Agner Fog, Technical University of Denmark

Optimizing software performance using vector instructions

Invited talk at Speed-B conference, October 19–21, 2016, Utrecht, The Netherlands.

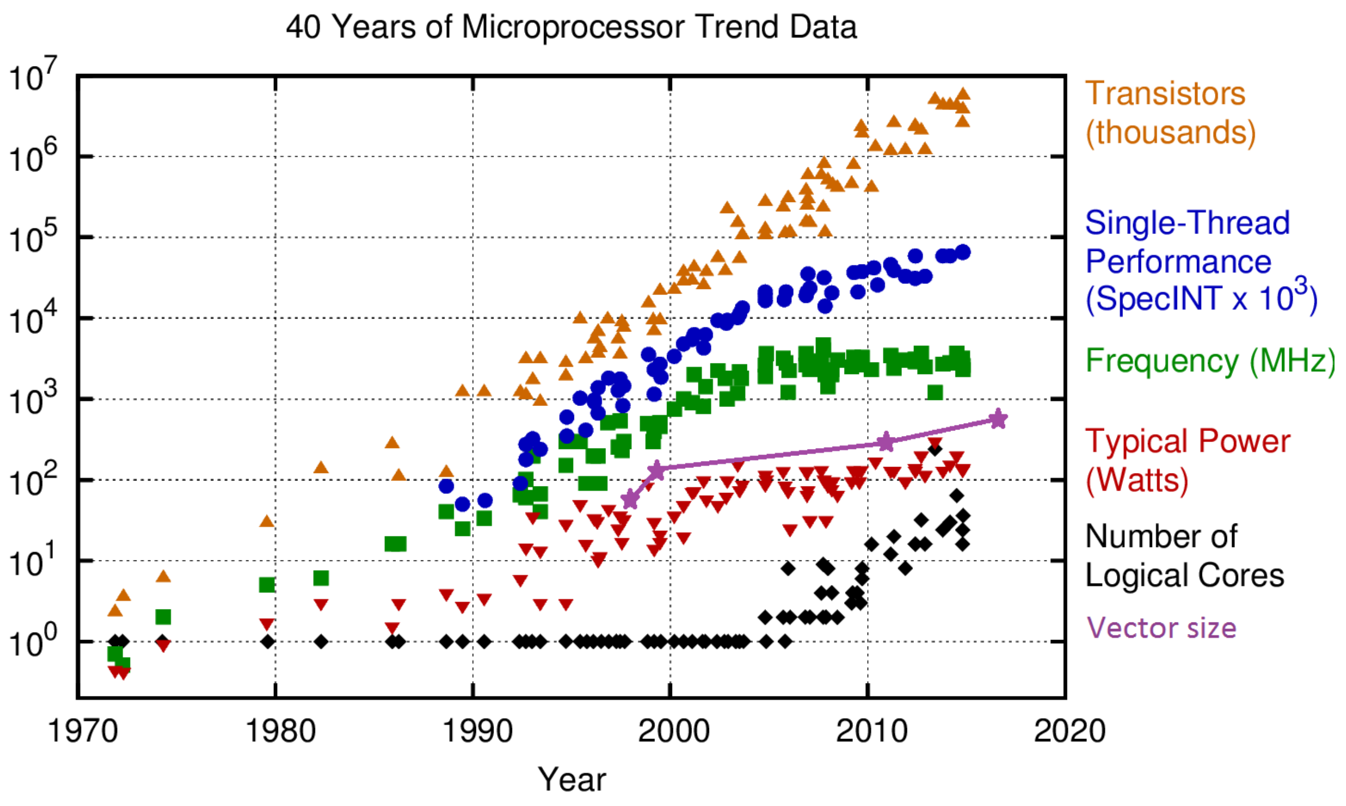
Abstract

Microprocessor factories have a problem obeying Moore's law because of physical limitations. The answer is increasing parallelism in the form of multiple CPU cores and vector instructions (Single Instruction Multiple Data - SIMD). This is a challenge to software developers who have to adapt to a moving target of new instruction set additions and increasing vector sizes. Most of the software industry is lagging several years behind the available hardware because of these problems. Other challenges are tasks that cannot easily be executed with vector instructions, such as sequential algorithms and lookup tables. The talk will discuss methods for overcoming these problems and utilize the continuously growing power of microprocessors on the market. A few problems relevant to cryptographic software will be covered, and the outlook for the future will be discussed.

