Master Informatics Eng.

2019/20

A.J.Proença

Multithreading & Data Parallelism

(some slides are borrowed, mod's in green)

Processor arch: beyond Instruction-Level Parallelism

- When exploiting ILP, goal is to minimize CPI
  - Pipeline CPI (*efficient to exploit loop-level parallelism*) =>
    - Ideal pipeline CPI + ✔
    - Structural stalls + ✔
    - Data hazard stalls + ✔
    - Control stalls + ✔
    - Memory stalls ... *cache techniques ...*
  - Multiple issue =>
    - find enough parallelism to keep pipeline(s) occupied
- Multithreading =>
  - find additional ways to keep pipeline(s) occupied

Multithreading

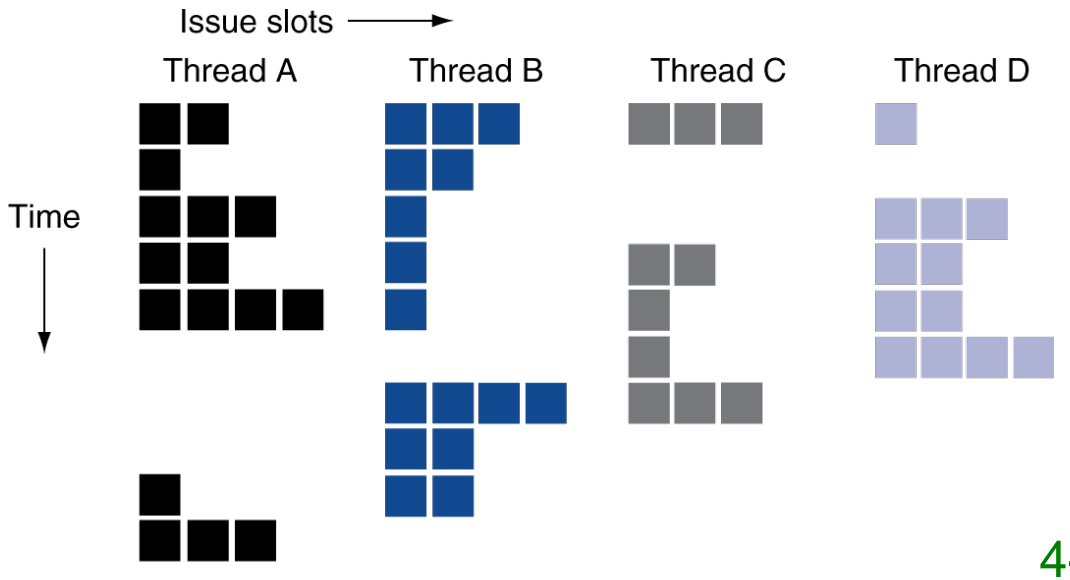
- Performing multiple threads of execution in parallel
  - Replicate registers, PC/IP, etc.
  - Fast switching between threads
- Fine-grain multithreading / time-multiplexed MT
  - Switch threads after each cycle
  - Interleave instruction execution
  - If one thread stalls, others are executed
- Coarse-grain multithreading
  - Only switch on long stall (e.g., L2-cache miss)
  - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
  - Schedule instructions from multiple threads
  - Instructions from independent threads execute when function units are available
  - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
  - Two threads: duplicated registers, shared function units and caches

