Isoefficiency : Measuring the Scalability of Parallel Algorithms and Architectures

Ananth Y. Grama, Anshul Gupta, and Vipin Kumar
University of Minnesota

presented by Rui Xu
03/10/2021

Overview

- Scalable parallel systems
- The isoefficiency function
  - ➢Terminology, definitions, and assumptions
  - ➢Cost-optimal
  - ➢Degree of concurrency
- Isoefficiency analysis
  - ➢Comparing two parallel algorithms
  - ➢Effects of machine specific parameter
  - ➢Impact of concurrency on scalability
  - ➢Impact of contention for shared data structures
- Summary

Scalable parallel systems

- Increasing the number of processors reduces efficiency
- Increasing the problem size increases efficiency
- Scalable parallel systems: keep efficiency constant by increasing both

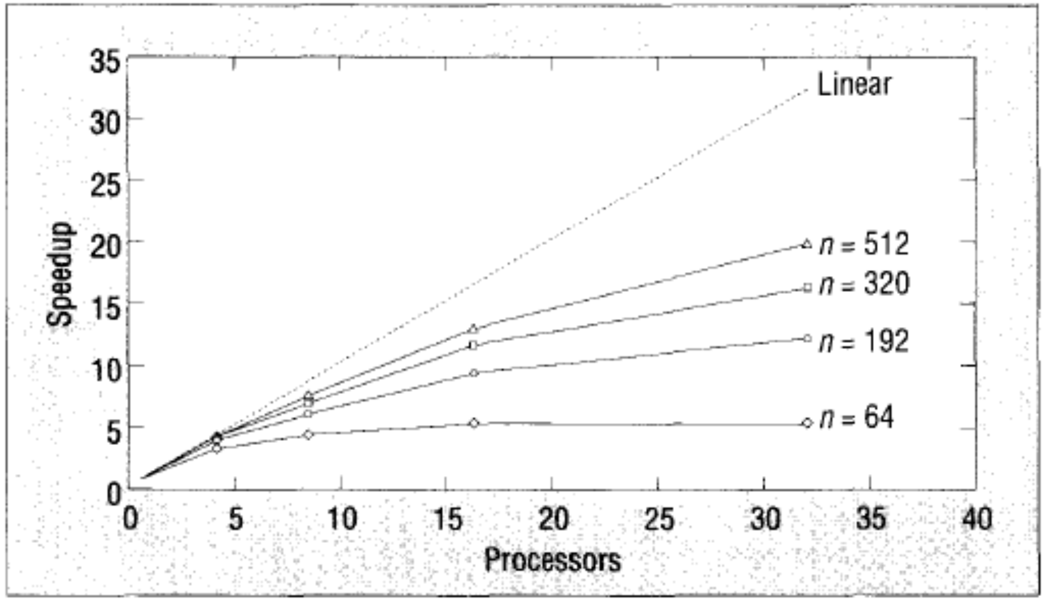


Figure 1. Speedup versus number of processors for adding a list of numbers on a hypercube.

Table 1. Efficiency as a function of n and p for adding n numbers on p-processor hypercubes.

	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
$n = 64$	1.0	.80	.57	.33	.17
$n = 192$	1.0	.92	.80	.60	.38
$n = 320$	1.0	.95	.87	.71	.50
$n = 512$	1.0	.97	.91	.80	.62

$$pT_P = T_1 + T_o$$

$$T_P = \frac{T_1 + T_o}{p}$$

$$S = \frac{T_1}{T_P} = \frac{pT_1}{T_1 + T_o}$$

$$\begin{aligned} E &= \frac{S}{p} \\ &= \frac{T_1}{T_1 + T_o} \\ &= \frac{1}{1 + \frac{T_o}{T_1}} \end{aligned}$$

Terminology

- Terminology and definition
  - Sequential execution time (***T_1***): the execution time to run an algorithm on a single processor
  - Parallel execution time (***T_p***): the execution time of the corresponding parallel algorithm on ***p*** identical processors
  - Total overhead (***T_0***): the sum total of time spent by all processors doing work which is not done by the sequential algorithm
  - The speedup (***S***): ratio of sequential execution time to the parallel execution time
  - The efficiency (***E***): ratio of the speedup to the number of processors used

