Parallel Programming

with MPI and OpenMP

Michael J. Quinn

Chapter 7

Performance Analysis

Learning Objectives

- Predict performance of parallel programs
- Understand barriers to higher performance

Outline

- General speedup formula
- Amdahl's Law
- Gustafson-Barsis' Law
- Karp-Flatt metric
- Isoefficiency metric

Speedup Formula

$$\Psi(p) = \frac{\text{Sequential execution time}}{\text{Parallel execution time}} = \frac{T(1)}{T(p)}$$

Speedup: ψ Greek letter psi, for pseedup ☺

Execution Time Components

- Inherently sequential computations:
  sequential part: $\sigma(n)$
- Potentially parallel computations:
  parallelizable part: $\varphi(n)$
- Communication operations:
  communication part: $\kappa(n,p)$
- σ, φ, κ Greek letters sigma, phi, kappa

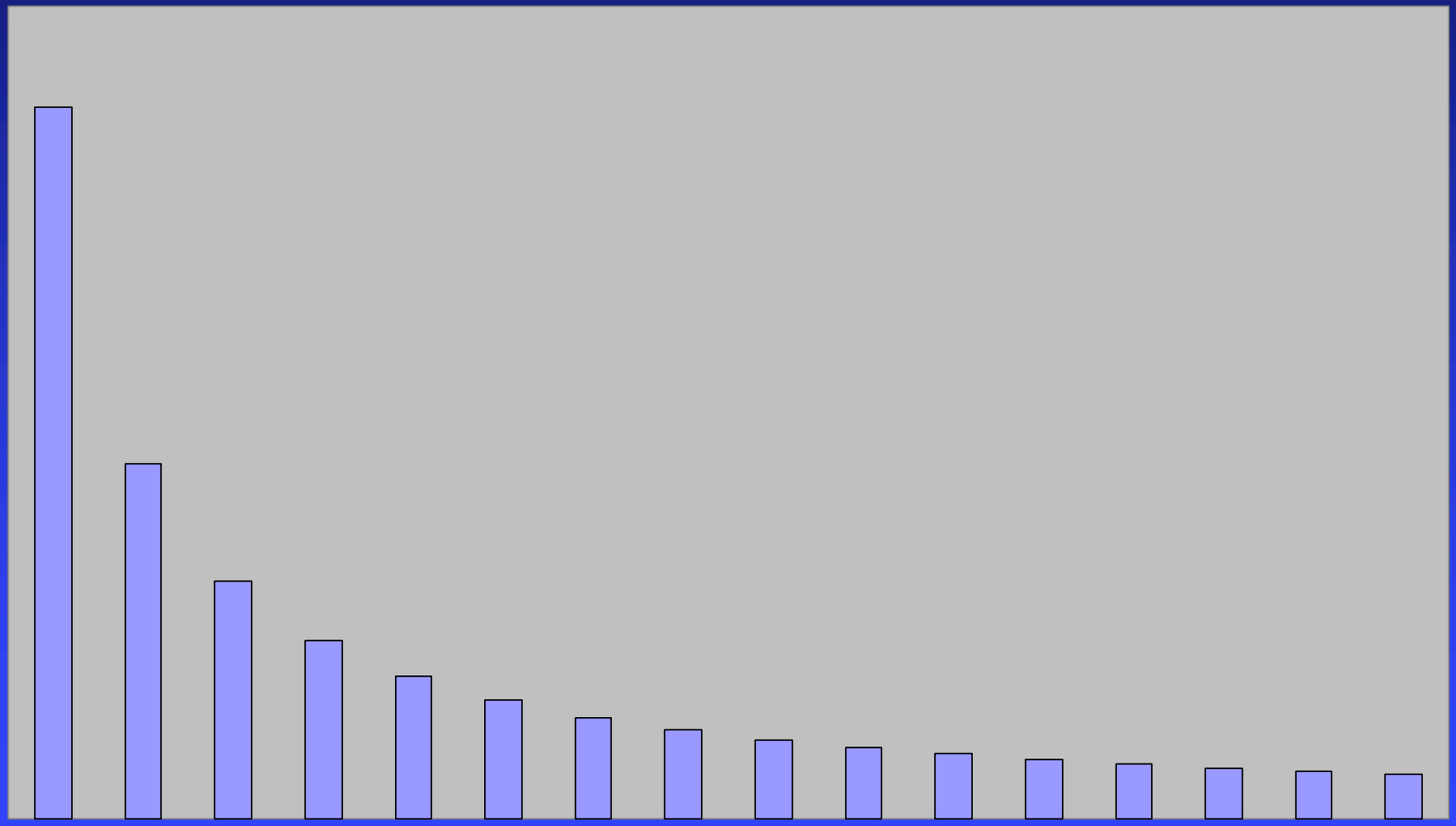
Speedup Expression

$$\psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/\ p+\kappa(n,p)}$$

Speedup(n,p) = T1 / Tp

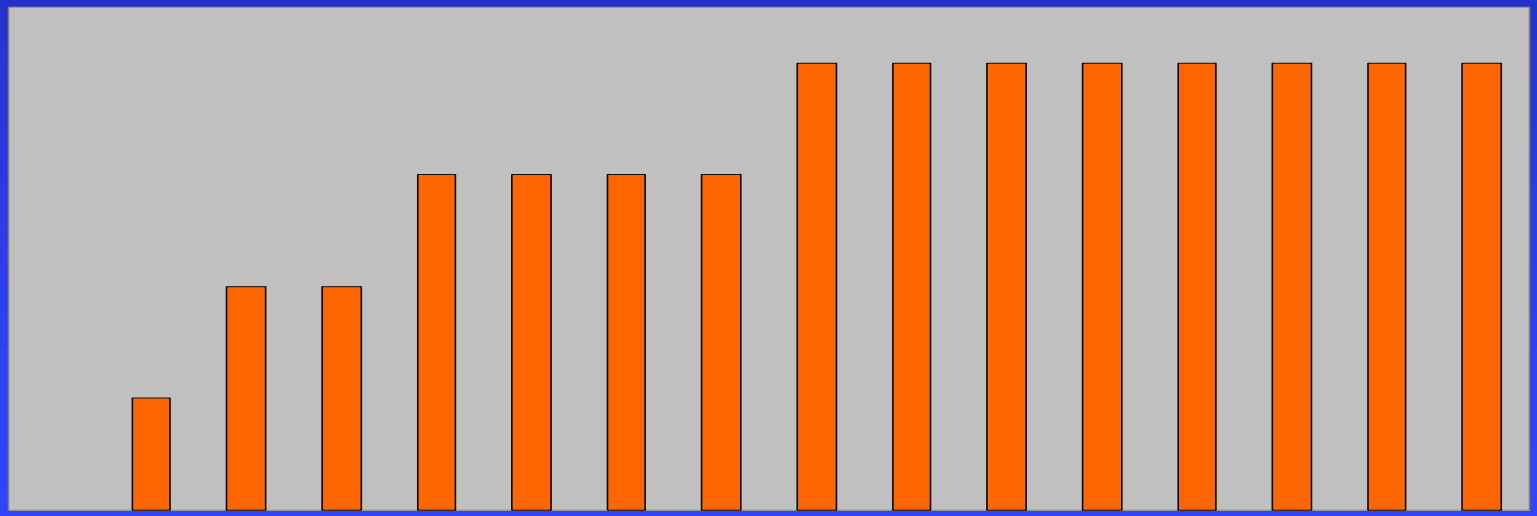
$\sigma(n)$: Sequential part
$\varphi(n)$: Parallelizable part
$\kappa(n,p)$: Communication overhead