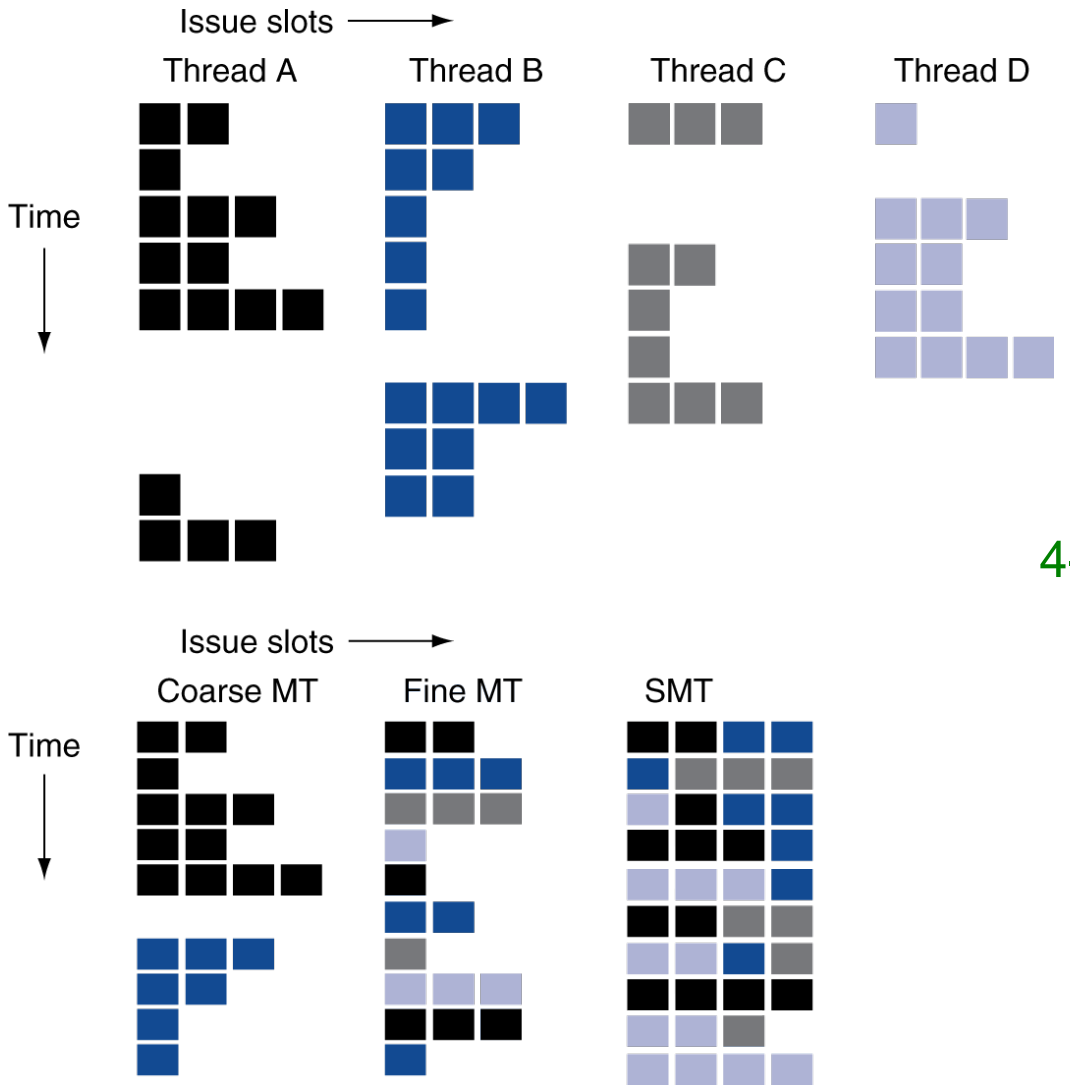
***HT**: Hyper-Threading, Intel trade mark for their SMT implementation*
*MT in Xeon Phi KNC: 4-way SMT with time-mux MT, **not HT**!*

