Parallel Program Performance Analysis

Chris Kauffman

Last Updated:
Wed Oct 13 09:26:00 AM CDT 2021

Logistics

This Week

- Reading: Grama Ch 5
- Finish up Dense Matrix Algorithms - Parallel LU decomposition
- Discuss Performance Analysis and Isoefficiency
- Discuss Mini-Exam Problem 3

Next Week

- Parallel Sorting
- Wed 10/20 Mini-Exam 2

Grading

- A1 Regrade Requests Due Today
- Mini-Exam 1 complete, will publish after class

A2 Still Cooking

- Will have ~2 weeks to work on it irrespective of release
- Will adjust schedule as needed

Basic Definitions

- Have discussed a variety of parallel algorithms, described communication costs, total parallel runtime
- Now time for a bit more rigor on how to measure "success" of parallelizing algorithms
- Start with a few basic terms for serial/parallel system to solve the same problem

$T_s = T_{ser} = W$	Serial runtime for best serial algorithm, "Work"
P	Number of processors in a parallel system
$T_p = T_{par}$	Parallel runtime a parallel system to solve a problem
$S = T_s/T_p$	Speedup for a parallel system over a serial system
$E = S/P$	Efficiency: speedup divided by #procs, range 0..1
$C = T_p \times P$	Cost of a parallel system to solve problem

Amdahl's Law and Parallel Overhead

Speedup is limited by the portion of the program that can be parallelized and the degree to which that portion can be parallelized.

Relates to Efficiency in the following way

$$E = \frac{S}{P} = \frac{T_{ser}}{T_{par}}\frac{1}{P} = \frac{T_{ser}}{T_{par} \times P} = \frac{T_{ser}}{C}$$

- If we get Perfect speedup, $T_{par} = T_{ser}/P$ so Efficiency $E = 1$.
- But you can't get perfect speedup since all problems have serial portions that can't be parallelized
- Most parallel programs also require communication not required in serial programs

Leads to notion of **Parallel Overhead**

$$T_o = T_{over} = T_{par} \times P - T_{ser}$$

T_{over} is 0 only when Efficiency is 1 and speedup is perfect, but they never are...

Exercise: Expensive Sum of N Numbers

- Standard serial algorithm to sum N numbers takes $T_{ser} = N$ steps
- Describe a parallel algorithm which uses
  - $P = N$ processors and achieves
  - $T_{par} = 2 \times \log_2 N$ steps
  - 1 step can be add 2 numbers or communicate single number
- For 8, 32, and 1024 processors, calculate
  - Runtimes T_{ser} and T_{par} in "steps"
  - Speedup: $S = T_{ser}/T_{par}$
  - Efficiency: $E = S/P$
  - Cost: $C = T_{par} \times P$
- What happens to efficiency as N increases
- Give an analytic expression for the Efficiency of this algorithm for any N
- Is this algorithm worth the cost in terms of processors?

Answers: Expensive Sum of N Numbers

Describe a parallel algorithm which uses

- $P = N$ processors and achieves
- $T_{par} = 2 \times \log_2 N$ steps

Each processor holds 1 of the N numbers. Odd processors send their number to even processors which then add them on to their own. Then proc#'s not divisible by 4 send to those that are which add again. Each iteration takes 2 steps (send/add) and halves the number of processors leading to $2 \log N$ steps.

		8	32	1024
Runtimes	T_{ser}, T_{par}	8,6	32,10	1024,20
Speedup	$S = T_{ser}/T_{par}$	1.33	3.20	51.2
Efficiency	$E = S/P$	0.16	0.10	0.05
Cost	$C = T_{par} \times P$	48	320	20480

Efficiency is $E = \frac{1}{2 \log N}$

- Big speedup, very costly with diminishing efficiency

Exercise: Realistic Summing

- Adding N numbers on $P < N$ processors can be done in $N/P + 2 * \log_2 P$ steps. How?
- Standard serial algorithm takes N steps to sum N numbers.
- Fill in the following table

P	N	Par Runtime T_p	Speedup $S = T_s/T_p$	Efficiency $E = S/P$	Cost $C = T_p \times P$
4	64				
8	64				
8	192				
16	192				
16	512				

- What happens to efficiency as N and P vary
- Give an analytic expression for the Efficiency for any N, P
- How fast does N need to increase to maintain efficiency?

Answers: Realistic Summing

Adding N numbers on $P < N$ processors can be done in $N/P + 2 \times \log_2 P$ steps. How?

- Each Proc starts with N/P numbers, sums in that many steps, then performs a parallel reductions completing in $2\log_2\{\mathrm{P}\}$ steps.