Find more on these topics at author website:

www.agner.org/optimize

Moore's law



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp, 2016 by A. Fog

The clock frequency has stopped growing due to physical limitations. Instead, the number of CPU cores and the size of vector registers is growing.

Hierarchy of bottlenecks

- Program installation
- Program load, JIT compile, DLL's
- System database
- Network access
- File input/output
- Graphical user interface
- RAM access, cache utilization
- Algorithm
- Dependency chains
- CPU pipeline and execution units

Speed ↓

Remove the most limiting bottlenecks first. Find the hot spots.

Platforms

- x86
- ARM
- GPU
- Many-core processors

Programming language

- Wizards, point-and-click tools
- Java, C#, Visual Basic
- C/C++
- C/C++ using intrinsic functions
- Assembly language

↑ fast development — fast execution ↓

C++ compilers:

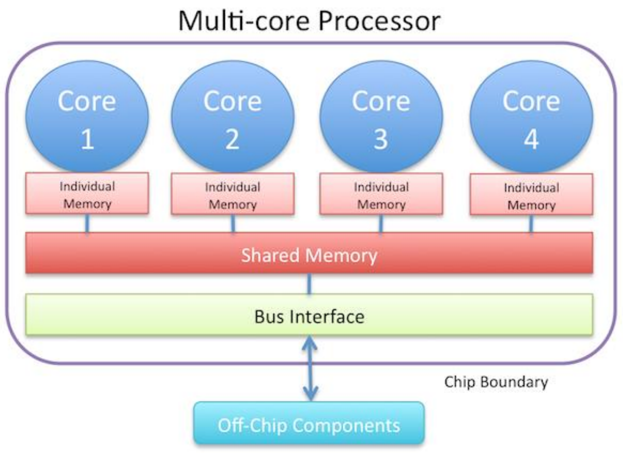
Gnu, Clang, Intel, PathScale, Microsoft

Memory allocation

- Data used together should be stored together
- Allocate few large blocks rather than many small
- Recycle allocated memory
- Avoid linked lists and STL containers
- Use local variables inside functions

Three parallelization methods

1. multiple cores

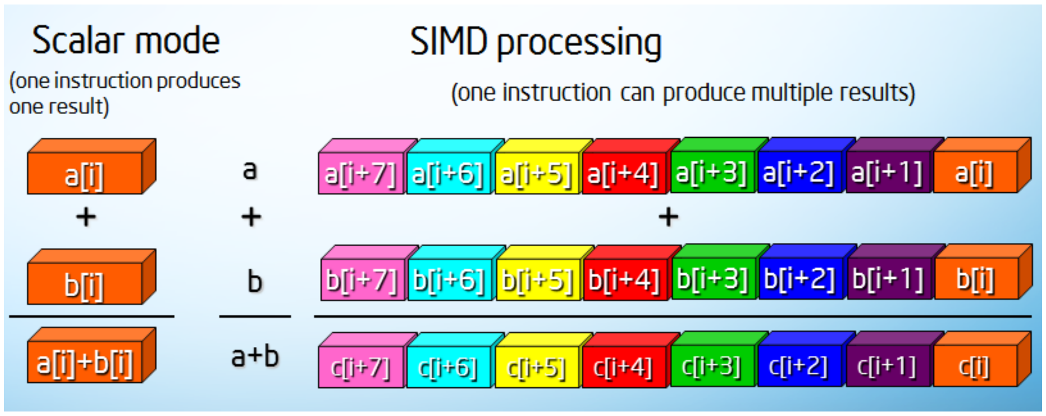


2. instruction level parallelism

```
R2 = R2 / R1
R4 = R3 * R1   (R3 delayed)
R1 = R1 + R4
```