Multithreading Example



4-way superscalar

Multithreading Example



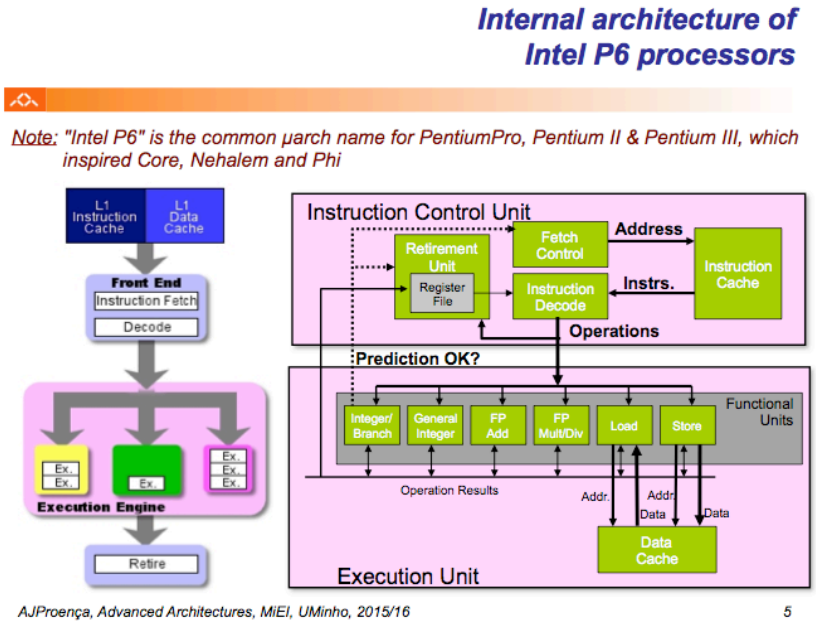
4-way superscalar

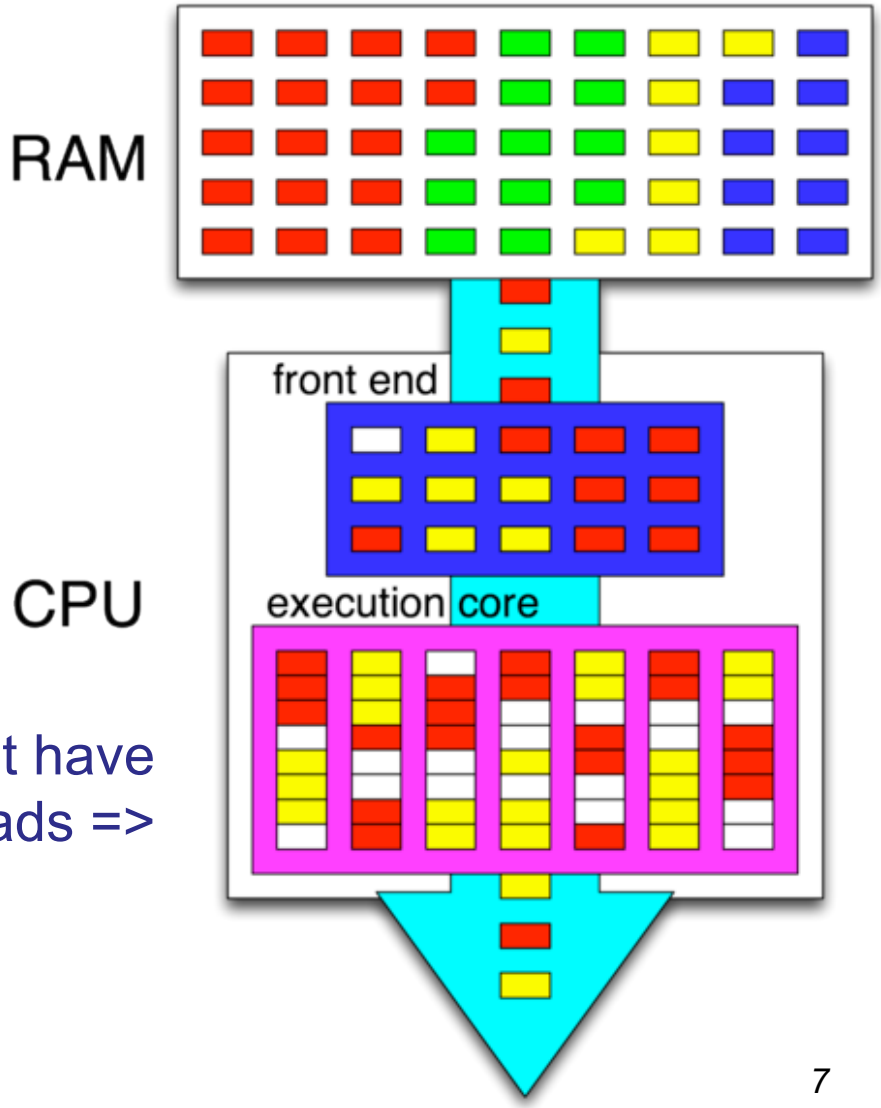
Article Talk

Hyper-threading

From Wikipedia, the free encyclopedia

As seen before...





The pipelined functional units might have better use if shared among more threads =>

Note: white boxes are bubbles...

Processor arch: beyond Instruction-Level Parallelism

- When exploiting ILP, goal is to minimize CPI
  - Pipeline CPI (*efficient to exploit loop-level parallelism*) =>
    - Ideal pipeline CPI + ✔
    - Structural stalls + ✔
    - Data hazard stalls + ✔
    - Control stalls + ✔
    - Memory stalls ... *cache techniques ...*
  - Multiple issue =>
    - find enough parallelism to keep pipeline(s) occupied
- Multithreading =>
  - find additional ways to keep pipeline(s) occupied
- Insert data parallelism features

Instruction and Data Streams

		Data Streams	
		Single	Multiple
Instruction Streams	Single	**SISD**: Intel Pentium 4	**SIMD**: SSE instructions of x86
	Multiple	**MISD**: No examples today	**MIMD**: Intel Xeon e5345

- SPMD: Single Program Multiple Data
  - A parallel program on a MIMD computer
  - Conditional code for different processors

