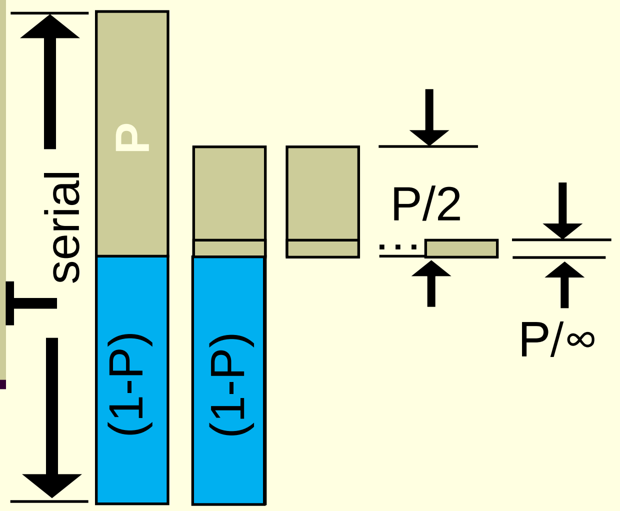
Amdahl’s Law

- **Amdahl's Law:**
  - Is a law governing the *speedup* of using parallel processors on an application, versus using only one serial processor.
  - Describes the upper bound of parallel execution speedup.

n = 2∞



0.5 + 0.05

$$T_{parallel} = \{(1\text{-}P) + P/n\}\ T_{serial}$$

n = number of processors

Speedup = $T_{serial} / T_{parallel}$

1.0/0.5 = 2.00

Parallel Efficiency

Parallel Efficiency:

- Is a measure of how efficiently processor resources are used during parallel computations.
- Is equal to (*Speedup / Number of Threads) * 100%.*

$$\begin{tikzcd}
& \text{CONCURENCY != PARALLELISM} \arrow[dl] \arrow[dr] & \\
\text{Composition Dealing} & & \text{Execution Doing}
\end{tikzcd}$$

Granularity

Definition:

- An approximation of the ratio of *computation* to *synchronization*.

The two types of granularity are:

- **Coarse-grained:** Concurrent calculations that have a large amount of computation between synchronization operations are known as *coarse-grained*.
- **Fine-grained:** Cases where there is very little computation between synchronization events are known as *fine-grained*.

Summary

- A *thread* is a discrete sequence of related instructions that is executed independently. It is a single sequential flow of control within a program.
- The *benefits* of using threads are increased performance, better resource utilization, and efficient data sharing.
- The *risks* of using threads are data races, deadlocks, code complexity, portability issues, and testing and debugging difficulty.
- Every process has at least one thread, which is the main thread that initializes the process and begins executing the initial instructions.
- All threads within a process share code and data segments.
- Concurrent threads can execute on a single processor. Parallelism requires multiple processors.

Summary (Continued)

- *Turnaround* refers to completing a single a task in the smallest amount of time possible, whereas accomplishing the most tasks in a fixed amount of time refers to *throughput*.
- Decomposing a program based on the number and type of functions that it performs is called *functional decomposition*.
- Dividing large data sets whose elements can be computed independently, and associating the required computation among threads is known as *data decomposition* in multithreaded applications.
- Applications that scale with the number of independent functions are probably best suited to functional decomposition while applications that scale with the amount of independent data are probably best suited to data decomposition.
- *Race conditions* occur because the programmer assumes a particular order of execution but does not use synchronization to guarantee that order.

Summary (Continued)

- Storage conflicts can occur when multiple threads attempt to simultaneously update the same memory location or variable.
- *Critical regions* are parts of threaded code that access (read or write) shared data resources. To ensure data integrity when multiple threads attempt to access shared resources, critical regions must be protected so that only one thread executes within them at a time.
- *Mutual exclusion* refers to the program logic used to ensure single-thread access to a critical region.
- *Barrier synchronization* is used when all threads must have completed a portion of the code before proceeding to the next section of code.
- *Deadlock* refers to a situation when a thread waits for an event that never occur. This is usually the result of two threads requiring access to resources held by the other thread.

