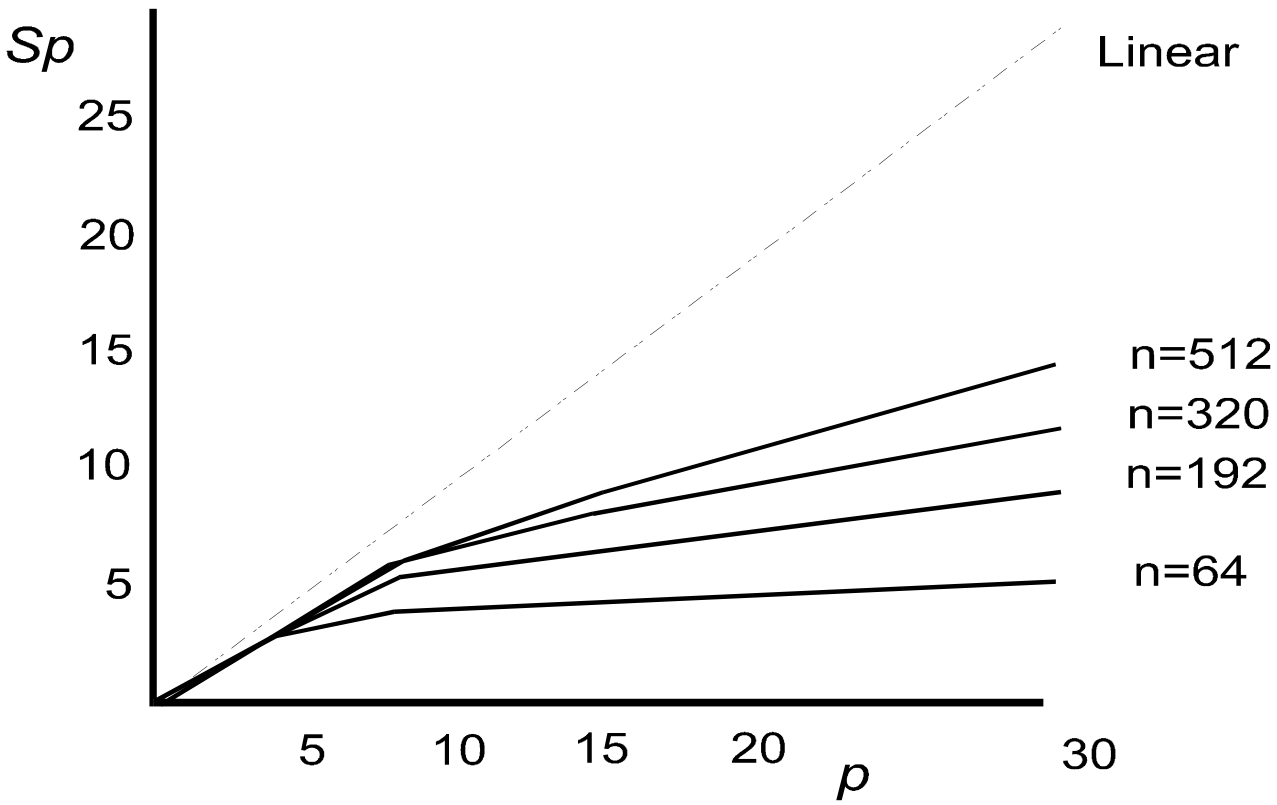
# Performance metrics
## 3 The effect of granularity on the performance

Finest granularity when each input item is processed by a separate processor. Using fewer processor **- scaling down -** the parallel system increases granularity. The performance of a scaled down system may much differ from the original one.

Algorithm with $W$ basic computation steps can be mapped at most onto $p=W$ processors, a single step for each. If fewer processors are used, then each executes a bigger part.

There is no trivial way for optimal scaling-down strategy. Moreover, the architectural features, like communication should be taken into account.

# Example to represent the speedup dependence on the granularity of the parallel computing system




Speedup (Sp) versus the number of processors (p) for adding n numbers on a hypercube

# Scalability
## 4 Scalability of parallel systems

With more processors, speedup is usually less than the number of processors *(p)*. Moreover, it does not increase linearly.