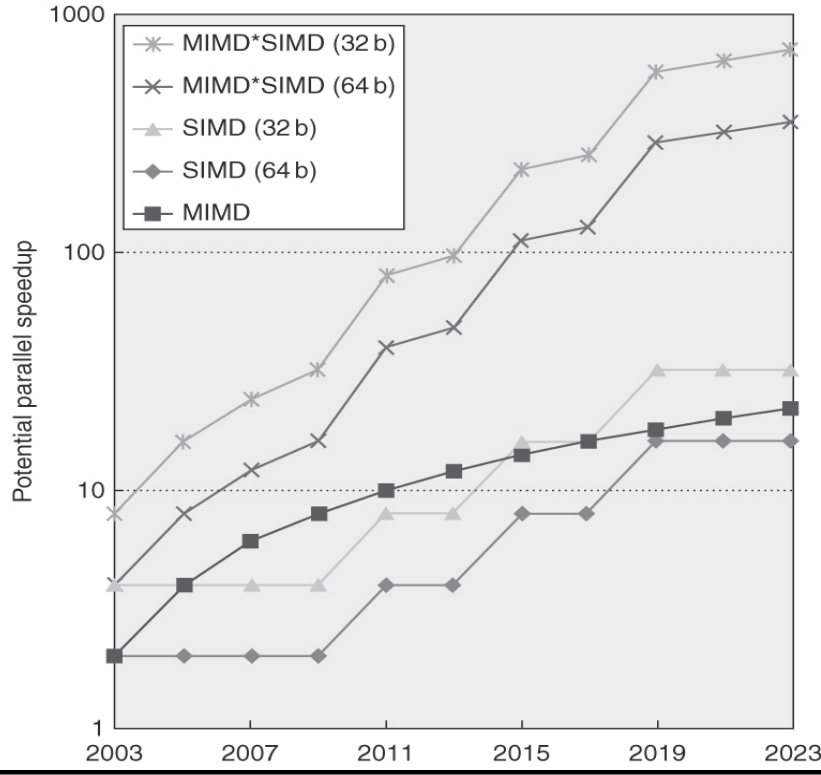
Introduction

- SIMD architectures can exploit significant data-level parallelism for:
  - matrix-oriented scientific computing
  - media-oriented image and sound processing
- SIMD is more energy efficient than MIMD
  - only needs to fetch one instruction per data operation
  - makes SIMD attractive for personal mobile devices
- SIMD allows programmers to continue to think sequentially

SIMD Parallelism

- Vector architectures *(slides 5 to 18)*
- SIMD & extensions *(slides 19 to 24)*
- Graphics Processor Units (GPUs) *(next set)*

- For x86 processors:
  - Expected grow:
    2 more cores/chip/year
  - SIMD width:
    2x every 4 years
  - Potential speedup:
    SIMD 2x that from MIMD!