parallel part / #processors: φ(n)/p

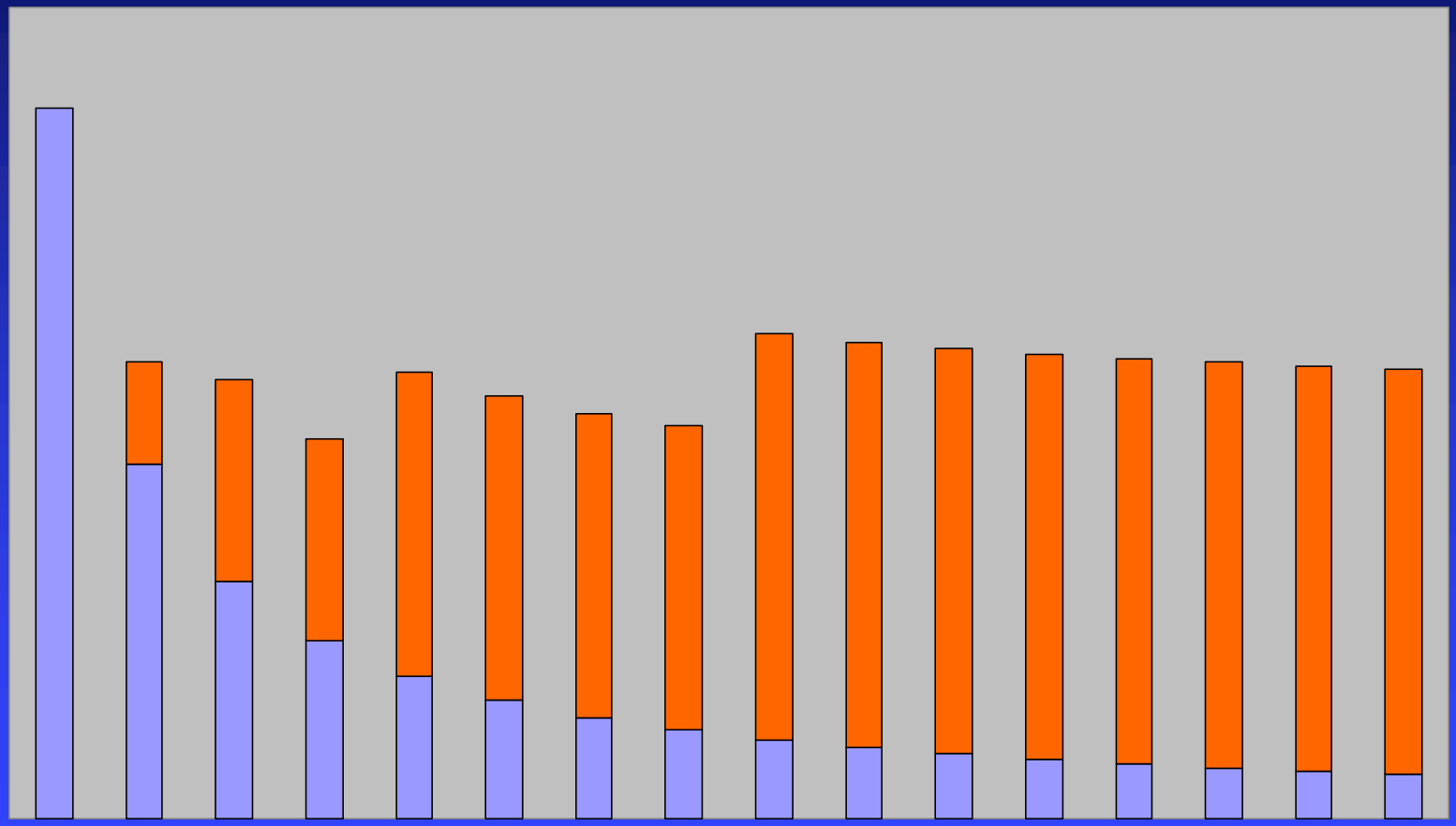


communication cost: $\kappa(n,p)$

A possible communication cost

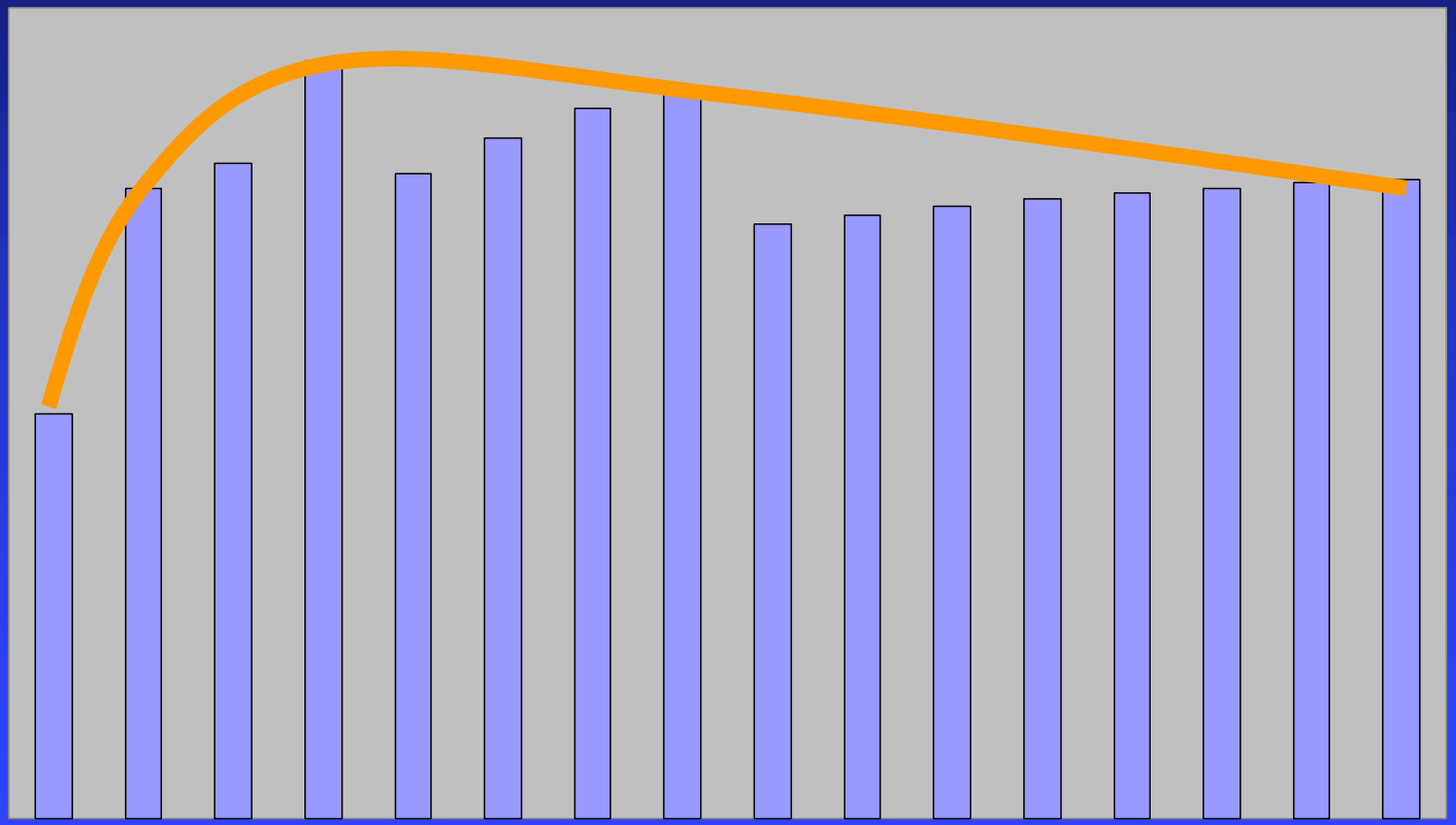


$$\varphi(n)/p + \kappa(n,p)$$



Speedup Plot

"elbowing out"



Efficiency ε (epsilon)

$$\text{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

$$\textit{Efficiency} = \frac{\text{Speedup}}{\#\,\text{Processors}} = \frac{T(1)}{pT(p)}$$

$$0 \leq \varepsilon(n,p) \leq 1$$

$$\varepsilon(n,p) = \frac{T(1)}{pT(p)} = \frac{\sigma(n)+\phi(n)}{p\sigma(n)+\phi(n)+p\kappa(n,p)}$$

All terms $> 0 \Rightarrow \varepsilon(n,p) > 0$

Denominator >= numerator $\Rightarrow \varepsilon(n,p)$ <= 1

T(1): cost of the sequential execution
pT(p): cost of the parallel execution

Amdahl's Law

$$\psi(n,p) = \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)}$$

$$\leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p} \qquad (\kappa(n,p)>0)$$

Amdahl ignored communication cost (he was thinking in terms of vector processors) and normalized T(1) = 1, f is sequential fraction, (1-f) parallel fraction

$$\psi \leq \frac{1}{f+(1-f)/p}$$

Example 1

- 95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$\psi \leq \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

- What's the best speedup you'll ever get?

