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.

See `L02.pdf` for a calculation.

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?"

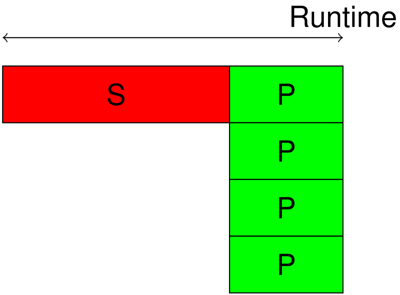
Formulation (1)

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

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

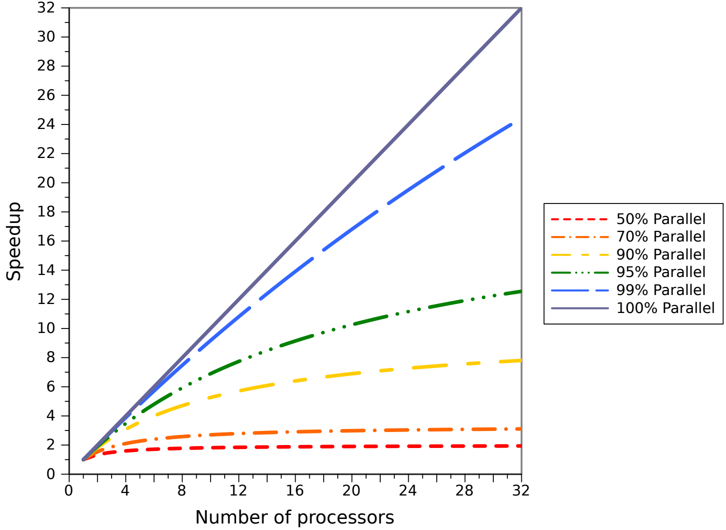
Formulation (3)

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 $(1 - P)$:

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

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

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.