**Amdahl's law:** a problem of size $W$ has a serial component $W_S$ then the upper bound to speedup is $W/W_S$, no matter how many processors we have. This we get via solving the equation when $p \rightarrow \infty$.

$$S_p = W\,T_1 / (W_S\,T_1 + (W - W_S) T_p)$$

The ability to maintain the efficiency close to *1*, when speedup increases linearly is the problem of scalability. **Scalable** systems have linear or (close to linear) speedup. **Scalability** reflects the ability to utilize increasing processor resources effectively.

# Performance and Scalability
## 5 Problem Size and Overhead function

**Problem size** *(W)* is <u>*different*</u> from the input size. The problem size refers to *computation work*, the <u>*total number of basic operations*</u>. It is a function of input size *(n)*.

**Overhead function *($T_O$):*** all causes of nonoptimal efficiency (communication, data rearrangement, synchronization, etc.) is the overhead) of parallel systems. It is related to the total time spent by all processors in addition to the operation the (fastest known) serial algorithm requires.
It is a function of both *W* and *p*: $T_O(W,p)$.

**Cost** of problem-solving by a parallel system is the product of the processor number *(p)* and the execution time ($T_P$ ), i.e.: $p\ T_P$,

Hence, the overhead:

$$T_O = p\ T_P - W$$

# Performance and Scalability
## 6 Isoefficiency function

In various parallel systems $W$ and $p$ must be increased at different rates in order to maintain a fixed (constant) efficiency.

Speedup and efficiency may be expressed in terms of the overhead function:

$$T_p = (W + T_O(W,p))1/p$$

$$S_p = W/T_p = p\,W / (W + T_O(W,p))$$

$$E = S_p/p = W/(W + T_O(W,p))$$

If the ratio $T_O(W,p)/W$ is maintained constant, efficiency can be kept fixed. Reordering:

$W = T_O(W,p)\,(E/1-E)$; or denoting: $E/1-E = K$

$$W = K\,T_O(W,p)$$

This is the **isoefficiency function** defining the easy with which constant efficiency may be kept. Highly scalable systems have small isoefficiency value, small increase in *(W)* is enough to keep the system efficient when we add processors.

# Performance and Scalability
# 7 Cost optimality

Let denote the time of the fastest sequential algorithm on a single processor: $W$.

Parallel systems are **cost-optimal** if $pT_p$ is proportional to $W$, i.e.

$$pT_p = \Theta(W).$$

Substituting $pT_p$, we get:

$$W + T_o(W,p) = \Theta(W)$$

$$T_o(W,p) = O(W)$$

These equations show, that a parallel system is cost-optimal if its overhead function does not asymptotically exceed the problem size.

# Performance and Scalability
# 8 Degree of Concurrency

The maximum number of tasks that can be executed concurrently is called **degree of concurrency**

Degree of concurrency is a measure of the numbers of operations that an algorithm can perform in parallel for a problem of size *W*. <u>It is independent of the parallel architecture.</u>

Let denote *C(W)* as the degree of concurrency of a parallel algorithm, then for a problem size *W* no more than *C(W)* processors can be used in an efficient way.

# Performance and Scalability
# 9 Sources of Parallel overhead

## Sources of Parallel overhead

Interprocessor communication
Load imbalance
Extra computation.

## Interprocessor communication

The time to transfer data among processors is usually the main source for overhead. If each processor spends $t_{comm}$ time for communication then interprocessor communication contributes $pt_{comm}$ to the overhead function.

# Performance and Scalability
# 9 Sources of Parallel overhead
**(continued)**

**Load imbalance**

It is often difficult to predict the size of subtasks. Hence, it is impossible to divide the problem among equal sub-tasks for the processors. Thus, some processors become idle, while others still work.

At certain points in the algorithm the processors must synchronize. If one is ready for synchronization, and the other is still working, then the first has to wait, i.e. it becomes idle.

If a sequential part is met in the algorithm, then only one of the processors work, others become idle. The size of this part is denoted $W_S$.

All these factors contribute to the overhead.

# Performance and Scalability
## 9 Sources of Parallel overhead
**(continued)**

## Extra computation

Often the fastest sequential algorithm (with problem size *W*) may be difficult to parallelize, but a poorer one (problem size *W'*) proves to be easily parallelizable (it has a higher degree of concurrency). In this case *W-W'* is a source of overhead.
If the parallel algorithm provides still more aggregate computation than the serial one, its usage is beneficial.

In parallel systems it may happen that some *results should be computed multiple times* by different processors, because it is easier to recompile than exchange them among the processors.