20

Example 2

- 20% of a program's execution time is spent within inherently sequential code. What is the limit to the speedup achievable by a parallel version of the program?

$$\lim_{p \to \infty} \frac{1}{0.2 + (1 - 0.2)/p} = \frac{1}{0.2} = 5$$

Pop Quiz

- An oceanographer gives you a serial program and asks you how much faster it might run on 8 processors. You can only find one function amenable to a parallel solution. Benchmarking on a single processor reveals 80% of the execution time is spent inside this function. What is the best speedup a parallel version is likely to achieve on 8 processors?

Pop Quiz

- A computer animation program generates a feature movie frame-by-frame. Each frame can be generated independently and is output to its own file. If it takes 99 seconds to render a frame and 1 second to output it, how much speedup can be achieved by rendering the movie on 100 processors, assuming that the frame can be rendered in parallel, but that the frame must be written out sequentially?

Limitations of Amdahl's Law

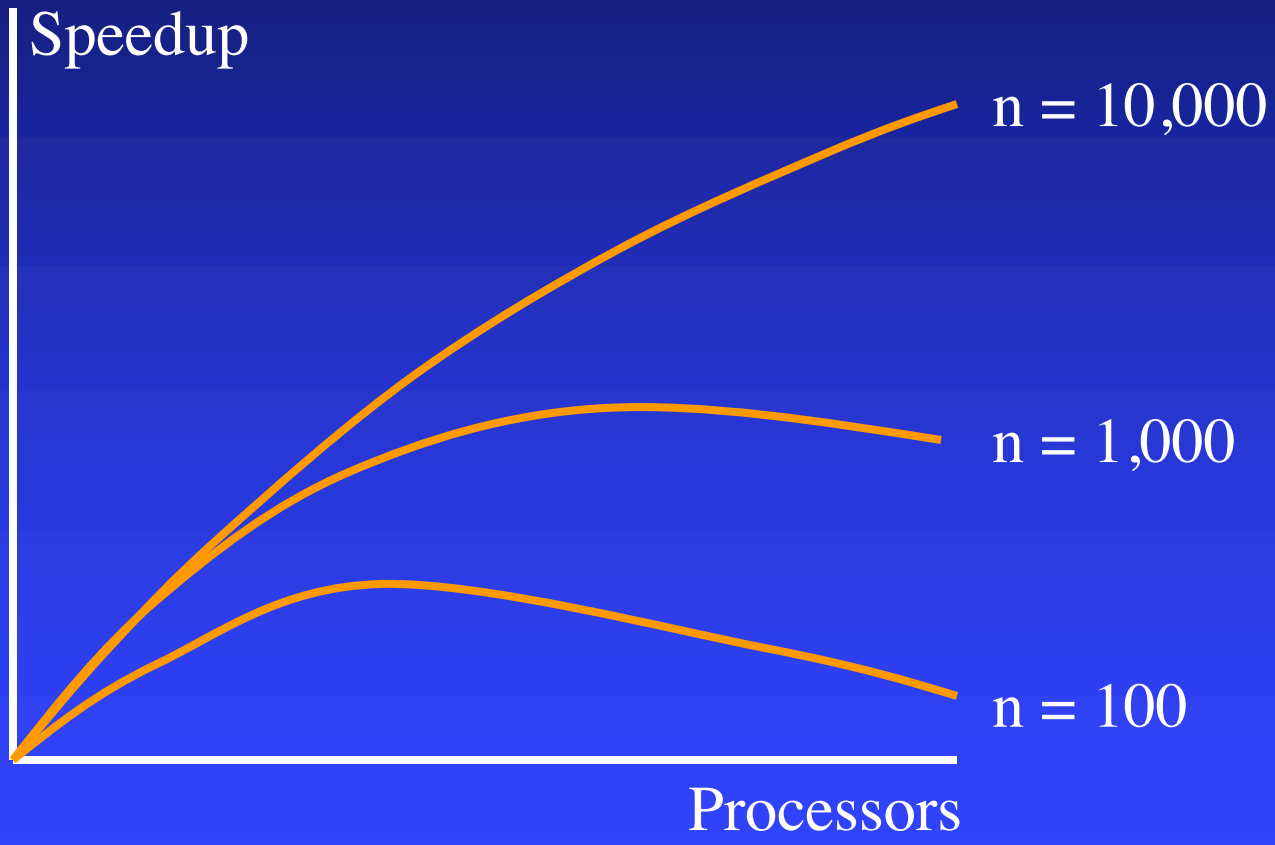
- Ignores $\kappa(n,p)$
- assumes that parallel fraction is "perfectly parallelizable" taking (1-f)/p
- therefore **over**estimates speedup achievable

Amdahl Effect

- Typically $\kappa(n,p)$ has lower complexity than $\varphi(n)/p$
- As n increases, $\varphi(n)/p$ dominates $\kappa(n,p)$
- Amdahl Effect:

  as n increases, speedup increases

Illustration of Amdahl Effect



Another Perspective

- Amdahl looks at the performance of one program on a varying #processors.