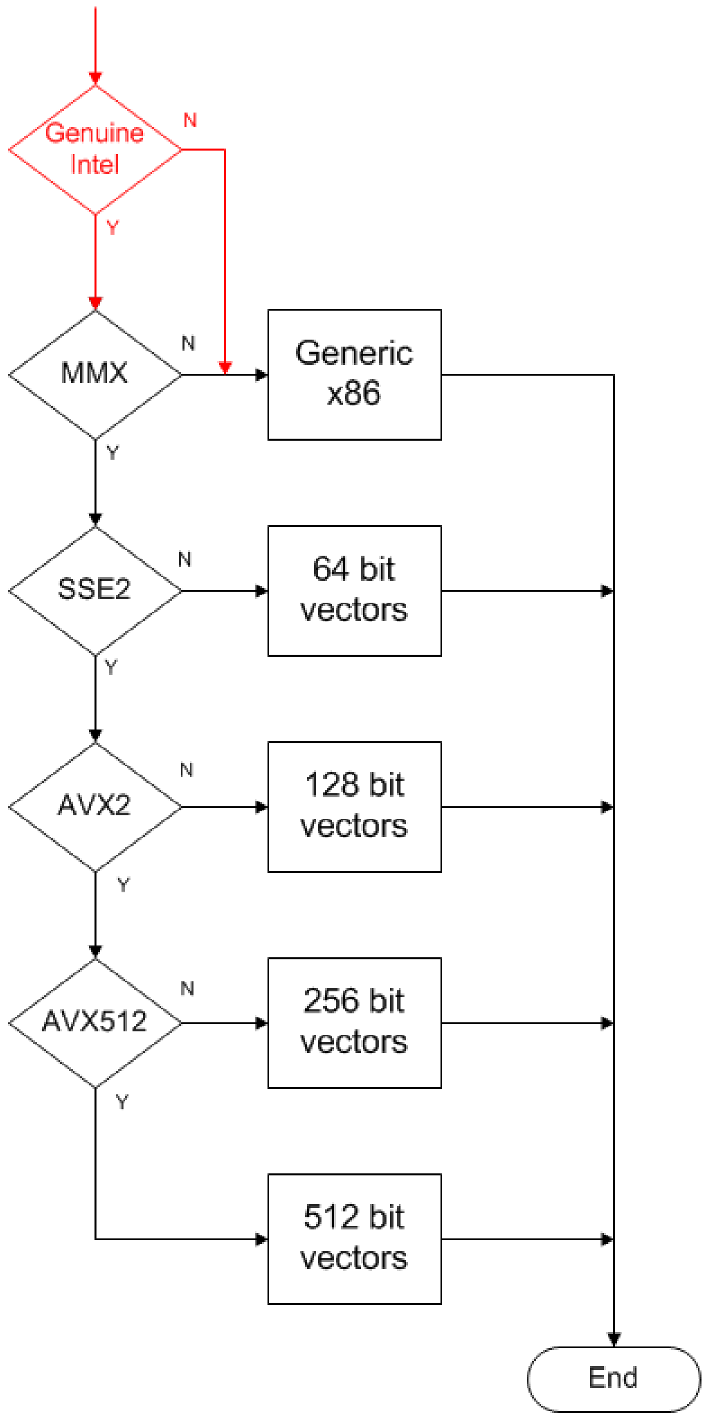
The lines can execute simultaneously or in any order if `R1` is renamed

3. vector instructions


Scalar mode
(one instruction produces one result)
SIMD processing
(one instruction can produce multiple results)
a[i]
+
b[i]
a[i]+b[i]
a
+
b
a+b
a[i+7] a[i+6] a[i+5] a[i+4] a[i+3] a[i+2] a[i+1] a[i]
+
b[i+7] b[i+6] b[i+5] b[i+4] b[i+3] b[i+2] b[i+1] b[i]
c[i+7] c[i+6] c[i+5] c[i+4] c[i+3] c[i+2] c[i+1] c[i]

Fine-grained versus coarse-grained parallelism

CPU dispatching



FTC: *"Intel sought to undercut the performance advantage of non-Intel x86 CPUs relative to Intel x86 CPUs when it redesigned and distributed software products, such as compilers and libraries".*
Link to discussion.

