

Evaluation of Parallel Algorithms

Stefan Lang

Interdisciplinary Center for Scientific Computing (IWR)
University of Heidelberg
INF 368, Room 532
D-69120 Heidelberg
phone: 06221/54-8264
email: `Stefan.Lang@iwr.uni-heidelberg.de`

WS 14/15

Themen

Evaluation of parallel algorithms

- Speedup, efficiency
- Degree of parallelism, costs
- Isoefficiency
- Amdahl's law
- Gustafson scaling
- Scalability

Evaluation of Parallel Algorithms I

How can the properties of a parallel algorithm for the solution of a problem $\Pi(N)$ be analyzed?

- Problem size N can be chosen arbitrary
- Solution of the problem with sequential resp. parallel algorithm
- Hardware assumptions:
 - ▶ MIMD parallel computer with P identical computing nodes
 - ▶ Communication network scales with number of computing nodes
 - ▶ Latency, band width and node performance are known
- Execution of the sequential program on a single node
- Parallel algorithm + parallel implementation + parallel hardware = parallel system
- The notion of scalability characterizes the ability of a parallel system to handle increasing resources provided by processors P or demands by the problem size N .

Goal: Analysis of scalability properties of a parallel system

Evaluation of Parallel Algorithms II

Measures for Parallel Algorithms

- Runtime
- Speedup and Efficiency
- Costs
- Degree of parallelism

Evaluation of Parallel Algorithms III

Definition of different execution times:

- The **sequential execution time** $T_S(N)$ denotes the runtime of a sequential algorithm for solution of problem Π for input size N .
- The **optimal execution time** $T_{best}(N)$ characterizes the runtime of the *best* (existing) sequential algorithm to solve the problem Π with input size N . This algorithm has for nearly all sizes of N the lowest time demands.
- The **parallel runtime** $T_P(N, P)$ describes the runtime of the parallel systems, to be investigated, for solution of Π in dependence of input size N and the processor count P .

Evaluation of Parallel Algorithms IV

The measurement of runtimes allows the definition of further units:

- **Speedup**

$$S(N, P) = \frac{T_{best}(N)}{T_P(N, P)}. \quad (1)$$

For all N and P applies $S(N, P) \leq P$.

Assume that holds $S(N, P) > P$, then exists a sequential program, that simulates the parallel program (processing in time slices mode). This hypothetical program would then have a runtime $PT_P(N, P)$ and it would be

$$PT_P(N, P) = P \frac{T_{best}(N)}{S(N, P)} < T_{best}(N), \quad (2)$$

In obvious contradiction to former definitions.

- **Efficiency**

$$E(N, P) = \frac{T_{best}(N)}{PT_P(N, P)}. \quad (3)$$

It applies $E(N, P) \leq 1$. The efficiency represents the share of the maximal achievable speedup. We say that $E \cdot P$ processors really work for the solution of Π and the rest $(1 - E)P$ does not contribute effectively to the problem solution.

Evaluation of Parallel Algorithms V

- **Costs**

As costs C the product

$$C(N, P) = PT_P(N, P), \quad (4)$$

is defined, since one would have to pay this duration of computing time in the computing center.

We denote an algorithm as *cost optimal*, if $C(N, P) = \text{const}T_{\text{best}}(N)$.

Obviously applies then

$$E(N, P) = \frac{T_{\text{best}}(N)}{C(N, P)} = 1/\text{const}, \quad (5)$$

the efficiency remains thus constant.

Evaluation of Parallel Algorithms VI

- **Degree of parallelism**

With $\Gamma(N)$ we denote the *degree of parallelism*. That is the maximal number of operations, that can be executed synchronously, in the best sequential algorithm.

- ▶ Obviously could be in principle executed the more operations the more operations had to be executed overall, thus the larger N is. The degree of parallelism is such dependent on N .
- ▶ On the other side the degree of parallelism can not be larger than the number of operations that have to be executed in total. Since this number is proportional to $T_S(N)$, we can say, that

$$\Gamma(N) \leq O(T_S(N)) \quad (6)$$

holds.

Evaluation of Parallel Algorithms: Speedup

Elementary is the behaviour of the speedup $S(N, P)$ of a parallel system in dependence of P .

With the second parameter N we have the choice of different scenarios.

