Outline

* Performance Metrics

Measuring Performance

- What do we mean by the performance of a processor ?
  - **ANSWER** : Almost nothing
- What should we ask instead ?
  - What is the performance with respect to a given program or a set of programs ?
  - Performance is inversely proportional to the time it takes to execute a **program**

Computing the Time a Program Takes

$$\tau = \#seconds$$
$$= \frac{\#seconds}{\#cycles} * \frac{\#cycles}{\#instructions} * (\#instructions)$$
$$= \underbrace{\frac{\#seconds}{\#cycles}}_{1/f} + \underbrace{\frac{\#cycles}{\#instructions}}_{CPI} * (\#instructions)$$
$$= \frac{CPI * \#insts}{f}$$

* CPI → Cycles per instruction
* f → frequency (cycles per second)

The Performance Equation

$$P \propto \frac{IPC * f}{\#insts}$$

* IPC → 1/CPI (Instructions per Cycle)
* What are the units of performance ?
  * **ANSWER** : arbitrary

Number of Instructions (#insts)

Static Instruction: The binary or executable of a program, contains a list of ***static instructions***.

Dynamic Instruction: A **dynamic instruction** is a running instance of a static instruction, which is created by the processor when an instruction enters the pipeline.

- Note that these are dynamic instructions
  - **NOT** static instructions
- A smart compiler can reduce the number of executed instructions

Number of Instructions(#insts) – 2

- Dead code removal
  - Often programmers write code that does not determine the final output
  - This code is redundant
  - It can be identified and removed by the **compiler**
- Function inlining
  - Very small functions have a lot of overhead → call, ret instructions, register spilling, and restoring
  - Paste the code of the callee in the code of the caller (known as **inlining**)

Computing the CPI

- CPI for a single cycle processor = 1
- CPI for an ideal pipeline(no hazards)
  - Assume we have *n* instructions, and *k* stages
  - The first instruction enters the pipeline in cycle 1
  - It leaves the pipeline in cycle k
  - The rest of the **(n-1) instructions** leave in the next (n-1) consecutive cycles

$$CPI = \frac{n + k - 1}{n}$$

Computing the Maximum Frequency

- Let the maximum amount of time that it takes to execute any instruction be :
  - $\mathbf{t_{max}}$ (also known as algorithmic work)
- Minimum clock cycle time of a single cycle pipeline → t_{max}
- In the case of a pipeline, let us assume that all the pipeline stages are **balanced**
- Time per stage → t_{max} / k

Maximum Frequency - II

* Let the **latch delay** be **l**
* We thus have :

$$t_{stage} = \frac{t_{max}}{k} + l$$

$$\frac{1}{f} = \frac{t_{max}}{k} + l$$

The minimum cycle time (1/f) is equal to t_{stage}. Let us thus, assume that our cycle time is as low as possible.

Performance of an Ideal Pipeline

* Let us assume that the number of instructions are a **constant**

$$P = \frac{f}{CPI}$$

$$= \frac{\frac{1}{\frac{t_{max}}{k} + l}}{\frac{n + k - 1}{n}}$$

$$= \frac{n}{\left(\frac{t_{max}}{k} + l\right) * (n + k - 1)}$$

$$= \frac{n}{\frac{(n-1)t_{max}}{k} + (t_{max} + \ln - l) + lk}$$

Optimal Number of Pipeline Stages

$$\frac{\partial\left(\frac{(n-1)t_{max}}{k}+(t_{max}+\ln-l)+lk\right)}{\partial k}=0$$

$$\Rightarrow -\frac{(n-1)t_{max}}{k^2}+l=0$$

$$\Rightarrow k=\sqrt{\frac{(n-1)t_{max}}{l}}$$

* k is inversely proportional to $\sqrt{l}$
* k is proportional to $\sqrt{t_{max}}$

Implications

- As we increase the latch delay, we should have less pipeline stages
  - We need to minimise the time wasted in accessing latches
- As we increase the amount of **algorithmic work**, we require more pipeline stages for ideal performance
  - More pipeline stages help distribute the work better, and increase the overlap across instructions

Implications - II

* As the number of instructions tends to ∞, the number of ideal pipeline stages also tends to ∞
  * The higher startup time gets amortized in the long run

A Non-Ideal Pipeline

* Our ideal CPI (CPI_{ideal} = 1) is 1
* However, in reality, we have stalls

$$CPI = CPI_{ideal} + stall\_rate * stall\_penalty$$

* Let us assume that the stall rate is a function of the program, and its nature of dependences

Non-Ideal Pipeline - II

- Let us assume that the stall penalty is proportional to the number of pipeline stages
- Both these assumptions are strictly not correct. They are being used to make a coarse grained mathematical model.
- CPI = (n+k-1)/n + rck
  - r → stall rate, c → constant of proportionality

Mathematical Model

$$P = \frac{f}{CPI}$$

$$= \frac{\frac{1}{\frac{t_{max}}{k} + l}}{\frac{n + k - 1}{n} + rck}$$

$$= \frac{n}{\frac{(n-1)t_{max}}{k} + (rcnt_{max} + t_{max} + \ln - l) + lk(1 + rcn)}$$