P	N	Par Runtime T_p	Speedup $S = T_s/T_p$	Efficiency $E = S/P$	Cost $C = T_p \times P$
4	64	20	3.20	0.80	80
8	64	14	4.57	0.57	112
8	192	30	6.40	0.80	240
16	192	20	9.60	0.60	320
16	512	40	12.80	0.80	640
32	512	26	19.69	0.62	832

See also Table 5.1

Answers: Realistic Summing 2 / 3

- What happens to efficiency as N and P vary
  - Fixed P, increasing N raises Efficiency
  - Fixed N, increasing P lowers Efficiency

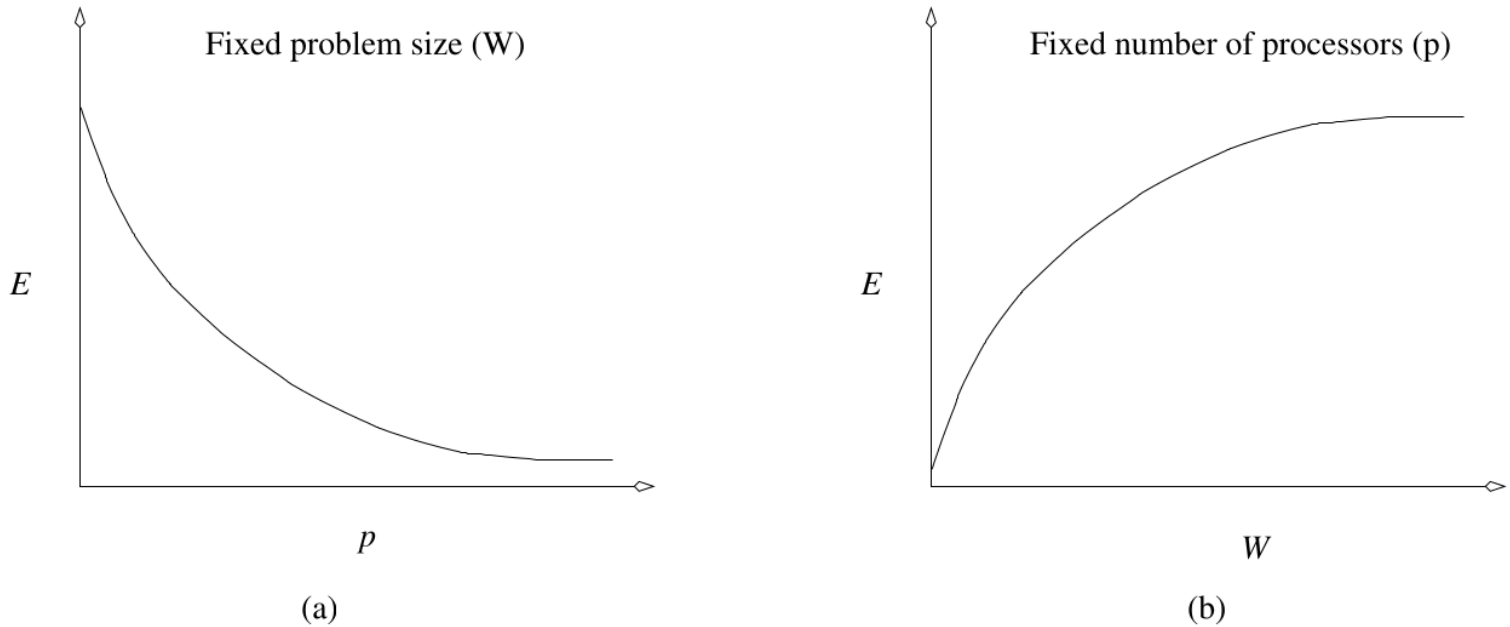

Figure 5.9 Variation of efficiency: (a) as the number of processing elements is increased for a given problem size; and (b) as the problem size is increased for a given number of processing elements. The phenomenon illustrated in graph (b) is not common to all parallel systems.