1. Fixed sequential execution time

- We determine N from the relation

$$T_{best}(N) \stackrel{!}{=} T_{fix} \rightarrow N = N_A \quad (7)$$

where T_{fix} is a parameter. The scaled speedup is then

$$S_A(P) = S(N_A, P), \quad (8)$$

therefore A stands for the name *Amdahl*.

- How behaves the scaled speedup?

Assumption: the parallel program is created from the best sequential program with sequential share $0 < q < 1$ and a completely parallelisable rest $(1 - q)$. The parallel runtime (for fixed N_A !) is then

$$T_P = qT_{fix} + (1 - q)T_{fix}/P. \quad (9)$$

Evaluation of Parallel Algorithms: Amdahl

For the speedup applies then

$$S(P) = \frac{T_{fix}}{qT_{fix} + (1 - q)T_{fix}/P} = \frac{1}{q + \frac{1-q}{P}} \quad (10)$$

Thus the Amdahl's law holds

$$\lim_{P \rightarrow \infty} S(P) = 1/q. \quad (11)$$

Consequences:

- The maximal achievable speedup is then determined purely by the sequential share.
- The efficiency strongly decreases, if one nearly wants to reach the maximal speedup.
- This achievement led at the end of the 60th to a very pessimistic estimation of the possibilities by parallel computing.
- This has changed first, when it has been recognized, that for most parallel algorithms the sequential share q decreases with increasing N .

The way out of this dilemma consists in solving with more processors always larger problems!

We now present three approaches how N can be increased with P .

Evaluation of Parallel Algorithms: Gustafson

2. Fixed parallel execution time

We determine N from the equation

$$T_P(N, P) \stackrel{!}{=} T_{fix} \rightarrow N = N_G(P) \quad (12)$$

for given T_{fix} and then consider the speedup

$$S_G(P) = S(N_G(P), P). \quad (13)$$

- This kind of scaling is also called „Gustafson scaling“.
- Motivation are for example applications in the area of weather forecast. Here one has a fixed time slot T_{fix} that is used to solve a problem as large as possible.

Evaluation of Parallel Algorithms: Memory Limitation

3. Fixed memory consumption per processor

Many simulation applications are memory constraint, the memory need grows as function $M(N)$. According to memory complexity not computing time since the memory needs determine what problems can be calculated with a machine.

Assumption: Let us assume, that the parallel computer consists of P identical processors, that each have memory of size M_0 , thus the scaling provides

$$M(N) \stackrel{!}{=} PM_0 \rightarrow N = N_M(P) \quad (14)$$

and we consider

$$S_M(P) = S(N_M(P), P). \quad (15)$$

as scaled speedup.

Evaluation of Parallel Algorithms: Efficiency Limitation

4. Constant Efficiency

We choose N such, that the parallel efficiency remains constant.

We require

$$E(N, P) \stackrel{!}{=} E_0 \rightarrow N = N_I(P). \quad (16)$$

This is denoted as *iso-efficient scaling*. Obviously is $E(N_I(P), P) = E_0$ thus

$$S_I(P) = S(N_I(P), P) = PE_0. \quad (17)$$

An iso-efficient scaling is not possible for any parallel system. One does not necessarily find a function $N_I(P)$, that fulfills (16) identical. Thus one can require on the other side, that a system is scalable exactly if such a function can be found.

Evaluation of Parallel Algorithms: Example I

For a deeper understanding of the notions we now consider an **example**

- We want to add N numbers on a hypercube with P processors. The approach is as follows:
 - ▶ Each has N/P numbers, that are added in the first step.
 - ▶ This P intermediate results are then added in a tree.
- We then get for the sequential computing time

$$T_{best}(N) = (N - 1)t_a \quad (18)$$

- the parallel computing time is

$$T_P(N, P) = (N/P - 1)t_a + \text{ld } P t_m, \quad (19)$$

where t_a is the time for the addition of two numbers and t_m the time for the message exchange (we assume, that $t_m \gg t_a$).

Evaluation of Parallel Algorithms: Example II

1. Fixed sequential execution time (Amdahl)

If we set $T_{best}(N) = T_{fix}$ then we get, if $T_{fix} \gg t_a$, in good approximation

$$N_A = T_{fix}/t_a.$$

For meaningful processor counts P applies: $P \leq N_A$.

