

# **Lecture 02—Amdahl's Law, Modern Hardware**

## **ECE 459: Programming for Performance**

Patrick Lam

University of Waterloo

January 7, 2015

# About Prediction and Speedups

Cliff Click said: “5% miss rates dominate performance.”

Why is that?

# About Prediction and Speedups

Cliff Click said: “5% miss rates dominate performance.”

Why is that?

Recall: 100-1000 slot penalty for a miss.

# Forcing Branch Mispredicts

blog.man7.org/2012/10/  
how-much-do-builtinexpect-likely-and.html

```
#include <stdlib.h>
#include <stdio.h>

static __attribute__((noinline)) int f(int a) { return a; }

#define BSIZE 1000000
int main(int argc, char* argv[])
{
    int *p = calloc(BSIZE, sizeof(int));
    int j, k, m1 = 0, m2 = 0;
    for (j = 0; j < 1000; j++) {
        for (k = 0; k < BSIZE; k++) {
            if (__builtin_expect(p[k], EXPECT_RESULT)) {
                m1 = f(++m1);
            } else {
                m2 = f(++m2);
            }
        }
    }

    printf("%d, %d\n", m1, m2);
}
```

Running times: 3.1s with good (or no) hint, 4.9s with bogus hint.

# Limitations of Speedups

Our main focus is parallelization.

- Most programs have a sequential part and a parallel part; and,
- Amdahl's Law answers, “what are the limits to parallelization?”

## Formulation (1)

$S$ : fraction of serial runtime in a serial execution.

$P$ : fraction of parallel runtime in a serial execution.

Therefore,  $S + P = 1$ .

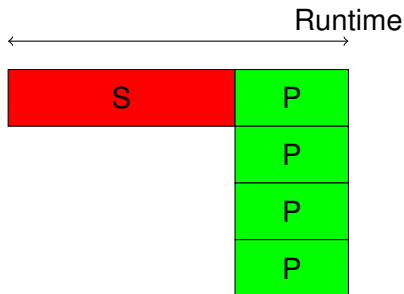
With 4 processors, best case, what can happen to the following runtime?



# Formulation (1)



We want to split up the parallel part over 4 processors



## Formulation (2)

$T_s$ : time for the program to run in serial

$N$ : number of processors/parallel executions

$T_p$ : time for the program to run in parallel

- Under perfect conditions, get  $N$  speedup for  $P$

$$T_p = T_s \cdot \left( S + \frac{P}{N} \right)$$



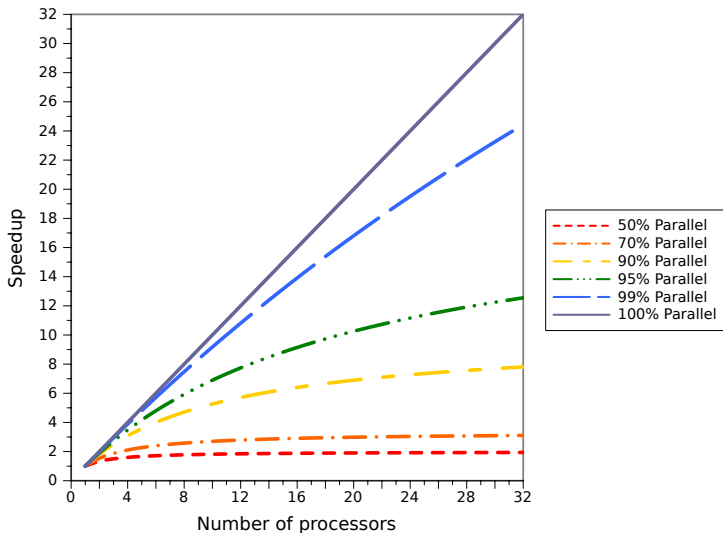
## Formulation (3)

How much faster can we make the program?

$$\begin{aligned} speedup &= \frac{T_s}{T_p} \\ &= \frac{T_s}{T_s \cdot (S + \frac{P}{N})} \\ &= \frac{1}{S + \frac{P}{N}} \end{aligned}$$

(assuming no overhead for parallelizing; or costs near zero)

# Fixed-Size Problem Scaling, Varying Fraction of Parallel Code



# Amdahl's Law

Replace  $S$  with  $(1 - P)$ :

$$\textit{speedup} = \frac{1}{(1-P) + \frac{P}{N}}$$

$$\textit{maximum speedup} = \frac{1}{(1-P)}, \text{ since } \frac{P}{N} \rightarrow 0$$

As you might imagine, the asymptotes in the previous graph are bounded by the maximum speedup.

# Assumptions behind Amdahl's Law

How can we invalidate Amdahl's Law?

# Assumptions behind Amdahl's Law

We assume:

- problem size is fixed (we'll see this soon);
- program/algorithm behaves the same on 1 processor and on  $N$  processors; and
- that we can accurately measure runtimes—  
i.e. that overheads don't matter.

# Amdahl's Law Generalization

The program may have many parts, each of which we can tune to a different degree.

Let's generalize Amdahl's Law.

$f_1, f_2, \dots, f_n$ : fraction of time in part  $n$

$S_{f_1}, S_{f_2}, \dots, S_{f_n}$ : speedup for part  $n$

$$\text{speedup} = \frac{1}{\frac{f_1}{S_{f_1}} + \frac{f_2}{S_{f_2}} + \dots + \frac{f_n}{S_{f_n}}}$$

## Application (1)

Consider a program with 4 parts in the following scenario:

Part	Fraction of Runtime	Speedup	
		Option 1	Option 2
1	0.55	1	2
2	0.25	5	1
3	0.15	3	1
4	0.05	10	1

We can implement either Option 1 or Option 2.  
Which option is better?

## Application (2)

“Plug and chug” the numbers:

### Option 1

$$\text{speedup} = \frac{1}{0.55 + \frac{0.25}{5} + \frac{0.15}{3} + \frac{0.05}{5}} = 1.53$$

### Option 2

$$\text{speedup} = \frac{1}{\frac{0.55}{2} + 0.45} = 1.38$$



# Empirically estimating parallel speedup P

Useful to know, don't have to commit to memory:

$$P_{\text{estimated}} = \frac{\frac{1}{\text{speedup}} - 1}{\frac{1}{N} - 1}$$

- Quick way to guess the fraction of parallel code
- Use  $P_{\text{estimated}}$  to predict speedup for a different number of processors

# Summary of Amdahl's Law

Important to focus on the part of the program with most impact.

Amdahl's Law:

- estimates perfect performance gains from parallelization; but,
- only applies to solving a **fixed problem size** in the shortest possible period of time

# Gustafson's Law: Formulation

$n$ : problem size

$S(n)$ : fraction of serial runtime for a parallel execution

$P(n)$ : fraction of parallel runtime for a parallel execution

$$T_p = S(n) + P(n) = 1$$

$$T_s = S(n) + N \cdot P(n)$$

$$speedup = \frac{T_s}{T_p}$$

# Gustafson's Law

$$\text{speedup} = S(n) + N \cdot P(n)$$

Assuming the fraction of runtime in serial part decreases as  $n$  increases, the speedup approaches  $N$ .

- Yes! Large problems can be efficiently parallelized.  
(Ask Google.)

# Driving Metaphor

## **Amdahl's Law**

Suppose you're travelling between 2 cities 90 km apart. If you travel for an hour at a constant speed less than 90 km/h, your average will never equal 90 km/h, even if you energize after that hour.

## **Gustafson's Law**

Suppose you've been travelling at a constant speed less than 90 km/h. Given enough distance, you can bring your average up to 90 km/h.