Answers: Realistic Summing 3 / 3

- Give an analytic expression for the Efficiency for any N, P
  - $E = S/P = \frac{N}{N/P + 2\log_2 P} \times \frac{1}{P} = \frac{1}{1 + \frac{2P\log_2 P}{N}}$
- How fast does N need to increase to maintain efficiency?
  - This is what **Isoefficiency** seeks to answer

Isoefficiency and Parallel Overhead

- **Isoefficiency** describes how to increase number of procs P and problem size to maintain constant efficiency E
- Summarized in the following relationship

$$W = KT_o(W, P)$$

  - $W = T_{ser}$: "work" required to solve a problem serially
  - $K = E/(1 - E)$: constant reflecting a target efficiency
  - $T_o(W, P)$: overhead of parallel system (algorithm + hardware)
- Smaller isoefficiency is better, indicates more processors can be added
- Note use of T_o as a function of Work/Problem Size W and number of Processors P

Isoefficiency for Summing N numbers on P Processors

$$T_{ser} = N \qquad T_{par} = \frac{N}{P} + 2\log_2 P$$

$$\begin{aligned} T_o(N,P) &= T_{ser} - P \times T_{par} \\ &= N - P \times \left(\frac{N}{P} + 2\log_2(P)\right) = N - N + 2P\log_2(P) \\ &= 2P\log_2(P) \\ W &= KT_o(W,P) = K \times 2P\log_2(P) \end{aligned}$$

Work / Problem size must increase at the same rate as Parallel Overhead to maintain constant efficiency

Example Isoefficiency Calculation

Summing N numbers with P Processors has isoefficiency function

$$W = KT_o(W, P) = K \times 2P \log_2(P)$$

If Increasing procs from $P_1 = 4$ to $P_2 = 8$, then Increase N by factor of

$$\frac{2P_2 \log_2(P_2)}{2P_1 \log_2(P_1)} = \frac{8 \log_2 8}{4 \log_2 4} = 24/8 = 3$$

to keep Efficiency E constant.

		Par Runtime	Speedup	Efficiency	Cost
P	N	T_p	$S = T_s/T_p$	$E = S/P$	$C = T_p \times P$
4	64	20	3.20	0.80	80
8	192	30	6.40	0.80	240
16	512	40	12.80	0.80	640

Exercise: Scalability of Cannon's Algorithm

- Recently studied Cannon's Algorithm for Dense Matrix Multiplication
- Distributed block matrix decomposition
- Parallel Runtime was a combination of communication + computation

$$T_{par} = N^3/P + 2(\sqrt{P} - 1) \times (t_s + t_w(N^2/P))$$

Questions

- What is T_{ser}, the serial runtime for matrix-matrix multiply?
- What is the parallel overhead T_{over} for Cannon's alg?
- Is it easy to analyze the Isoefficiency equation $W = KT_{over}$ for Cannon's Algorithm

Answers: Scalability of Cannon's Algorithm

- Cannon's Runtime

$$T_{par} = N^3/P + 2(\sqrt{P} - 1) \times (t_s + t_w(N^2/P))$$

- $T_{ser} = N^3$ for matrix multiply which is close enough for our purposes
- Parallel Overhead

$$\begin{aligned}
T_{over} &= PT_{par} - T_{ser} \\
&= P \times (N^3/P + 2(\sqrt{P} - 1) \times (t_s + t_w(N^2/P))) - N^3 \\
&= 2P(\sqrt{P} - 1) \times (t_s + t_w(N^2/P)) \\
&= 2(\sqrt{P} - 1) \times (Pt_s + t_w N^2) \\
&= 2(P^{3/2}t_s + P^{1/2}t_w N^2 - PT_s - t_w N^2) \\
&= O(P^{3/2}t_s + P^{1/2}N^2 t_w)
\end{aligned}$$

Isoefficiency and Complex Parallel Overhead

Cannon's alg left us with a weird parallel overhead involving both P the number of processors and N the problem size.

$$T_{over} = O(P^{3/2}t_s + P^{1/2}N^2t_w)$$

Grama section 5.4.2 suggests a means to simplify this in isoefficiency analysis

- Analyze each term in the sum against the problem work W
- Solve for W in terms of P to get balance
- Focus attention on the rate of growth in terms of P

Cannon's Isoefficiency

- Work $W = N^3$ for matrix mult
- First term is easy:

$$W = N^3 = KP^{3/2}t_s$$

- Second term is more intricate

$$N^3 = KP^{1/2}N^2t_w$$
$$N = KP^{1/2}t_w$$
$$N^3 = K^3P^{3/2}t_w^3$$

- Textbook suggests ignoring the K^3 and dropping the t_w/t_s terms to focus on contributions of P
- Both terms suggest to maintain constant efficiency work W must grow at the rate of $O(P^{3/2})$ to maintain efficiency

Final Notes

- Speedup is the metric that is most reported but does not tell the full story of parallel system scalability
- Isoefficiency is more descriptive of scalability but also not widely accepted / reported
- Alternative scalability metrics exist including Scaled Speedup, discussed in Grama 5.7