Analysis and modeling of

Computational Performance

Review of computer architectures 2: Latency and throughput

Performance equation

CPU time = number of seconds / program =
number of seconds / cycle *
number of cycles / instruction *
number of instructions / program

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Problems:
  - number of instructions – different compilers produce different instructions for the same source code (i.e. the same number of effectively performed source code operations)
  - clock cycles – different possible frequencies depending on conditions, e.g. energy saving modes of operation
  - CPI – number of cycles per instruction
    - how to determine CPI in order to apply the performance equation?

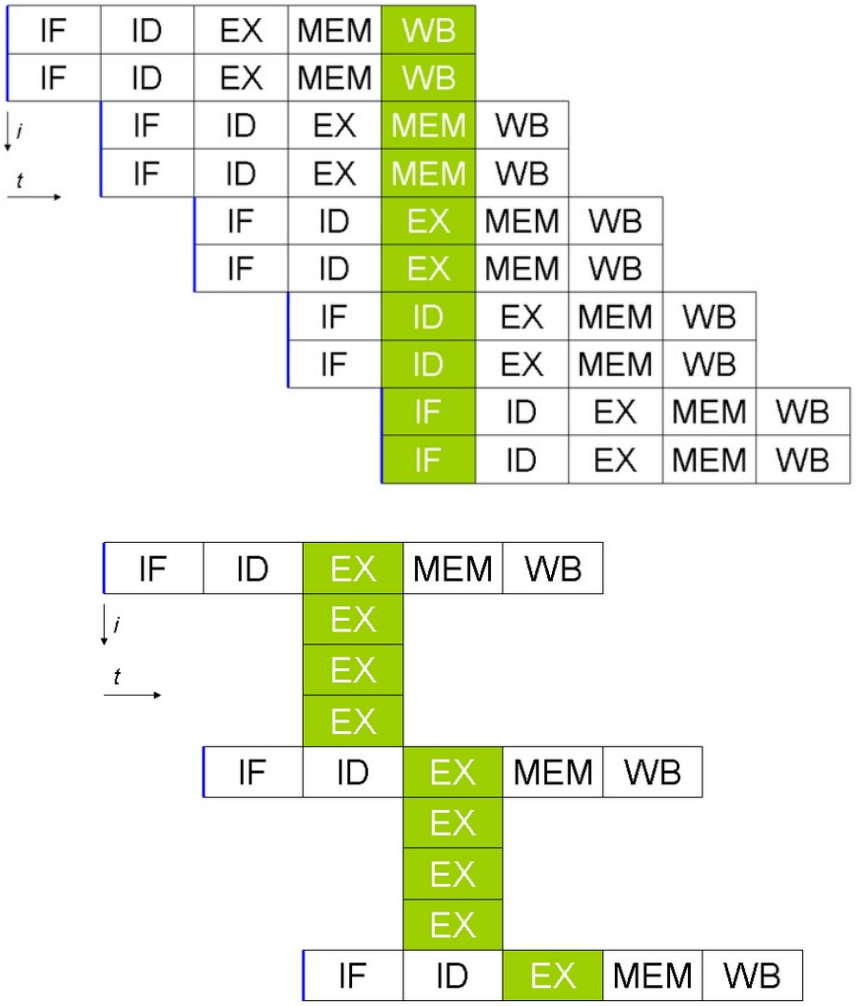
CPI – cycles per instruction measure

- Sequential processing
  - 1 cycle per phase, k phases
  - k cycles per instruction
  - CPI=k
- Pipelining
  - CPI=k cannot be used to calculate CPU time
  - for long instruction sequences better value is the average number of cycles between subsequent retired instructions
    - CPI = 1 – for the example

- Normally applied – the average CPI
  - CPI = total number of cycles / total number of instructions
  - easy to measure, difficult to obtain theoretically

Contemporary processors/cores

- Further enhancements of instruction processing
  - superscalar processors/cores
    - multiple functional units
    - multiple pipelines
    - CPI<1
  - vector capabilities
    - vector registers
    - SIMD instructions
    - CPI<1
  - cache memories
    - including caches for virtual memory page tables



+ deeper pipelines, branch prediction, out of order execution, prefetchnig, speculative execution, hardware multithreading etc.