Definitions

- Assuming the sequential execution time $T_1 = W \times t_c$, where ***W*** is the problem size and $\boldsymbol{t_c}$ is the cost of executing each operation

$$E = \frac{1}{1 + \frac{T_0}{W t_c}}$$

> ➢If ***W*** = constant and ***p*** increases, ***E*** decreases because the total overhead $\boldsymbol{T_0}$ will increase
>
> ➢If ***p*** = constant and ***W*** increases, ***E*** increases for scalable parallel systems because $\boldsymbol{T_0}$ grows slower than $\boldsymbol{\Theta}(\boldsymbol{W})$

The isoefficiency function

- Highly scalable system: ***W*** needs to grow only linearly with respect to ***p*** to maintain ***E*** at a desired value ($\boldsymbol{W}$ = $\boldsymbol{K}\,\boldsymbol{T_0}$, where ***K*** is a function of ***E*** and $\boldsymbol{t_c}$)

- ***W*** = f(***p***) is the isoefficiency function, assuming that the efficiency of the parallel systems can be kept constant

- A small isoefficiency function means highly scalable, i.e. ***W*** = $\boldsymbol{\Theta}(\boldsymbol{p}^3)$ which means the problem size should grow $\boldsymbol{\Theta}(\boldsymbol{p}^3)$ to maintain the same efficiency

$$\frac{T_o}{W} = t_c\left(\frac{1-E}{E}\right)$$

$$W = \frac{1}{t_c}\left(\frac{E}{1-E}\right)T_o$$

$$K = \frac{1}{t_c}\left(\frac{E}{1-E}\right)$$

Cost-optimal

- A parallel system is cost-optimal if and only if the product of the number of processors and the parallel execution time is proportional to the execution time of the best serial algorithm on a single processor:

$$pT_p \propto W$$

or

$$W \propto T_0$$

- The lower bound of $\boldsymbol{W} = \boldsymbol{\Theta}(\boldsymbol{p})$, which is ideally scalable parallel system

Degree of concurrency

- If ***C(W)*** is an algorithm's degree of concurrency, then given a problem of size ***W***, at most ***C(W)*** processors can be employed effectively.

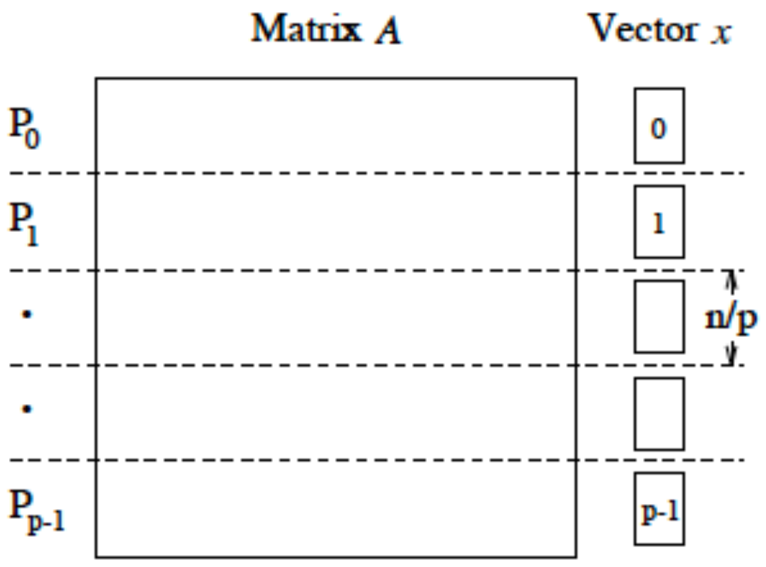
- For example, given a problem of size ***W***, at most $\boldsymbol{\Theta}\left(\boldsymbol{W}^{2/3}\right)$ pocessors can be used, so given ***p*** processors, the size of the problem should be at least $\boldsymbol{\Theta}\left(\boldsymbol{p}^{3/2}\right)$ in order to use all the processors.

- Thus, isoefficiency function due to concurrency is $\boldsymbol{\Theta}\left(\boldsymbol{p}^{3/2}\right)$

- System's overall isoefficiency function is the maximum of the isoefficiency functions due to concurrency, communication, and other overhead