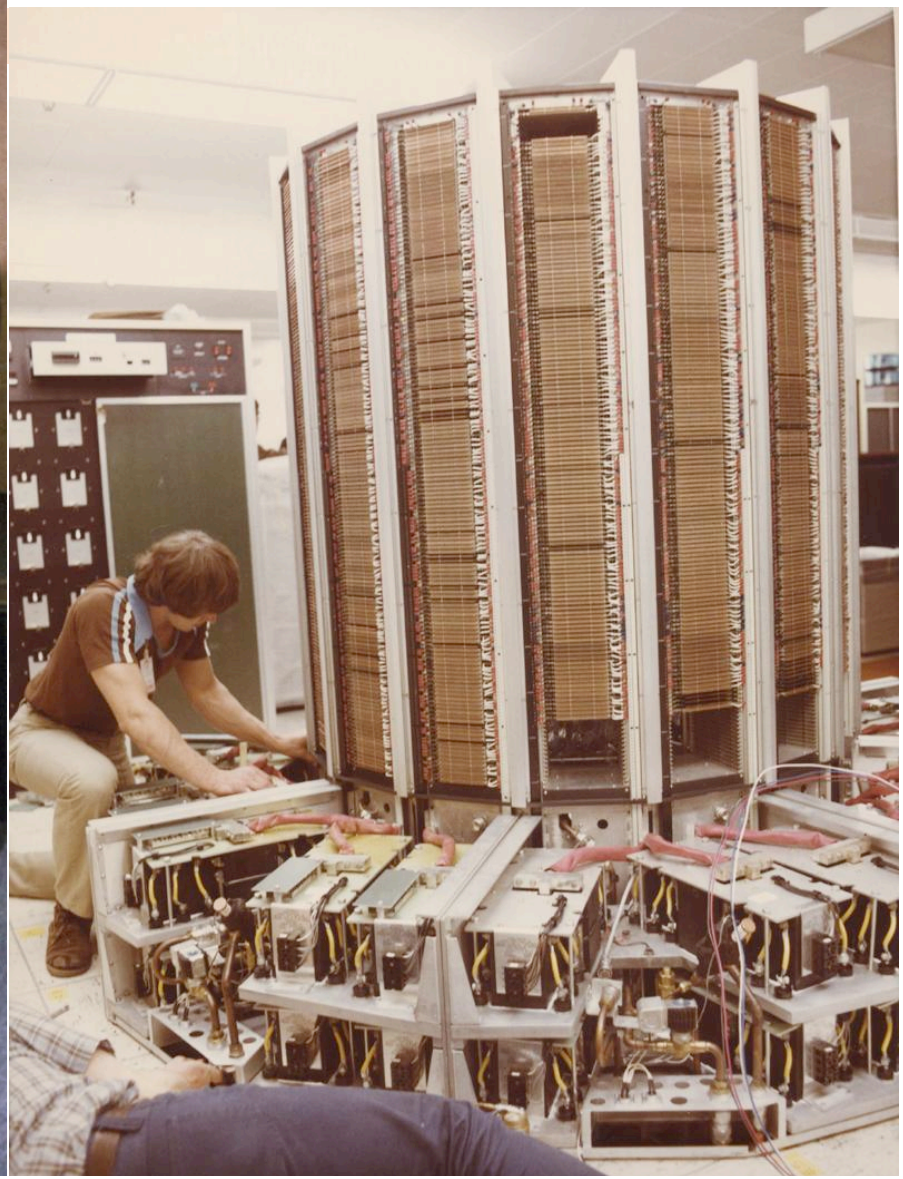


Vector Architectures

- Basic idea:
  - Read sets of data elements *(gather from memory)* into "vector registers"
  - Operate on those registers
  - Store/scatter the results back into memory

- Registers are controlled by the compiler
  - Used to hide memory latency
  - Leverage memory bandwidth

Cray-1 Supercomputer (1976)



Challenges

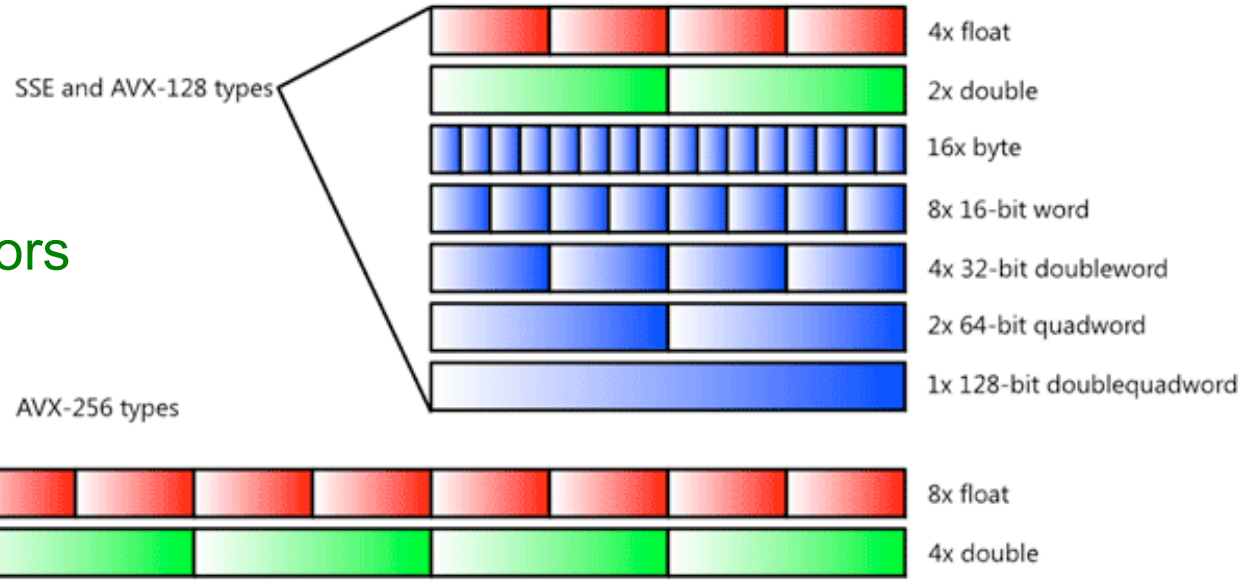
- Start up time
  - Latency of vector functional unit
  - Assume the same as Cray-1
    - Floating-point add => 6 clock cycles
    - Floating-point multiply => 7 clock cycles
    - Floating-point divide => 20 clock cycles
    - Vector load => 12 clock cycles
- Improvements:
  - > 1 element per clock cycle *(1)*
  - Non-64 wide vectors *(2)*
  - IF statements in vector code *(3)*
  - Memory system optimizations to support vector processors *(4)*
  - Multiple dimensional matrices *(5)*
  - Sparse matrices *(6)*
  - Programming a vector computer *(7)*

Vector Programming *(7)*

- Compilers are a key element to give hints on whether a code section will vectorize or not
- Check if loop iterations have data dependencies, otherwise vectorization is compromised
- Vector Architectures have a too high cost, but simpler variants are currently available on off-the-shelf devices; however:
  - most do not support non-unit stride => care must be taken in the design of data structures
  - same applies for gather-scatter...