Mathematical Model - II

$$\frac{\partial\left(\frac{(n-1)t_{max}}{k}+(rcnt_{max}+\ t_{max}+\ln-l)+lk(1+rcn)\right)}{\partial k}=0$$

$$\Longrightarrow -\frac{(n-1)t_{max}}{k^2}+l(1+rcn)=0$$

$$\Longrightarrow k=\sqrt{\frac{(n-1)t_{max}}{l(1+rcn)}}\approx\sqrt{\frac{t_{max}}{lrc}}\qquad(as\ n\ \longrightarrow\ \infty)$$

Implications

- For programs with a lot of dependences (high value of r) → Use less pipeline stages
- For a pipeline with forwarding → c is smaller (than a **pipeline** that just has interlocks)
  - It requires a larger number of pipeline stages for optimal performance

Implications

- The optimal number of pipeline stages is directly proportional to $\sqrt{(t_{max} / l)}$
  - This ratio is not significantly changing across technologies.
  - This explains why the number of pipline stages has remained more or less constant for the last 5-10 years

Example

Example Consider two programs that have the following characteristics.

Program 1		Program 2	
Instruction Type	Fraction	Instruction Type	Fraction
loads	0.4	loads	0.3
Branches	0.2	Branches	0.1
ratio(taken branches)	0.5	ratio(taken branches)	0.4

The ideal CPI is 1 for both the programs. Let 50% of the load instructions suffer from a load use hazard. Assume that the frequency of P_1 is 1, and the frequency of P_2 is 1.5. Here, the units of the frequency are not relevant. Compare the performance of P_1 and P_2.

Example

Answer:

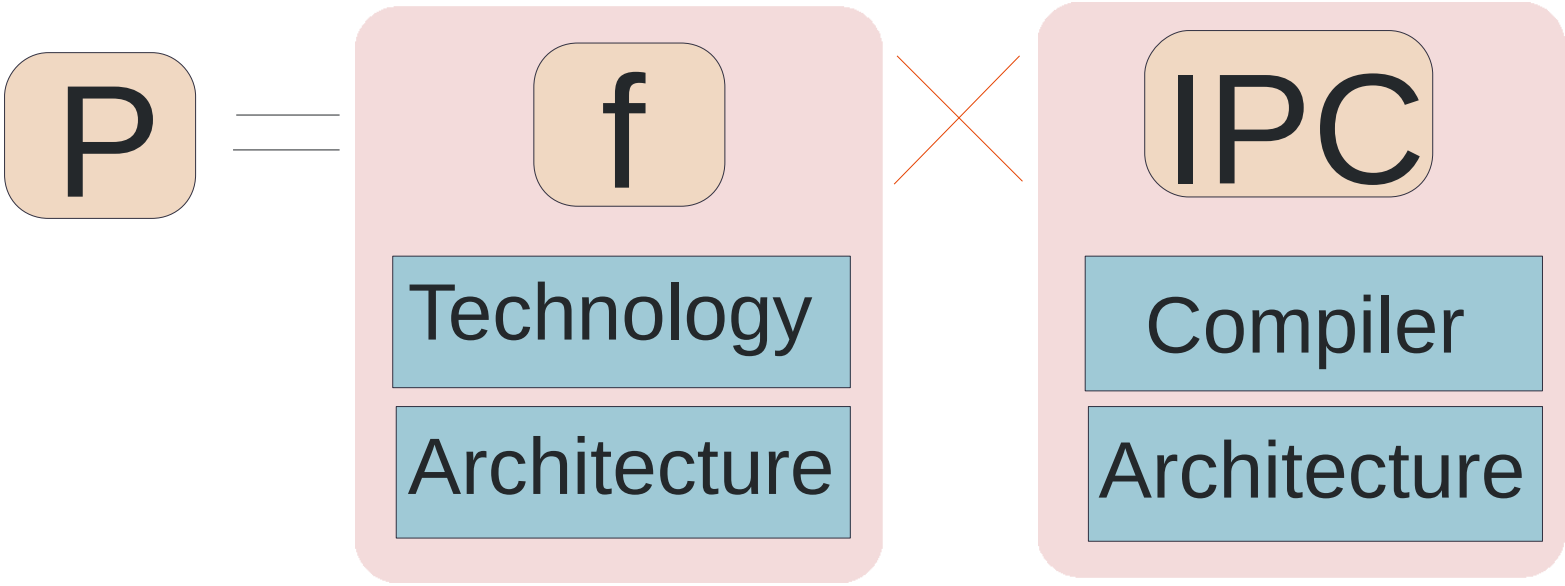
$$CPI_{new} = CPI_{ideal} + 0.5 \times (ratio(loads)) \times 1 + ratio(branches) \times ratio(taken\,branches) \times 2 \quad (9.13)$$

We thus have:

$$CPI_{P_1} = 1 + 0.5 \times 0.4 + 0.2 \times 0.5 \times 2 = 1 + 0.2 + 0.2 = 1.4$$
$$CPI_{P_2} = 1 + 0.5 \times 0.3 + 0.1 \times 0.4 \times 2 = 1 + 0.15 + 0.08 = 1.23$$

The performance of P_1 can be expressed as $f/CPI = 1 / 1.4 = 0.71$ (arbitrary units). Similarly, the performance of P_2 is equal to $f/CPI = 1.5/1.23 = 1.22$ (arbitrary units). Hence, P_2 is faster than P_1. We shall often use the term, arbitrary units, a.u., when the choice of units is irrelevant.

Performance, Architecture, Compiler



THE END