Summary (Continued)

- *Livelock* refers to a situation when threads are not making progress on assigned computations, but are not idle waiting for an event.
- *Speedup* is the metric that characterizes how much faster the parallel computation executes relative to the best serial code.
- *Parallel Efficiency* is a measure of how busy the threads are during parallel computations.
- *Granularity* is defined as the ratio of computation to synchronization.
- *Load balancing* refers to the distribution of work across multiple threads so that they all perform roughly the same amount of work.

Osnovni načini CPU podrške za MM

- Načini podrške za MM
  - Standard CPU
  - Support for SIMD
  - Multi thread
  - Multi core
  - Media coprocessor
  - Media procesor
  - ASIC

Standardni CPU

- Ako se prisjetimo Arhitekture računala onda znamo da obični procesori mogu izvesti jednu ALU operaciju po periodu
- Operacija je širine koju ima ALU

- Kako bi ubrzali izvođenje moramo mnogo pažnje posvetiti dohvatu podataka te optimizacijama petlji

Standardni CPU

- Za DSP operacije (npr DCT) izuzetno je korisno ako procesor ima sklopovski izvedeno množenje i pripadnu naredbu

- Dodatna značajna prednost ako postoji naredba množenja sa zbrajanjem (Multiply Accumulate, MLA)
  - Kod primjera računanja DCT, DFT i slično imamo većinu “leptir” operacija kod kojih je MLA osnovna karika

Standardni CPU ubrzanja

- Efikasno korištenje protočne strukture
  - Loop-unrolling
  - Branch prediction
- ....
- Nedovoljno !

SIMD

- Analizom mnogih aplikacija ustanovljeno:
  - Koje aplkacije su najzahtjevnije
  - Gdje je efikasnost najmanja
  - Koja su uska grla
  - ....
- Povećanje brzine nije rješenje
  - P4 fijasko
    - (enormni pipeline: energija, toplina, branch miss)
  - ARH1 !!