SIMD Extensions

- Media applications operate on data types narrower than the native word size
  - Intel SIMD Ext started with 64-bit wide vectors and grew to wider vectors and more facilit
  - Current AVX generation is 512-bit wide

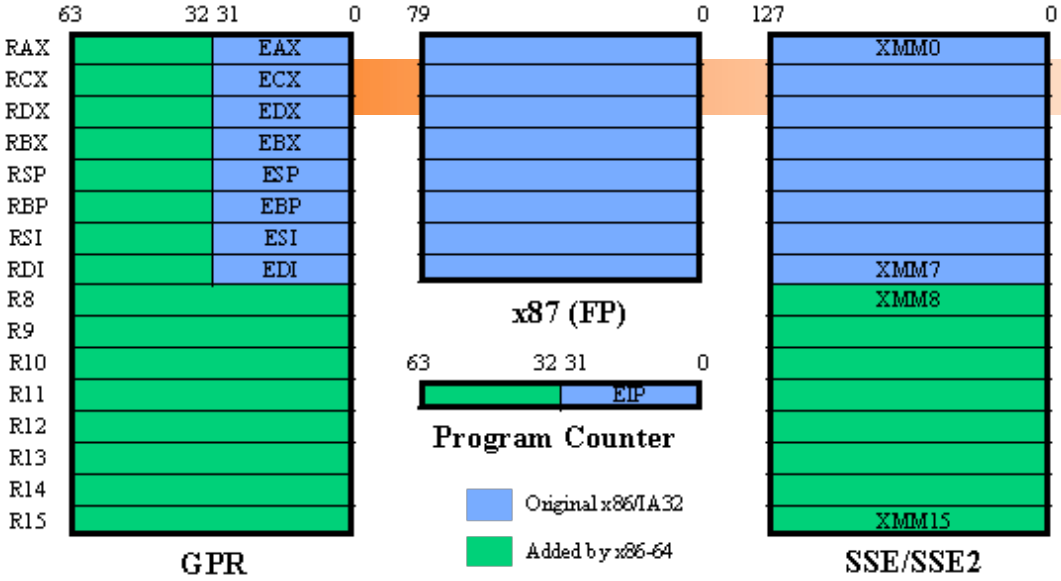


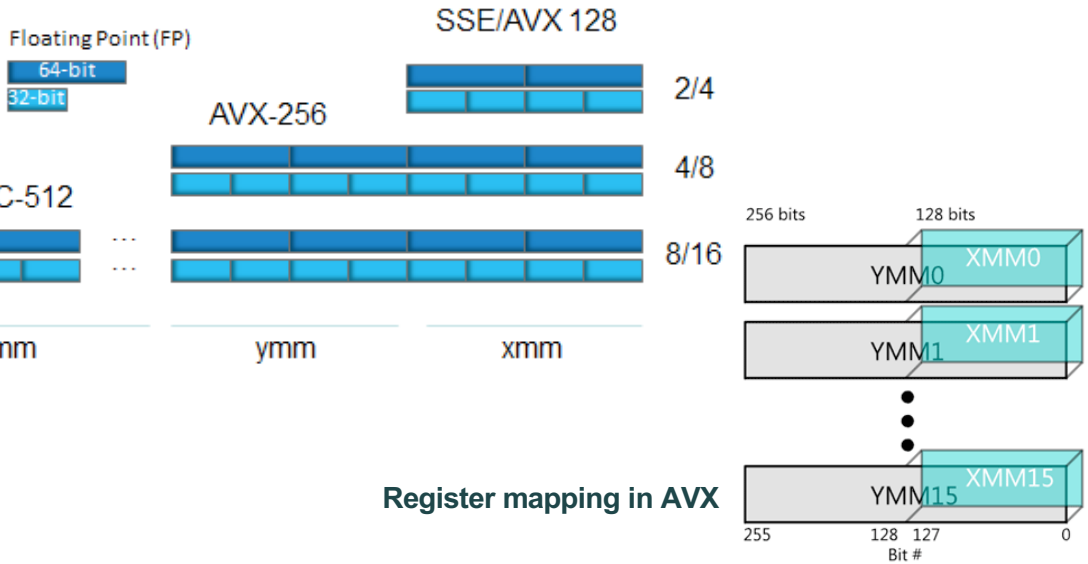
- Limitations, compared to vector architectures:
  - Number of data operands encoded into op code
  - No sophisticated addressing modes (strided, scatter-gather, but…)
  - No mask registers