Common pitfalls for CPU dispatching:

- Coding for known present processors rather than future processors
- Failure to handle unknown processors properly
- Thinking in terms of specific processor models rather than processor features
- Making too many branches
- Underestimating the time lag between software development and use
- Underestimating the costs of developing, testing and maintaining multiple code versions
- Ignoring virtualization

Efficient dispatch method: Make a function pointer that is set to the appropriate version after first call.

Optimizations done by the compiler

- Function inlining
- Constant folding and constant propagation
- Register variables, live range analysis
- Common subexpression elimination
- Loop unrolling
- Loop invariant code motion
- Induction variables
- Instruction scheduling
- Algebraic reductions

Obstacles to optimization by compiler

- Cannot optimize across modules
- Pointer aliasing
- Pure functions
- Algebraic reduction of floating point

Optimizations done by the CPU

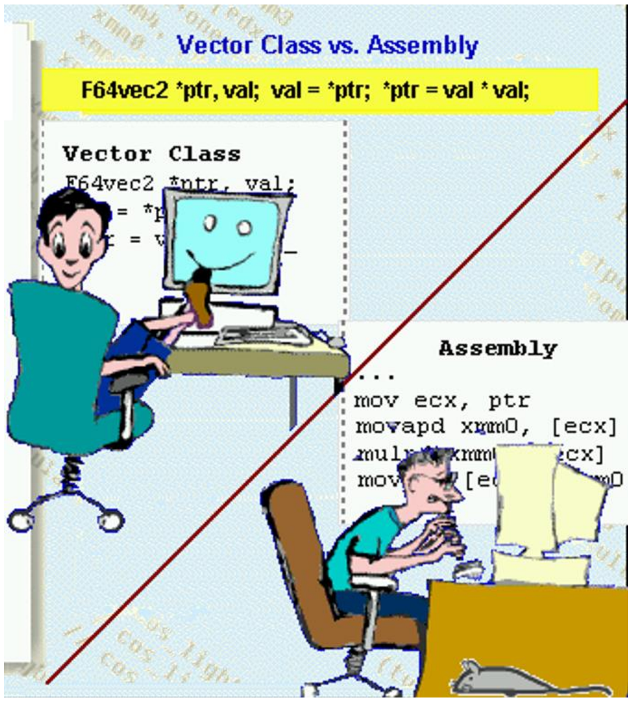
- Out of order execution
- Register renaming
- Branch prediction
- Data prefetching

Obstacles to optimization by the CPU

- Long dependency chains
- Loop-carried dependency chains
- Poorly predictable branches
- Memory allocation in small noncontiguous blocks

Vector coding methods

- Assembler
- Inline assembly
- Intrinsic functions
- Vector classes
- Automatic vectorization by compiler
- Use third party function library



```
F64vec2 *ptr, val;  val = *ptr;  *ptr = val * val;
```

```
...
mov ecx, ptr
movapd xmm0, [ecx]
```