SIMD

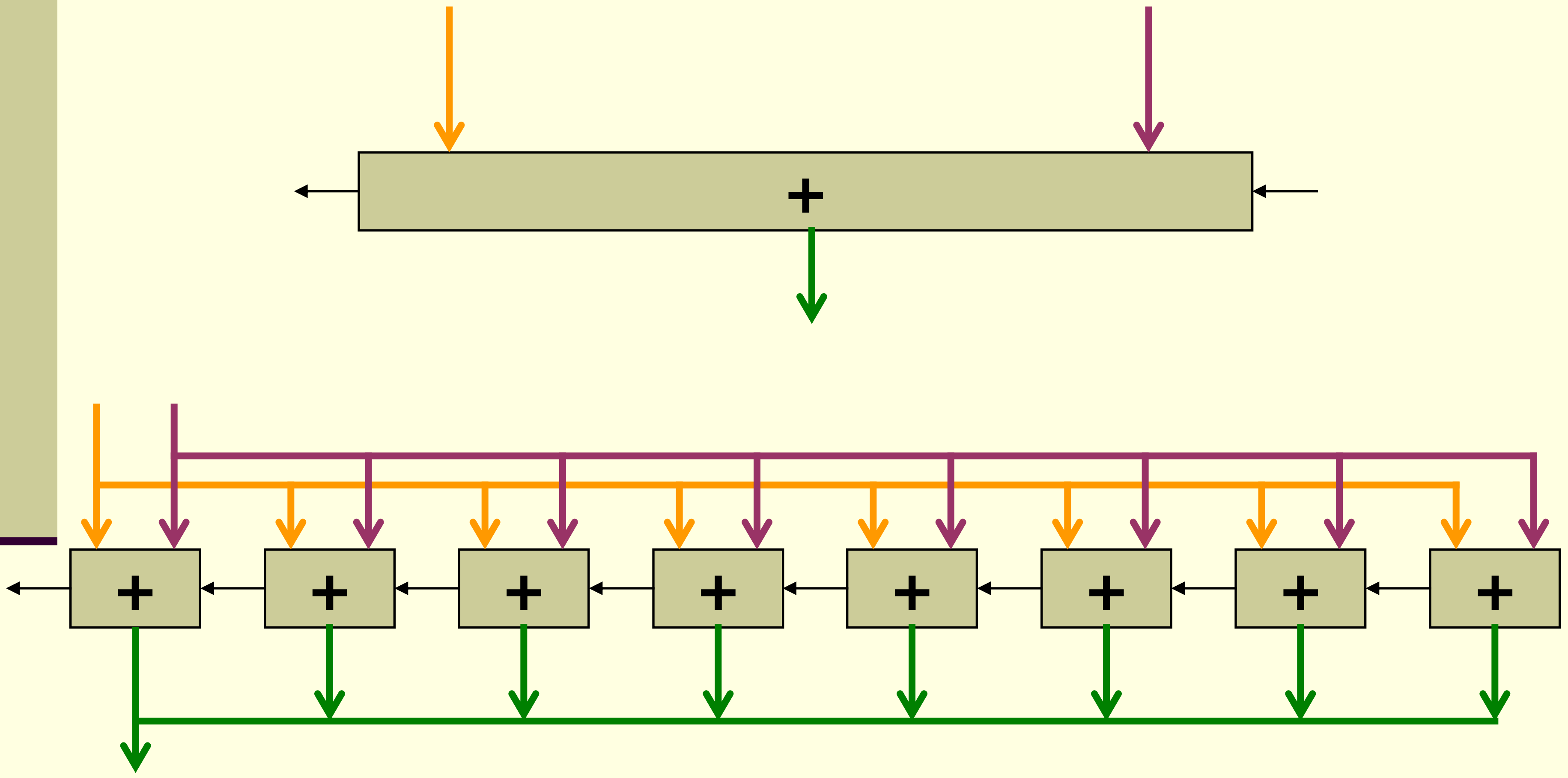
- Ispada da su najzahtjevnije aplikacije koje koriste vrlo jednostavne operacije i operande
  - 8 bita (video)
  - 16/32 bita audio
- ... Ali puuuuuno podataka

Podrška za SIMD

- SIMD (Single Instruction Multiple Data)
  - Arhitektura puta podataka u procesoru koja omogućuje obradu više podataka (u načelu manje preciznosti) sa jednom naredbom

- Osnovna ideja:
  - Ako imamo npr 64 bitovnu ALU onda je potpuno neefikasno s njom obrađivati 8 bitovne podatke
  - Reorganizirati ALU na način da se može "podijeliti"