Isoefficiency analysis: stripe based matrix-vector product on a hypercube

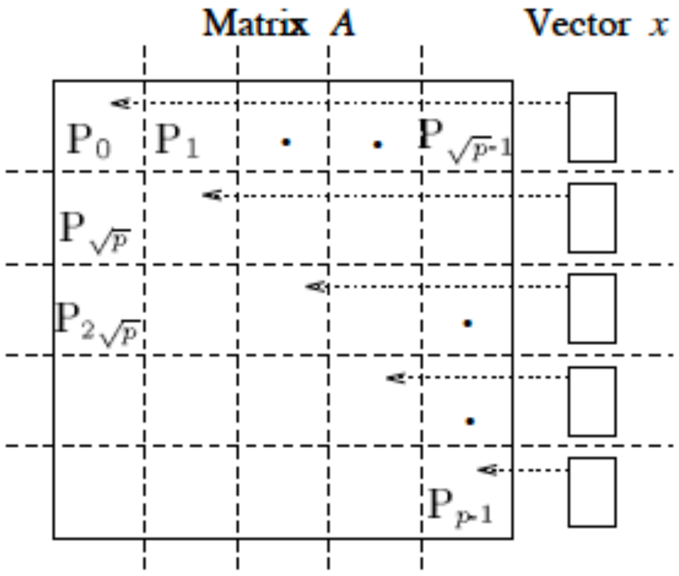
- The problem of multiplying an $n \times n$ matrix with an $n \times 1$ vector
- Problem size, $\boldsymbol{W} = n^2$
- Parallel execution time $$T_P = t_c \frac{n^2}{p} + t_s \log p + t_w n$$
- Total overhead $$T_o = t_s p \log p + t_w np$$
- $W = Kt_s p\log(p)$ and $W = K^2 {t_w}^2 p^2$
- So isoefficiency function is $\boldsymbol{\Theta(p^2)}$



(a) Initial partitioning of the matrix and the starting vector x

Isoefficiency analysis: checkerboard based matrix-vector product on a hypercube

- Parallel execution time $T_P = t_c \frac{n^2}{p} + t_s + 2t_s \log\sqrt{p} + 3t_w \frac{n}{\sqrt{p}} \log\sqrt{p}$
- Total overhead $T_o = t_s p \log p + \frac{3}{2} t_w n \sqrt{p} \log p$
- $W = Kt_s p\log(p)$ and $W = K^2 \frac{9}{4} \frac{{t_w}^2}{{t_c}^2} p\log^2 p$
- Overall isoefficiency is $\boldsymbol{\Theta(p\log^2 p)}$
- The checkerboard algorithm has a higher scalability



(a) Initial data distribution and communication steps to align the vector along the diagonal

Effects of machine specific parameter

- The effects of processor and communication speeds
- Cooley-Tukey algorithm for computing n-point, single dimensional unordered radix-2 FFT
- Isoefficiency function: $W = t_s p\log(p)$ and $W = Cp^C\log(p)$, where

$$C = \frac{E}{1-E}\frac{t_w}{t_c}$$

- If ***C*** < 1: $\boldsymbol{\Theta(p\log(p))}$ else: $\boldsymbol{\Theta(p^c\log(p))}$
- And ***C*** is hardware dependent parameter, which depends on CPU speed and communication bandwidth

Impact of concurrency on scalability

- Dijkstra’s All Pair’s Shortest Path Algorithm
- The best-known serial algorithm takes $O(n^3)$ time
- A simple parallel version by executing a single-source shortest-path problem independently on each processor with $O(n^2)$ time
- This simple algorithm can use at most n processors
- And since the problem size is $O(n^3)$, problem size must grow at least $\boldsymbol{\Theta(p^3)}$ to use more processors and maintain constant efficiency
- So in this algorithm, isoefficiency is dominated by concurrency and absence of communication here is no longer an advantage

Impact of contention for shared data structures

- Dynamic load balancing
- Isoefficiency due to communication overhead is $\boldsymbol{\Theta}\left(\mathbf{p}\log^2 \boldsymbol{p}\right)$
- Only one processor can access the global variable at a time; we must also analyze the system's isoefficiency due to contention
- At some point, the shared variable access becomes a bottleneck, and the overall execution time cannot be reduced further
- We can eliminate this bottleneck by increasing ***W*** at a rate such that the ratio between ***W/p*** and O(***p log W***) remains the same
- Thus, isoefficiency due to contention is $\boldsymbol{\Theta}\left(\boldsymbol{p}^2\log \boldsymbol{p}\right)$
- Overall isoefficiency is $\boldsymbol{\Theta}\left(\boldsymbol{p}^2\log \boldsymbol{p}\right)$

Summary

- If the problem size grows at the rate specified by the isoefficiency function, then the system’s speedup is linear

- For a class of parallel systems, the isoefficiency function specifies the relationship between the problem size’s growth rate and the number of processors on which the problem executes in minimum time