For the speedup we obtain in the case of $N_A/P \gg 1$

$$S_A(P) = \frac{T_{fix}}{T_{fix}/P + \text{ld } P t_m} = \frac{P}{1 + P \text{ld } P \frac{t_m}{T_{fix}}}. \quad (20)$$

2. Fixed parallel execution time (Gustafson)

Here one obtains

$$\left(\frac{N}{P} - 1\right) t_a + \text{ld } P t_m = T_{fix} \implies N_G = P \left(1 + \frac{T_{fix} - \text{ld } P t_m}{t_a}\right). \quad (21)$$

The maximal usable processor count is again limited: $2^{T_{fix}/t_m}$. Is $\text{ld } P t_m = T_{fix}$, then when using more processors than that in every case the maximal allowed computing time is exceeded.

Despite that we can suppose, that $2^{T_{fix}/t_m} \gg T_{fix}/t_a$ holds.

Evaluation of Parallel Algorithms: Example III

The scaled speedup S_G is under the assumption $N_G(P)/P \gg 1$:

$$S_G(P) = \frac{N_G(P)t_a}{N_G(P)t_a/P + \text{ld } P t_m} = \frac{P}{1 + \text{ld } P \frac{t_m}{T_{fix}}}. \quad (22)$$

It applies $N_G(P) \approx PT_{fix}/t_a$.

For same processor count is then S_G greater than S_A .

3. Fixed memory per processor (memory limitation)

If the memory demands are $M(N) = N$, then applies for $M(N) = M_0P$ the scaling

$$N_M(P) = M_0P.$$

We can now use an unlimited number of processors, on the other hand the parallel computing time increases also unlimited. For the scaled speedup we get:

$$S_M(P) = \frac{N_M(P)t_a}{N_M(P)t_a/P + \text{ld } P t_m} = \frac{P}{1 + \text{ld } P \frac{t_m}{M_0 t_a}}. \quad (23)$$

For the choice $T_{fix} = M_0 t_a$ this is the same formula as S_G . In both cases we see, that the efficiency decreases with P .

Evaluation of Parallel Algorithms: Example IV

4. Iso-efficient scaling

We choose N such, that the efficiency remains constant, resp. the speedup grows linearly:

$$S = \frac{P}{1 + \frac{P \text{Id} P}{N} \frac{t_m}{t_a}} \stackrel{!}{=} \frac{P}{1 + K} \implies N_I(P) = P \text{Id} P \frac{t_m}{K t_a},$$

for an arbitrary choosable $K > 0$. Since $N_I(P)$ exists, the algorithms can be regarded as scalable. For the speedup applies $S_I = P/(1 + K)$.

Iso-efficiency Analysis I

We now introduce further a formalism to clarify the principle of iso-efficient scaling

- Goal answering of questions:
„Is this algorithm for matrix multiplication on the hypercube better scalable than that for fast fourier transform on the array topology“
- Problem size: Parameter N has been chosen up to now arbitrary.
- N can denote in matrix multiplication either the number of matrix elements or too the number of elements per row.
- In this situation the first case would lead to $2N^{3/2}t_f$, whilst the second case $2N^3t_f$ for the sequential runtime.
- Meaningful comparison of algorithms necessitates invariance of the cost measure regarding the choice of the parameter for problem size.

Iso-efficiency Analysis II

- We choose as measure for the costs W of a (sequential) algorithm its execution time, we therefore define

$$W = T_{best}(N) \quad (24)$$

itself. This execution time is furthermore proportional to the number of operations to be executed in the algorithms.

- For the degree of parallelism Γ we obtain:

$$\Gamma(W) \leq O(W),$$

since there can not be executed more operations in parallel as there are operations in total.

- Via $N = T_{best}^{-1}(W)$ we can write

$$\tilde{T}_P(W, P) = T_P(T_{best}^{-1}(W), P),$$

where we however leave away the $\tilde{\sim}$ sign in the following.

Iso-efficiency Analysis III

We define the *Overhead* as

$$T_o(W, P) = PT_P(W, P) - W \geq 0. \quad (25)$$

$PT_P(W, P)$ is the time, that a simulation of the sequential program would need on one processor. This is in every case not smaller than the best sequential execution time W . The overhead contains additional computing time because of communication, load imbalance and „superfluous“ calculations.