Performance related characteristics

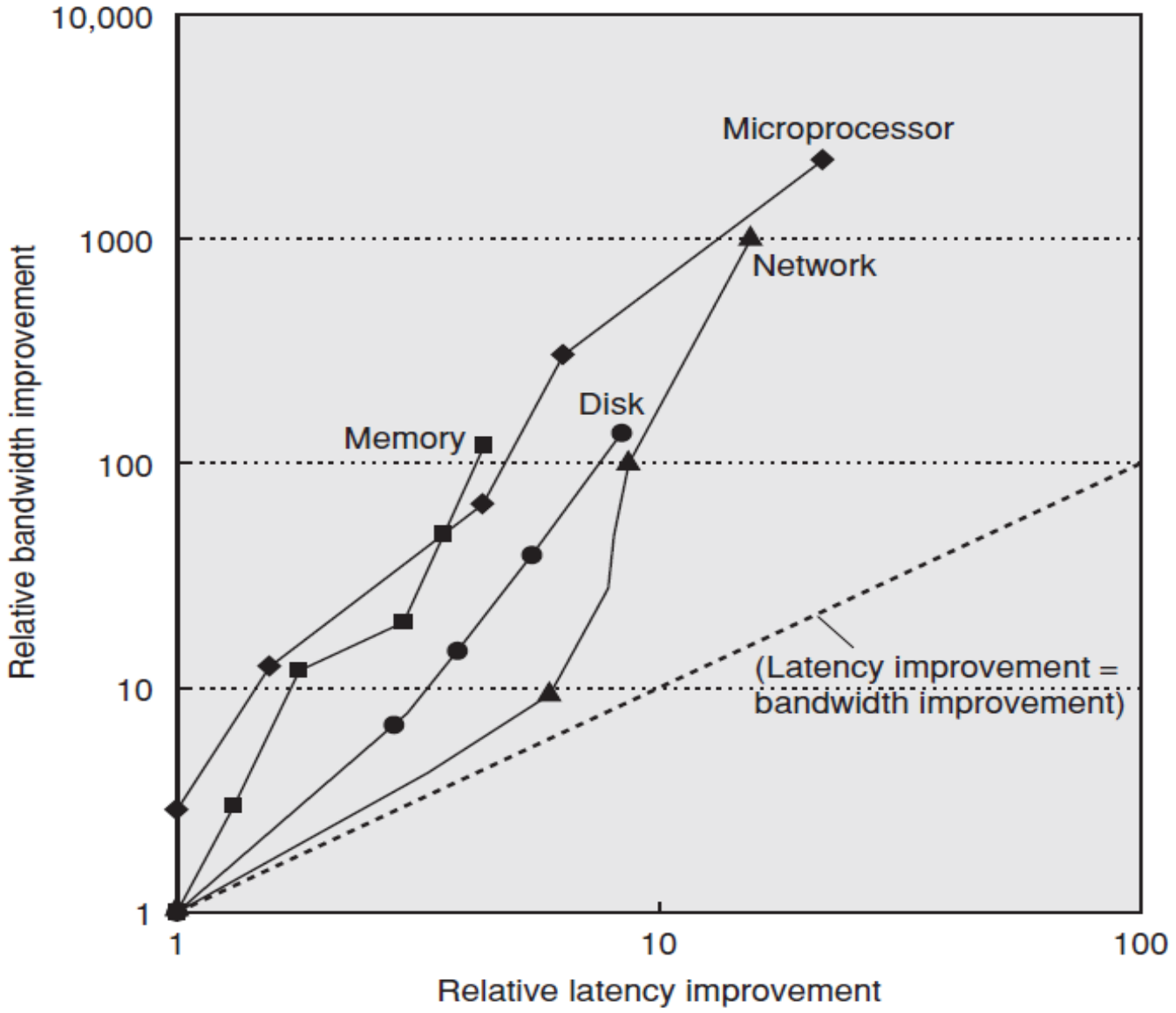
- Latency
  - general:
    - time between the stimulation and the response, between the cause and the effect, between the beginning of operation and its end
      - in some situations equivalent to "response time"
  - specific for instruction processing:
    - time of pipeline processing (excluding fetching, decoding, etc.)
    - experimental: number of cycles after retiring an instruction till retiring the next instruction, related to the previous one by data dependence
      - for long sequences of such instructions, the performance (the average CPI) will be related to instruction latency

```
for(i=0;i<1000000;i++)
   a = 1.000001*a;
}
```

Performance related characteristics

- Throughput
  - often related to the maximal capacity of the performing hardware
    - "bandwidth"
    - for instruction processing:
      - the maximal number of retired instructions per time unit or clock cycle
      - may be different for different kinds of instructions
        - » depends on the number of specific pipelines
  - the related measure – IPC
    - the number of instructions per cycle
      - maximal – theoretical peak performance
      - as the average quantity:
        - » IPC = (number of instructions) / (number of clock cycles)
        - » IPC = 1 / CPI
        - » characterizes the actual performance of execution

Latency and throughput



Performance measures

- MIPS
  - number of instructions per second (in millions)
  - MIPS = IPC * frequency [MHz]
  - historical meaning only
    - does not characterize the performance for real workloads with different combinations of various instructions
- GFLOP/s
  - number of floating point instructions (FLOPs) per second (in billions - GFLOP/s)
  - relevant for "number crunching"
    - applications where floating point operations dominate
  - theoretical peak performance:
    - GFLOP/s = IPC (for floating point operations) * frequency [GHz]
  - actual performance
    - GFLOP/s = the number of FLOPs performed / execution time

Little's law

- Little's law (from queuing theory)
  - the average number L of customers in a stationary system is equal to the average effective arrival rate λ multiplied by the average time W that a customer spends in the system: $L = \lambda W$
    - the arrival rate is assumed to be equal to departure rate
  - for superscalar superpipelined instruction processing:
    - L – the number of instructions processed concurrently by a core
    - λ – the number of instructions retired per cycle
      - increases with the number of pipelines
    - W – the number of cycles to process each of the instructions
  - **in order to maximize the throughput**, $IPC = \lambda = L / W$, i.e. to keep it as close as possible to the theoretical maximum, given the long pipelines W, **maximize the number of concurrent instructions** L
    - sufficient number of independent instructions in the code
    - efficient fetching and decoding
    - no stalls in the pipelines, no hazards, no data dependencies