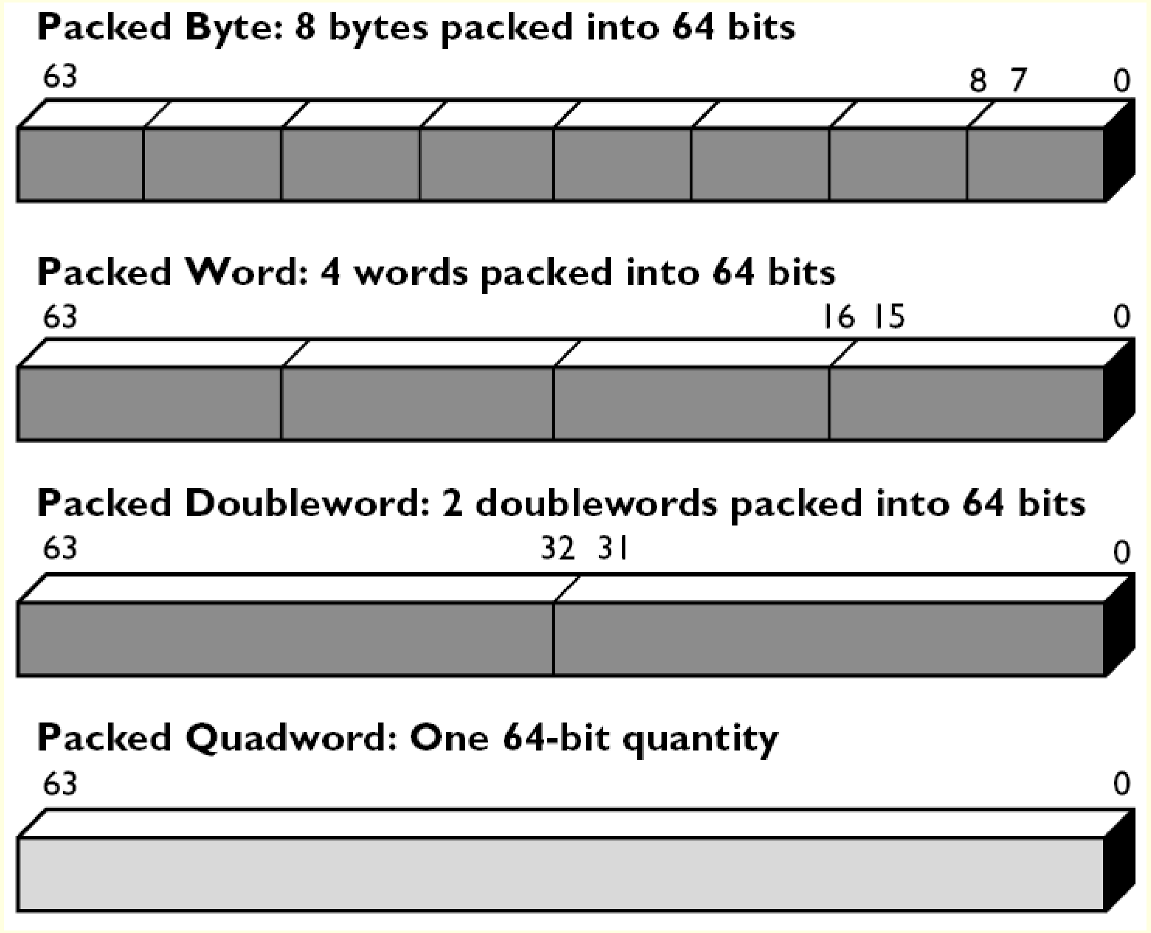
ALU – obična i SIMD



Najpoznatiji primjeri

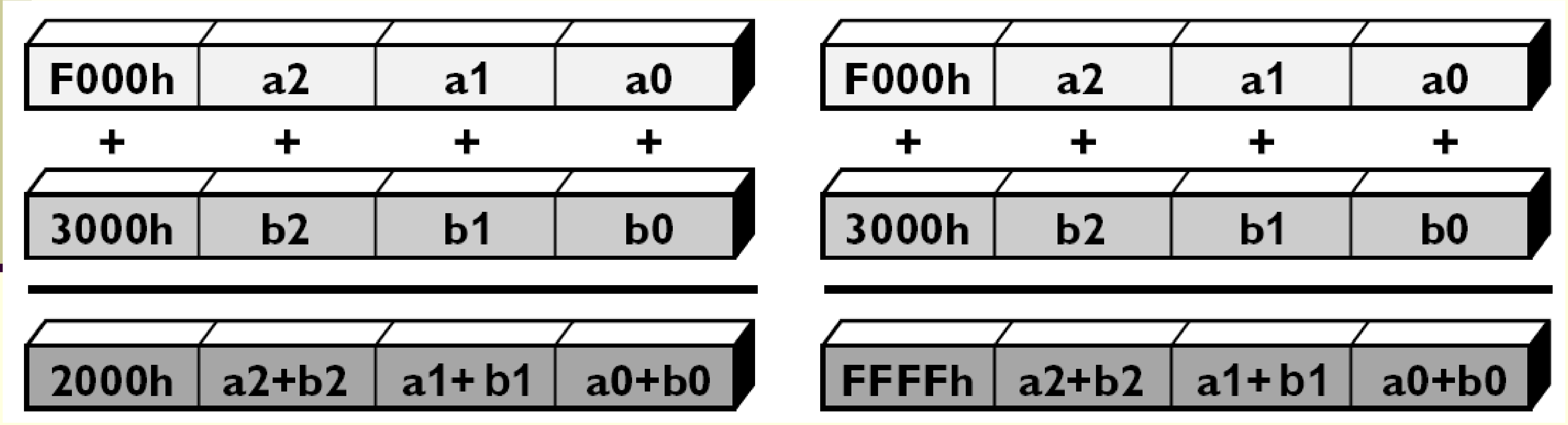
- MMX
- Stavljen na tržište 1996 (Pentium sa MMX i Pentium II)
- Uveden novi tip podataka: Packed 64 bit
- Zašto 64 bita?
  - zadovoljavajuće ubrzanje
  - Ne treba velike zahvate na arhitekturi