  This is not what happens in practice
  - e.g., a small problem does not benefit from parallelism, while a large problem does not fit on a sequential machine
  - We often use faster/larger computers (more processors, more local memory) to solve larger problems
- **Let's treat time as a constant and allow problem size to increase with number of processors**

Gustafson-Barsis's Law

- Problem size is an increasing function of *p*
  - **This is what we did in the 90s**

    e.g. Adaptive Quadrature on the Sandia nCube (16K processors)
  - **tiny n on 1-8 PEs, small n on 4-32 PEs,**
  - **medium n on 16-128 PEs, large n on 64-512 PEs, very large on 256-2K PEs**

    (we never got more than 2K PEs)

More metrics …

- Karp and Flatt proposed another metric

  establishing the serial fraction, including startup, synchronization, overhead

  based on actual performance measurements

- Iso-efficiency metric establishes a relationship between n and p and efficiency **taking machine topology** into account

Iso-efficiency: summing on a grid

- 1 time-step to add, 500 to communicate
- time $T_p = N/p + 500*4*(\sqrt{p}-1)$
- speedup $S_p = T_1/T_p = Np/(N+2000p(\sqrt{p}-1))$
- efficiency $E_p = S_p/p = N/(N+2000p(\sqrt{p}-1))$
- Make a table E_p
  - rows N= 80,000 960,000 8,960,000
  - cols p = 4 16 64

$E_p = N/(N+2{,}000p(\sqrt{p}-1))$	p = 4	p = 16	p = 64
N = 80,000	**80 / 88 ~ 91%**	80 / 176 ~ 45%	80 / 976 ~8%
N = 960,000	960 / 968 ~99%	**960 / 1056 ~91%**	960 / 1856 ~51%
N = 8,960,000	8960/8968 ~100%	8960/9056 ~99%	**8,960/(8,960+896) ~91%**

All diagonal values: 10/11

Observations

- Down the column: larger n → larger E
- Down the row: larger p → smaller E
- to keep E = 10/11
  when growing p, we need to grow n

Formalization of Gustafson

Scalability

- Ability to keep the efficiency fixed,

  when p is increasing, provided we also increase n

- e.g. Add n numbers on p processors
  - Look at the (n,p) efficiency table
  - Efficiency is fixed with p increasing **only if** n is increased

Quantified..

- Efficiency is fixed with p increasing **only if** n is increased
- How much for this example?

$$E = n / (n + 2{,}000p(\sqrt{p}-1)) = 10/11$$

$$10(n + 2{,}000p(\sqrt{p}-1)) = 11n$$

$$n = 20{,}000p(\sqrt{p}-1)$$

Check with the table

- $20{,}000\ p\ (\sqrt{p}-1)$

- $p = 4$: $n = 20{,}000 * 4*1 = 80{,}000$
- $p = 16$: $n = 20{,}000 * 16*3 = 960{,}000$
- $p = 64$: $n = 20{,}000 * 64*7 = 8{,}960{,}000$

ISO Efficiency

- The Order of magnitude p in terms of n to keep the efficiency constant

- Iso-efficiency of **adding numbers on a 2 D grid**:

$$n = O(p \sqrt{p})$$

Summary (1/3)

- Performance terms
  - Execution time, Speedup, Efficiency
- Model of speedup
  - Serial component, Parallel component
  - Communication component
  - Overhead

Summary (2/3)

- What prevents ideal speedup?
  - Serial operations
  - Communication operations
  - Process start-up
  - Imbalanced workloads
  - overheads (architectural, computational)

Summary (3/3): Use

- Amdahl's Law
  - for understanding lack of speedup
- Gustafson-Barsis' Law
  - for understanding scalability
- Iso-efficiency metric
  - for understanding cost on a particular machine topology