SIMD Implementations

- Implementations:
  - Intel MMX (1996)
    - Eight 8-bit integer ops or four 16-bit integer ops
  - Streaming SIMD Extensions (SSE) (1999)
    - Eight 16-bit integer ops
    - Four 32-bit integer/fp ops or two 64-bit integer/fp ops
  - Advanced Vector eXtensions (AVX) (2010)
    - Eight 32-bit fp ops or Four 64-bit fp ops (integer in AVX-2)
    - 512-bits wide in AVX-512 (and also in Larrabee & Phi-KC)
  - Operands **must be in consecutive and aligned** memory locations

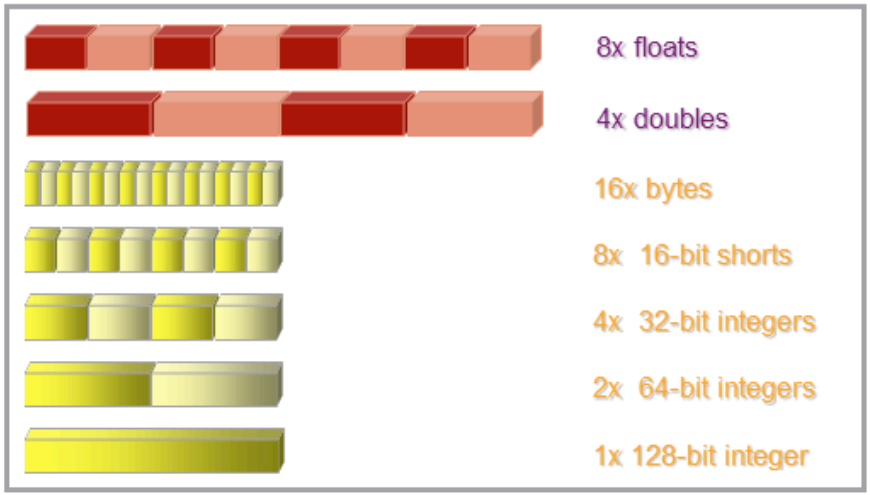
Extensões de processamento vetorial no Intel 64