Vector classes

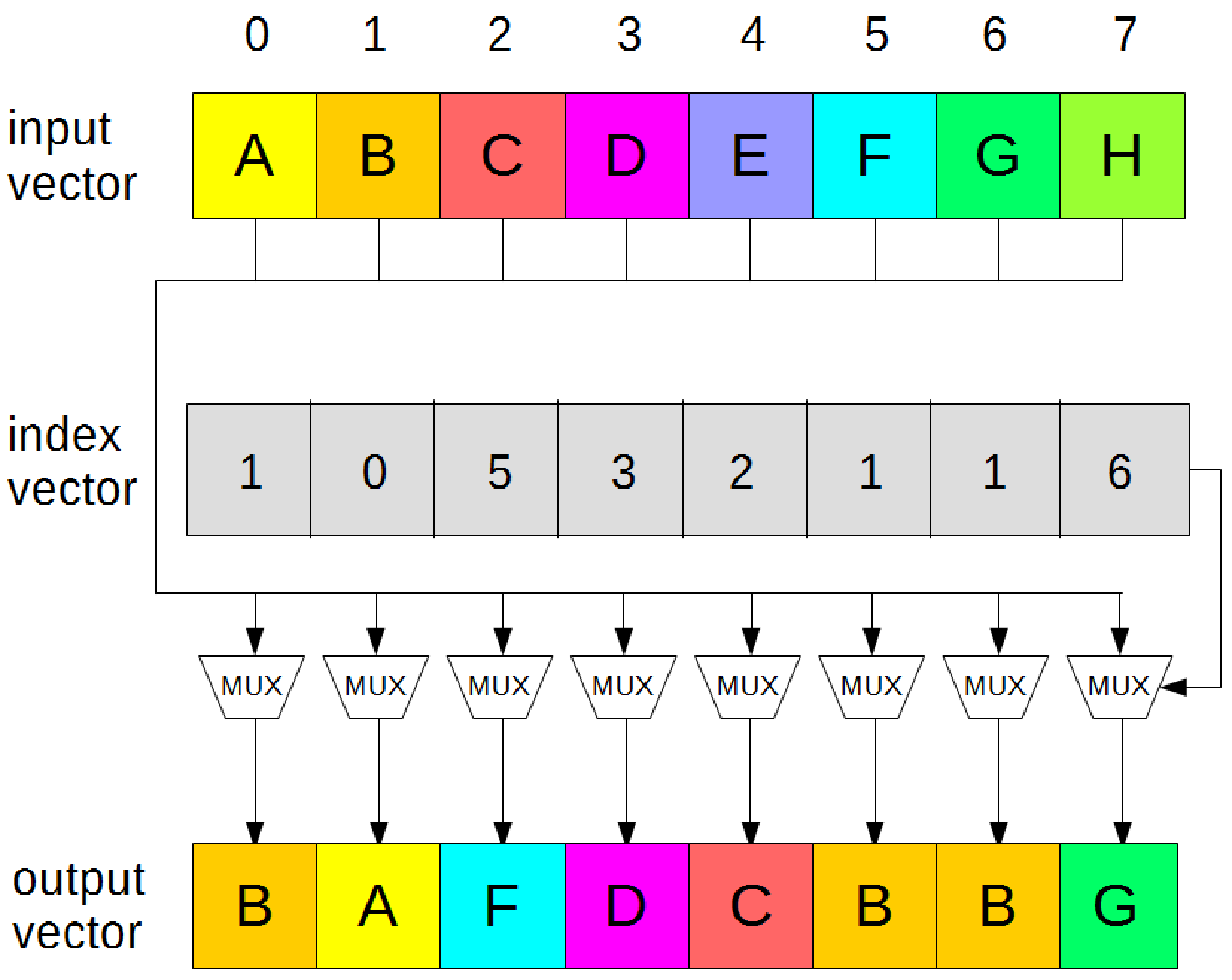
```
Vec16f a, b, c; // Declare vector objects

a = b + c;      // 16 parallel additions
```

[www. agner.org/optimize/#vectorclass](www.agner.org/optimize/#vectorclass)

Obstacles to vectorization

- sequential code
- pointer aliasing
- array size not divisible by vector size
- lookup tables

Use permute instructions for table lookup



A permute instruction can be used for parallel table lookup by putting the lookup table in the input vector. Some instructions have two input vectors, which doubles the size of the table.

Largest table size with 8-bit granularity is 16 elements (SSSE3, 32 with AMD XOP).