Isoefficiency function From the overhead we obtain

$$T_P(W, P) = \frac{W + T_o(W, P)}{P}.$$

thus we obtain for the speedup

$$S(W, P) = \frac{W}{T_P(W, P)} = P \frac{1}{1 + \frac{T_o(W, P)}{W}},$$

resp. for the efficiency

$$E(W, P) = \frac{1}{1 + \frac{T_o(W, P)}{W}}.$$

In the sense of an iso-efficient scaling we now ask: How needs W to grow as function of P that the efficiency remains constant. Because of the formula above this is the case when $T_o(W, P)/W = K$, with an arbitrary constant $K \geq 0$. The efficiency is then $1/(1 + K)$.

Iso-efficiency Analysis IV

- A function $W_K(P)$ is called *iso-efficiency function* if it fulfills the equation

$$T_o(W_K(P), P) = KW_K(P)$$

identical.

- A parallel system is called scalable (exactly) iff it has an iso-efficiency function.
- The asymptotic growing of W with P is a measure for the scalability of the system:
Has for example a system S_1 an iso-efficiency function $W = O(P^{3/2})$ and a system S_2 an iso-efficiency function $W = O(P^2)$ then S_2 scales *worse* than S_1 .

Iso-efficiency Analysis V

When is there an iso-efficiency function?

- We progress from the efficiency

$$E(W, P) = \frac{1}{1 + \frac{T_o(W, P)}{W}}$$

- for *fixed* W and growing P . It holds for each parallel system, that

$$\lim_{P \rightarrow \infty} E(W, P) = 0$$

as can be seen by the following thoughts: Since W is fixed, also the degree of parallelism is fixed and then there exists a lower bound for the parallel computing time: $T_P(W, P) \geq T_{min}(W)$, this means the calculation can not be faster than T_{min} , without dependance on the number of used processors. Thus however implies asymptotically

$$\frac{T_o(W, P)}{W} \geq \frac{PT_{min}(W) - W}{W} = O(P)$$

and therefore the efficiency drops against 0.

Iso-efficiency Analysis VI

If we consider now the efficiency at *fixed* P and growing work W , then applies for many (not all!) parallel systems, that

$$\lim_{W \rightarrow \infty} E(W, P) = 1.$$

Obviously this means regarding the efficiency formula, that

$$T_o(W, P)|_{P=\text{const}} < O(W) \quad (26)$$

for fixed P the overhead grows less than linear with W . In this case for each P a W can be found such that a desired efficiency is achieved. Equation (26) ensures such the existence of an iso-efficiency function. For example the matrix transposition can be encountered as a not scalable system. We will later derive, that the overhead amounts in this case $T_o(W, P) = O(W \text{Id } P)$. Such no iso-efficiency function can exist.

Optimal parallelisable systems

We want to analyze now the question how iso-efficiency functions have to grow at least. For this we remark finally, that

$$T_P(W, P) \geq \frac{W}{\Gamma(W)},$$

since $\Gamma(W)$ (dimensionless) is the maximal count of operations executed synchronously in the sequential algorithms for effort W . Therefore $W/\Gamma(W)$ is a lower bound for the parallel computing time.

Iso-efficiency Analysis VII

Now there can surely not be executed more operations in parallel than can be executed in total, thus holds $\Gamma(W) \leq O(W)$. We want to denote a system as *optimal parallelisable*, if

$$\Gamma(W) = cW$$

holds with a constant $c > 0$. Now applies

$$T_P(W, P) \geq \frac{W}{\Gamma(W)} = \frac{1}{c},$$

the minimal parallel computing time remains constant. For the overhead we obtain that in this case

$$T_o(W, P) = PT_P(W, P) - W = P/c - W$$

and such for the iso-efficiency function

$$T_o(W, P) = P/c - W \stackrel{!}{=} KW \iff W = \frac{P}{(K+1)c} = O(P).$$

Optimal parallelisable systems such have an iso-efficiency function $W = O(P)$. We remark thus, that a iso-efficiency function grows at least linear with P .

In the following lectures we will determine the iso-efficiency functions for a series of algorithms, therefore we relinquish for an extensive example here.