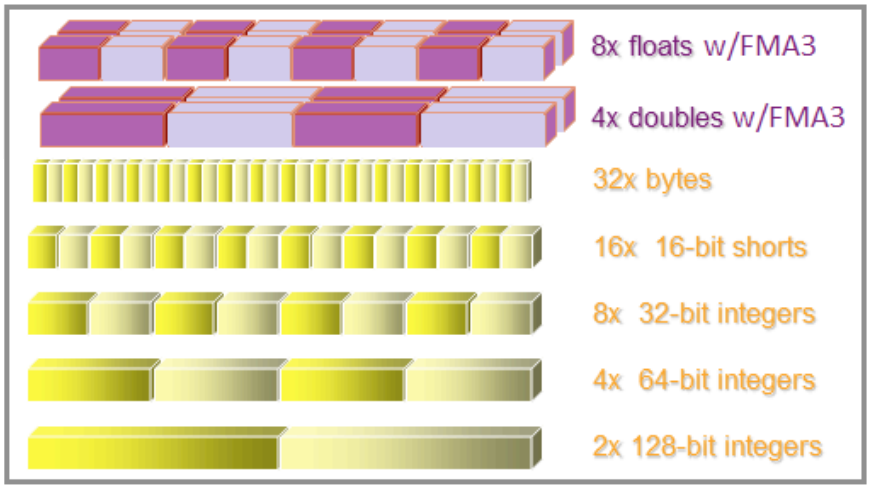




Register mapping in AVX

Advanced Vector eXtensions, AVX 1.0

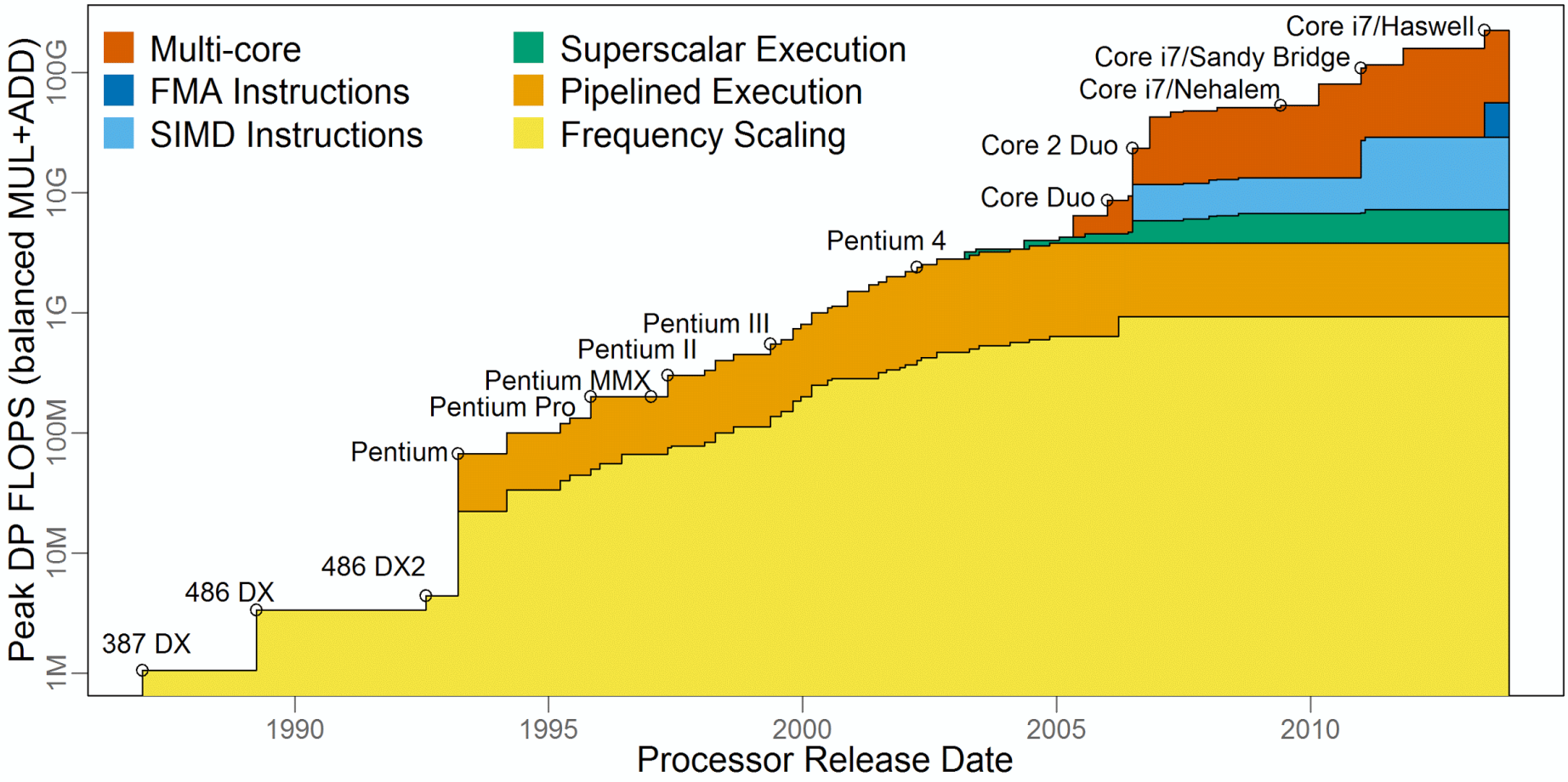




AVX 2.0

Next: AVX-512

Additional features in Intel x86



from Marat Dukhan, 2014

Beyond Vector/SIMD architectures

- Vector/SIMD-extended architectures are hybrid approaches
  - mix (super)**scalar + vector** op capabilities on a single device
  - **highly pipelined** approach to reduce memory access penalty
  - **tightly-closed access to shared memory**: lower latency
- Evolution of Vector/SIMD-extended architectures
  - **CPU cores with wider vectors and/or SIMD cores**:
    - DSP VLIW cores with vector capabilities: **Texas Instruments** (...?)
    - PPC cores coupled with SIMD cores: **Cell** (past...) , **IBM Power BQC...**
    - ARM64 cores coupled with SIMD cores: from Tegra to Parker (**NVidia**) (...?)
    - x86 many-core: **Intel** MIC / Xeon KNL, **AMD** FirePro**...**
    - other many-core: **ShenWay** 260, **Adapteva** Epiphany-V**...**
  - **coprocessors (require a host scalar processor): accelerator devices**
    - on disjoint physical memories (e.g., **Xeon KNC** with PCI-Expr, **PEZY-SC**)
    - focus on SIMT/SIMD to hide memory latency: **GPU**-type approach
    - ISA-free architectures, code compiled to silica: **FPGA**