Largest table size with 16-bit granularity is 64 elements (AVX512BW)

Largest table size with 32-bit granularity is 32 elements (AVX512F)

Application-specific instructions in latest x86 processors

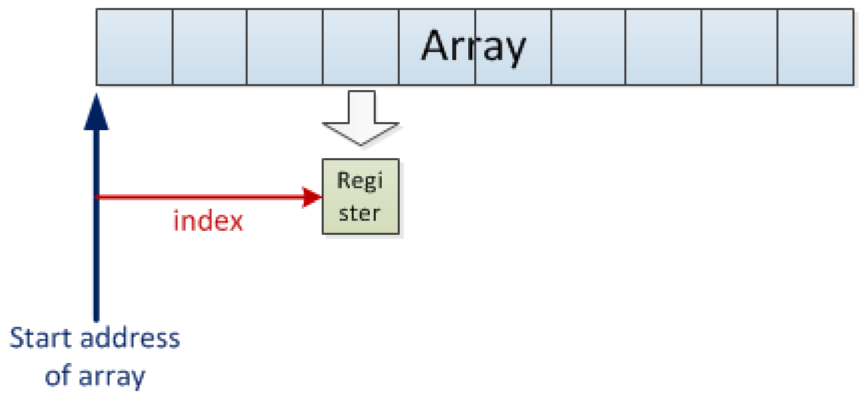
- AES instructions. 128 bit vectors
- CRC32. 32 bits
- SHA. Hashing 128 bit vectors
- RDRAND, RDSEED. Physical random generator, 64 bits

Possible future trends

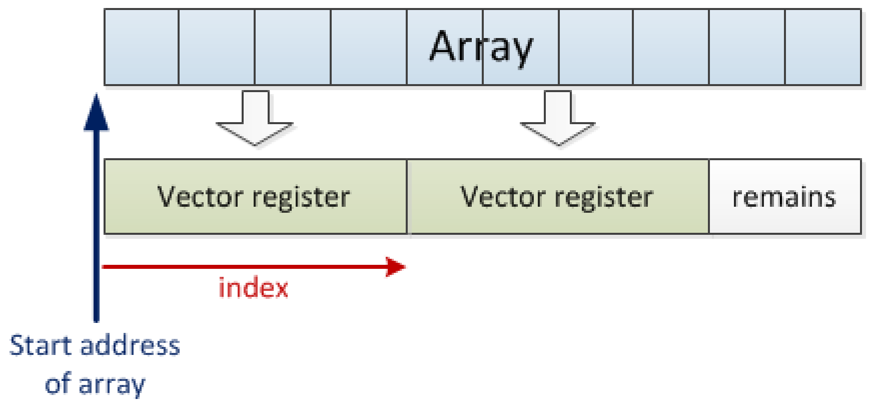
- More CPU cores
- Longer vector registers (1024, 2048 bits)
- More application-specific instructions
- Programmable logic (FPGA)

Proposal for instruction set that does not need CPU dispatching

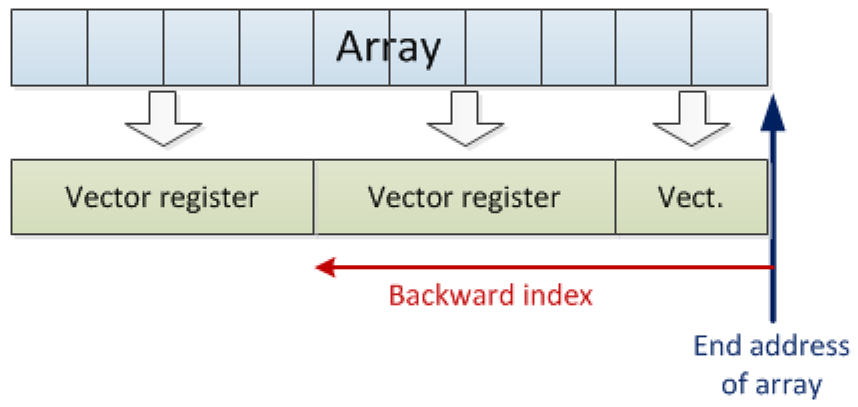
Simple loop:



Vector loop:



Loop with variable vector length



http://www.forwardcom.info.