MMX tipovi podataka


Packed Byte: 8 bytes packed into 64 bits
63
8 7
0
Packed Word: 4 words packed into 64 bits
63
16 15
0
Packed Doubleword: 2 doublewords packed into 64 bits
63
32 31
0
Packed Quadword: One 64-bit quantity
63
0

Aritmetika sa zasićenjem

- Izuzetno korisno kod MM algoritama
- Značajno poboljšava performanse



wrap-around saturating

Rezultati...

- Inicjalno velika medijska “buka” ali rezultati nisu zadovoljili na dulje vrijeme..
- Ubrzanja su dobra u odnosu na uloženo ali su uočene potrebe za poboljšanjem
- Nedostaci:
  - Ne mogu se koristiti paralelno FPU i INT
  - Nedovoljna podrška za 8 i 32 bitovne podatke
  - Samo integer aritmetika
  - OS nije problem osvježiti

3DNow!

- AMD-ova proširenja MMX-a
  - Predstavljena 1998.
  - Podrška za 32 bitovni FP
  - Neke od ovih naredaba Intel uveo u SSE
  - Kasnije prošireno na 3DNow!+

SSE

- Streaming SIMD Extensions
- Predstavljeno 1999 sa Pentium III

- Popravljeni najvažniji nedostaci MMX-a
- 8 potpuno novih 128 bitovnih registara XMM0..XMM7 (još 8, ..XMM15 u 64-bit)
- Omogućena 128-bit (4x32bit) SIMD FP podrška: aritmetika, usporedbe, shuffle,...
- Integer SIMD podrška nad 64 bita
- Preko 60 novih naredbi

SSE nedostaci

- Spremanje stanja registara loše izvedeno
- Resursi za izvođenje zajednički sa FPU

SS.....

- SSE2
  - Pentium4(2001), 144 nove naredbe, 64 bitovni FP, proširenje MMX naredbi radi i sa XMM, performanse nisu naročito bolje od MMX
- SSE3
  - Pentium4(2004), horizontalne operacije, konverzije FP-INT jednostavnije,
- SSSE3
  - Intel Core, 16 novih naredaba, AMD ne podržava

SSE4

- 54 nove naredbe
  - SSE4.1 (47), SSE4.2 (7)
- Bolja implementacija nekih operacija

SIMD Zaključak

- SIMD proširenja donose značajna poboljšanja performansi
- Nažalost, korištenje SIMD zahtjeva ogromno znanje i puuuno vremena ali rezultira izuzetno efikasnim kodom
- Nove inačice SIMD dovode do sve boljih rezultata

- No, i SIMD ima svojih granica .... Što dalje?